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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. 0382 _____ Ильин Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта. За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить 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 и серийным номером пользователя. Полученные строки выводятся на экран. Отладьте полученный исходный модуль. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.
2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его. Таким образом, будет получен «хороший» .EXE.

3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».
4. Запустить FAR и открыть файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».
5. Открыть отладчик TD.EXE и загрузить CO. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.
6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».
7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

За основу был взят шаблон .COM модуля из методического пособия, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. Для определения типа PC и версии системы были написаны процедуры: IBM_TYPE, DOS_VER, OEMN, USERN. Тип IBM PC был получен согласно байту по адресу 0F000:0FFFFh. Для определения версии системы же использовалась функция 30H прерывания 21H. Ее выходными параметрами являются: AL – номер основной версии, AH – номер модификации, BH – серийный номер OEM, BL:CX – 24-битовый серийный номер пользователя.

В результате шага имеем “хороший” .COM модуль и “плохой” .EXE модуль. Выводы, полученные при их запуске, представлены на рисунке 1 и рисунке 2 соответственно.

Рисунок 1 – результат запуска модуля lb1_com.com

Рисунок 2 – результат запуска модуля lb1_com.exe

Для написания “хорошего” .EXE модуля разобьем программу на сегменты кода, данных и стека, также добавим главную процедуру. В .COM модуле имелась директива org 100h, что нужна, поскольку при загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается лишь после этого блока. В .EXE модуле же мы в этом не нуждаемся, поскольку блок PSP расположен вне сегмента кода. На рисунке 3 представлены результаты запуска данного модуля.

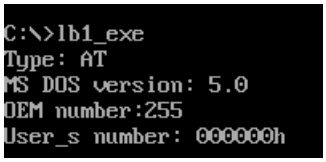

Рисунок 3 – результат запуска модуля lb1_exe.exe

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?
   Один сегмент.
2. EXE-программа?
   Может содержать от одного до четырёх сегментов.
3. Какие директивы должны быть обязательно в тексте COM-программы?
   ORG 100h (для смещение относительно нулевого адреса на 256 байт – место для PSP). ASSUME CS:SEG, DS:SEG, SS:NOTHING, ES:NOTHIG чтобы связать сегмент с сегментными регистрами.

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

   Нет. Так как в .com файле отсутствует relocation table, команды вида mov *register*, *segment_name* не поддерживаются.

Отличия форматов файлов .com и .exe модулей:

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

   .COM файл (см. рисунок 4) состоит из единственного сегмента, началом которого инициализируются все сегментные регистры. Код располагается с адреса 0h, однако при загрузке программы в память в начало этого сегмента будет добавлен PSP размером 100h байт, поэтому выставляется смещение 100h.

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

   «Плохой» .exe (см. рисунок 5-6) файл состоит из единственного сегмента. Код располагается с адреса 300h (512 байт с адреса 0 – заголовок и relocation table, 256 байт – смещение).

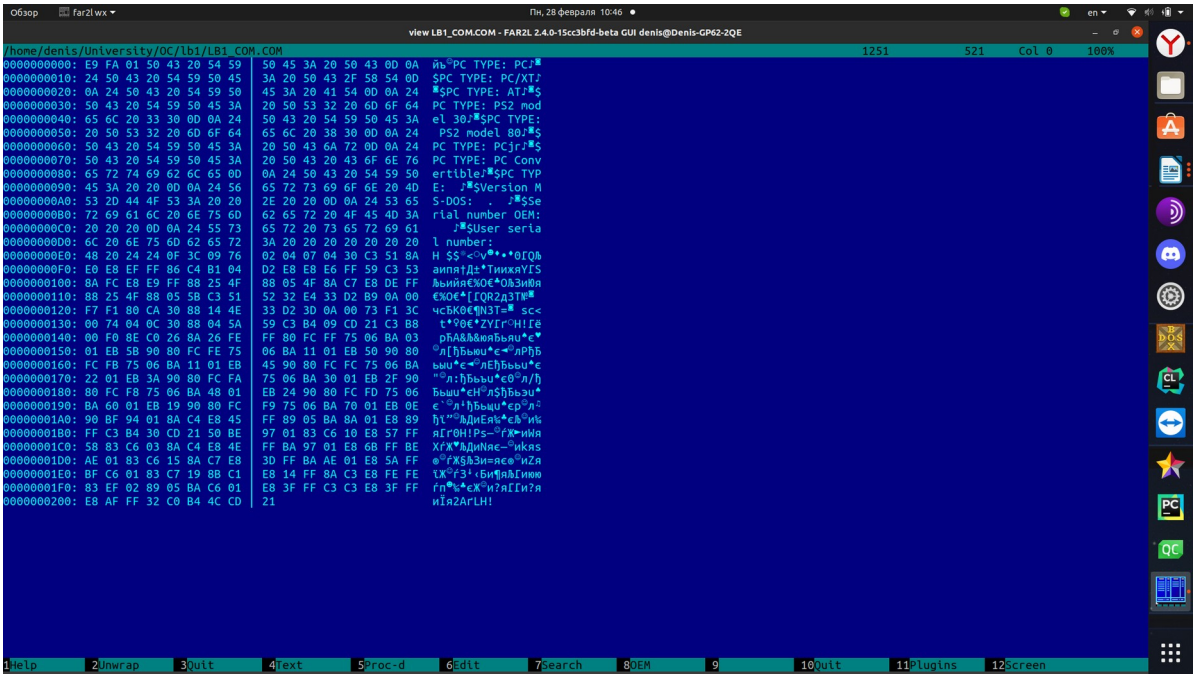

Рисунок 4 – .com модуль в бинарном виде

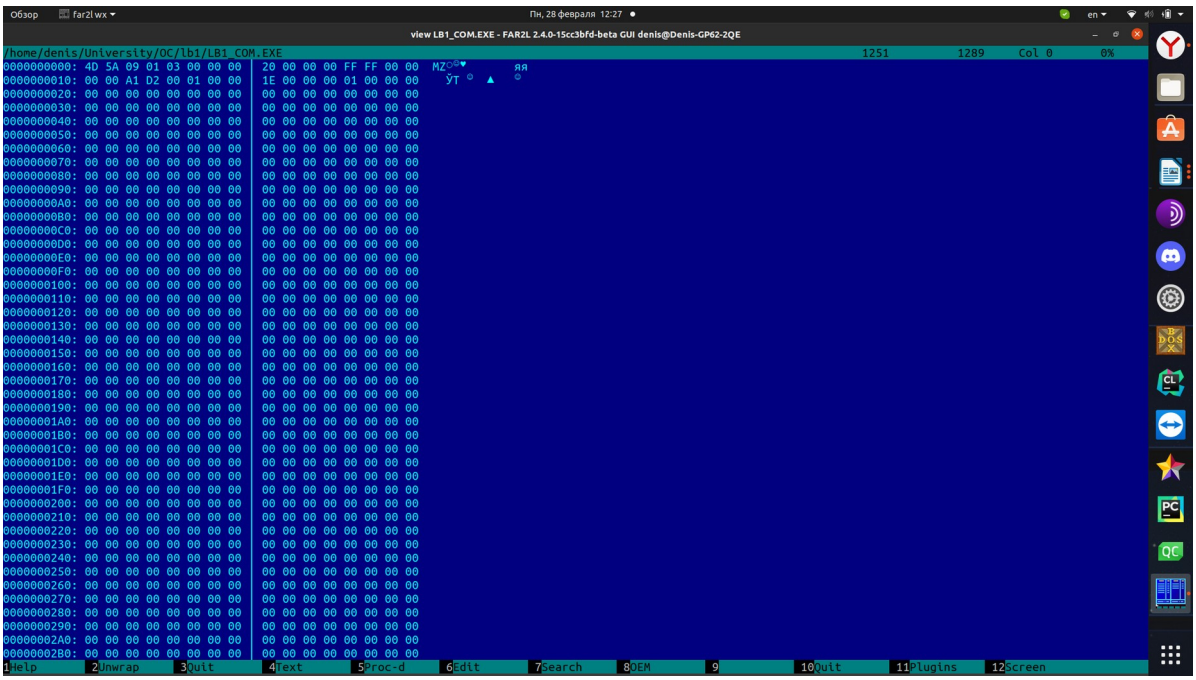

Рисунок 5 – Плохой .exe модуль в бинарном виде, часть 1

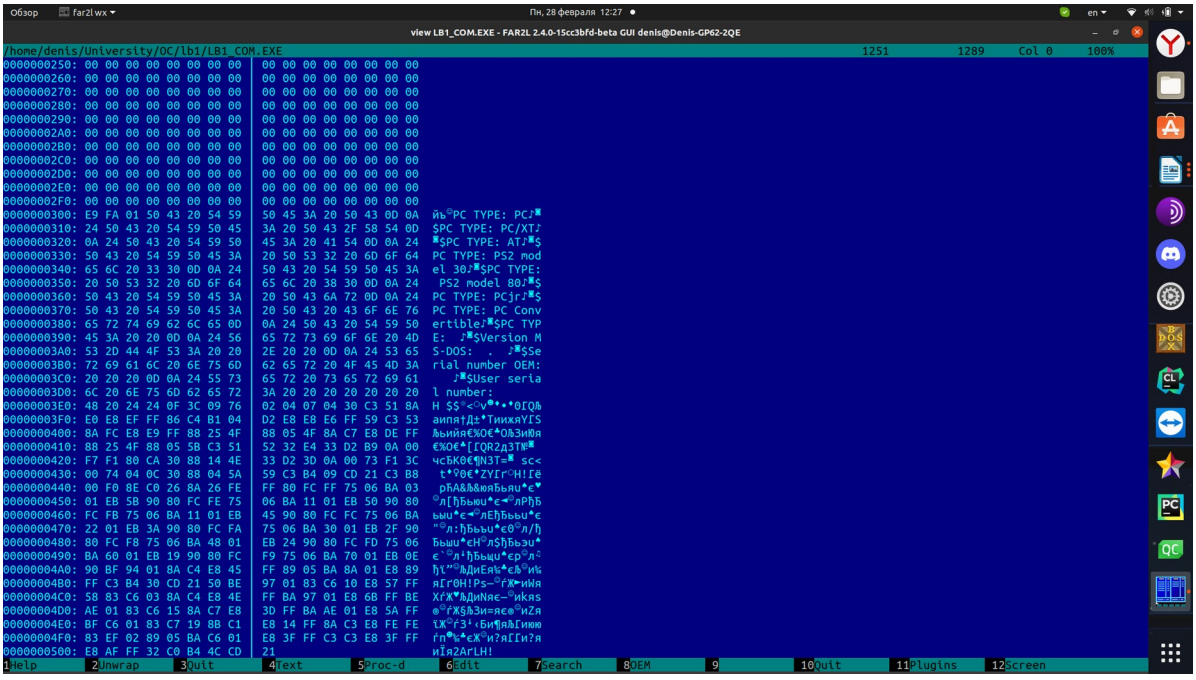

Рисунок 6 – Плохой .exe модуль в бинарном виде, часть 2

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

   В хорошем .exe модуле (см. рисунок 7-8) программа имеет несколько сегментов. В начале модуля расположен заголовок и relocation table (512

байт), далее расположены сегменты в том порядке, в котором они определены в коде. Сначала сегмент стэка – 512 байт, далее сегмент данных, после него – сегмент кода.

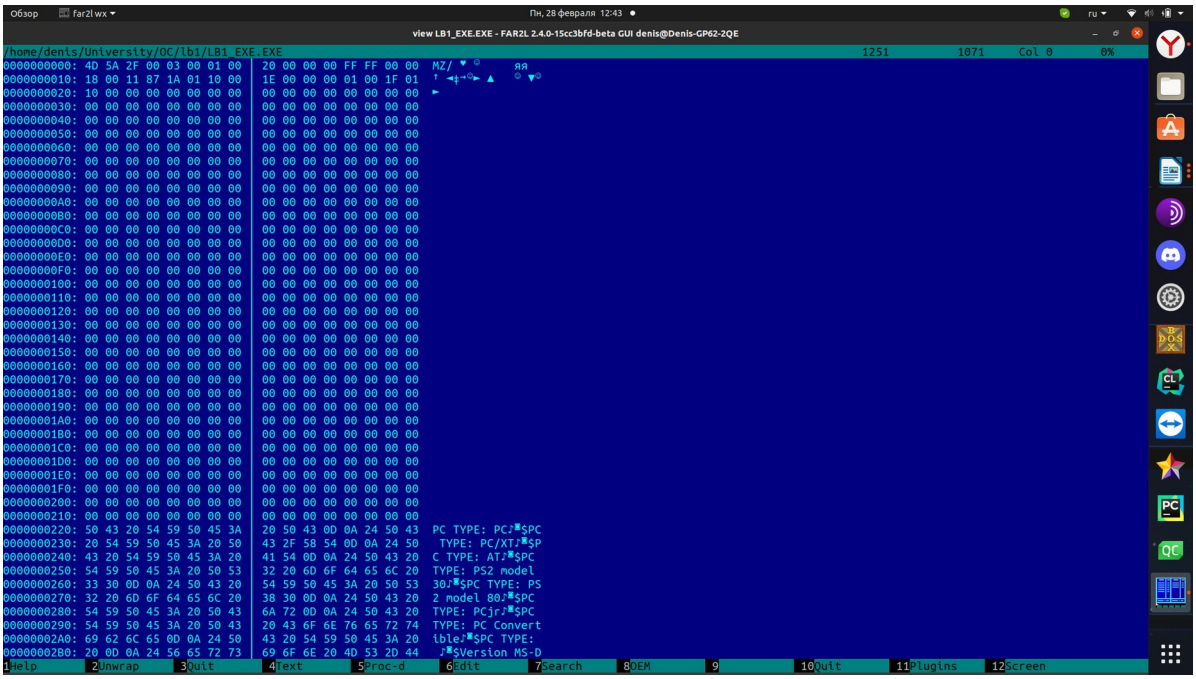

Рисунок 7 – Хороший .exe модуль, часть 1

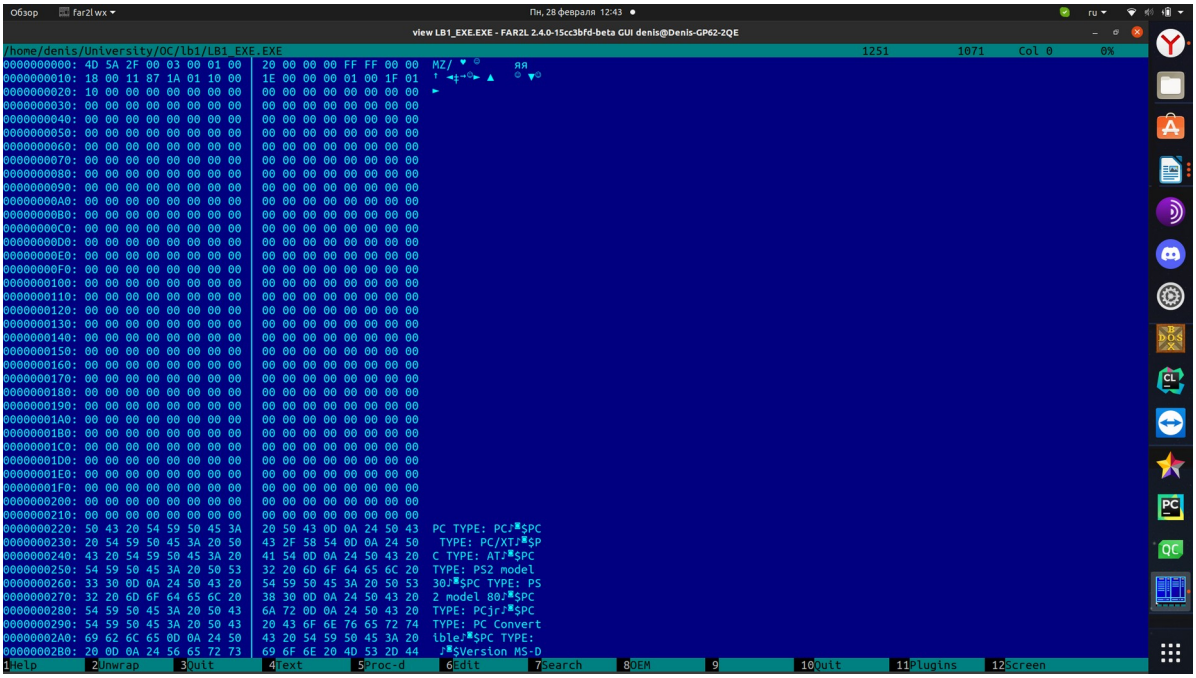

Рисунок 8 – Хороший .exe модуль, часть 2

Загрузка .com модуля в основную память:

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

   Загрузка .com модуля в память происходит следующим образом: в основной памяти выделяется свободный сегмент, в первых 256 байтах этого сегмента генерируется PSP, далее записывается сама программа. Код располагается с адреса CS:0100 = 48DD:0100 (см. рисунок 9).

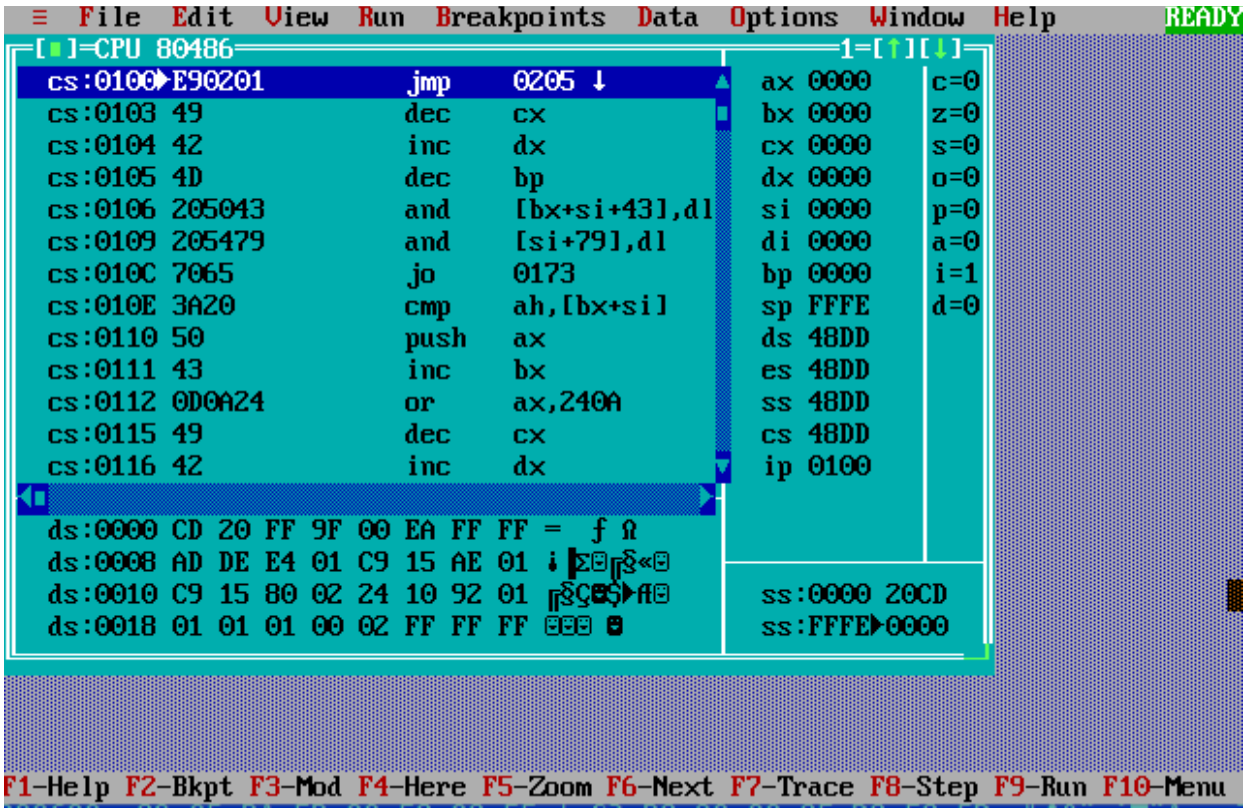

Рисунок 9 – Начальное состояние загруженного .com модуля

2. Что располагается с адреса 0?

   PSP.

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

   Все сегментные регистры указывают на начало PSP. Из значение равны 48DD (см. рисунок 9).

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

Под стэк отведен весь сегмент, в который загружена программа. SS=48DD указывает на начало сегмента SP=FFFE указывает на последний адрес сегмента кратный двум. Адреса: 48DD:0000 – 48DD:FFFE (см. рисунок 9).

Загрузка «хорошего» .exe модуля в основную память:

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

   Сначала в память загружается PSP, после которого загружается .exe модуль в соответствии с информацией в заголовке. Значение регистров см. на рисунке 9.
2. На что указывают DS и ES?

   На начало PSP.
3. Как определяется стэк?

   Стэк определяется при помощи описание стэкового сегмента в коде и директивы ASSUME. SS указывает на начало сегмента стэка, а SP – на конец стэка (на рисунке 8 видно, что SP = 0100h так как размер стэка – 256 байт).
4. Как определяется точка входа?

   Точка входа определяется при помощи директивы END.

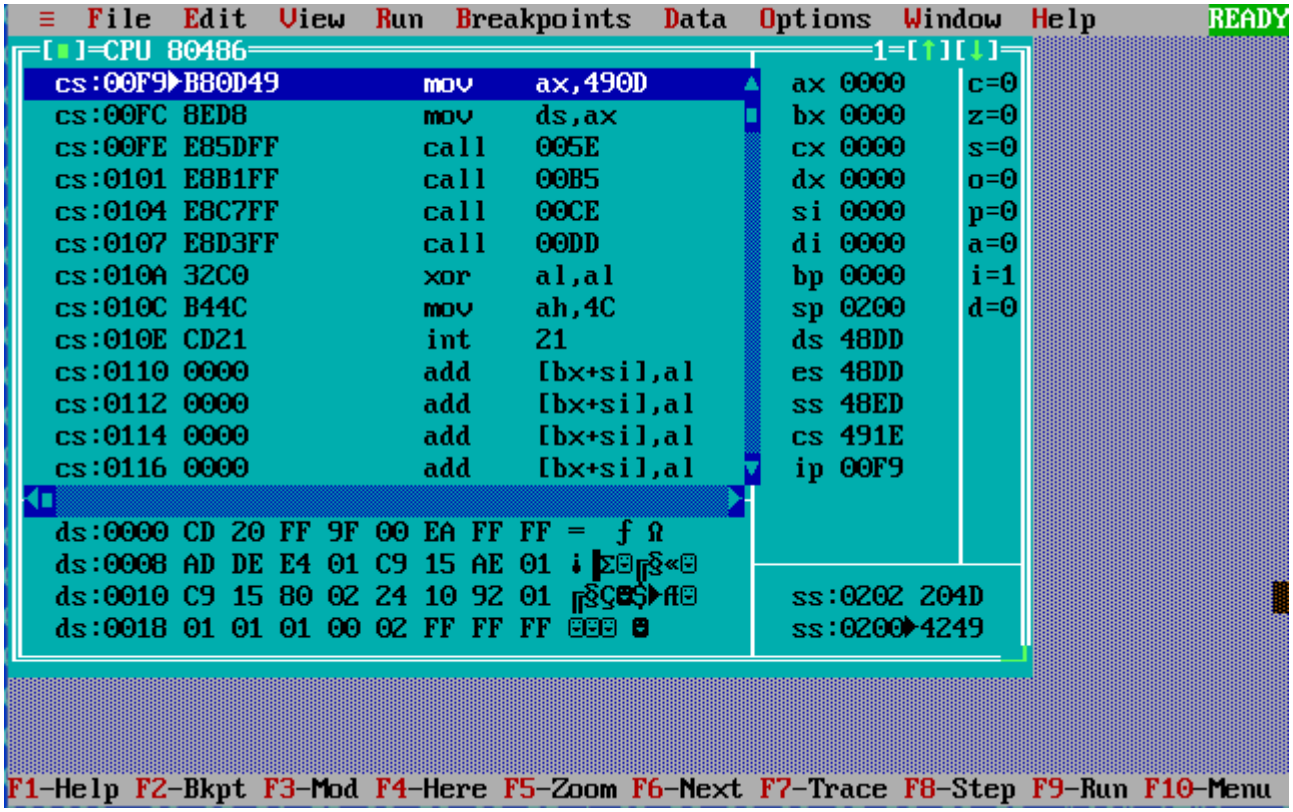

Рисунок 8 – Начальное состояние загруженного .exe модуля

Выводы.

В ходе работы было изучено то, как устроены файлы для COM и EXE исполняемых модулей, а также сами загрузочные модули, их сходства и различия, алгоритм загрузки в память. Написана и протестирована программа, которая выводит тип ПК, версию ОС DOS, OEM номер и номер пользователя.

ПРИЛОЖЕНИЕ А.

Исходный код модулей

lb1_com.asm:

```
; Шаблон текста программы на ассемблере для модуля типа .COM
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H

START: JMP BEGIN

; ДАННЫЕ
PC db 'PC TYPE: PC',0DH,0AH,'$'
PC_XT db 'PC TYPE: PC/XT',0DH,0AH,'$'
AT db 'PC TYPE: AT',0DH,0AH,'$'
PS2_30 db 'PC TYPE: PS2 model 30',0DH,0AH,'$'
;PS2_50_60 db 'PC TYPE: PS2 model 50 or 60',0DH,0AH,'$'
PS2_80 db 'PC TYPE: PS2 model 80',0DH,0AH,'$'
PCjr db 'PC TYPE: PCjr',0DH,0AH,'$'
PC_Convertible db 'PC TYPE: PC Convertible',0DH,0AH,'$'
PC_Unknown db 'PC TYPE:  ',0DH,0AH,'$'

VERSIONS db 'Version MS-DOS:  .  ',0DH,0AH,'$'
SERIAL_NUMBER db  'Serial number OEM:   ',0DH,0AH,'$'
USER_NUMBER db  'User serial number:       H $'


;ПРОЦЕДУРЫ
;--------------------------------------------------

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
; 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,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 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 10с/с, 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
;--------------------------------

print_dx PROC near
    mov AH,09h
    int 21h
    ret
print_dx ENDP

PRINT_PC_TYPE PROC near
    mov ax, 0F000h
    mov es, ax
    mov ah, es:[0FFFEh]

pc_l:
    cmp ah, 0FFh
    jne pc_xt_l_1
    mov dx, offset PC
    JMP final

pc_xt_l_1:
    cmp ah, 0FEh
```

```
    jne pc_xt_1_2
    mov dx, offset PC_XT
    JMP final

pc_xt_1_2:
    cmp ah, 0FBh
    jne at_1
    mov dx, offset PC_XT
    JMP final

at_1:
    cmp ah, 0FCh
    jne ps2_30_1
    mov dx, offset AT
    JMP final

ps2_30_1:
    cmp ah, 0FAh
    jne ps2_80_1
    mov dx, offset PS2_30
    JMP final

ps2_80_1:
    cmp ah, 0F8h
    jne pcjr_1
    mov dx, offset PS2_80
    JMP final

pcjr_1:
    cmp ah, 0FDh
    jne pc_convertible_1
    mov dx, offset PCjr
    JMP final

pc_convertible_1:
    cmp ah, 0F9h
    jne pc_unknown_1
    mov dx, offset PC_Convertible
    JMP final

pc_unknown_1:
    mov di, offset PC_Unknown + 10
    mov al, ah
    call BYTE_TO_HEX
    mov [di], AX
    mov dx, offset PC_Unknown

final:
    call print_dx
    ret
PRINT_PC_TYPE ENDP

PRINT_VER_OS PROC near
   mov ah,30h
   int 21h
   push ax

   mov si,offset VERSIONS
   add si,16
   call BYTE_TO_DEC
```

```
   pop ax
   add si,3
   mov al,ah
   call BYTE_TO_DEC
   mov dx,offset VERSIONS
   call print_dx

   mov si,offset SERIAL_NUMBER
   add si,21
   mov al,bh
   call BYTE_TO_DEC
   mov dx ,offset SERIAL_NUMBER
   call print_dx

   mov di, offset USER_NUMBER
     add di, 25
     mov ax, cx
     call WRD_TO_HEX
     mov al, bl
     call BYTE_TO_HEX
     sub di,2
     mov [di], ax
     mov dx, offset USER_NUMBER
     call print_dx
     ret

final_2:
   ret
PRINT_VER_OS ENDP

; КОД
BEGIN:
    call PRINT_PC_TYPE
    call PRINT_VER_OS
; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START ;конец модуля, START - точка входа
```

lb1_exe.asm:

```
AStack SEGMENT STACK
    DW 12 DUP(?)
AStack ENDS

DATA SEGMENT

; ДАННЫЕ
PC db 'PC TYPE: PC',0DH,0AH,'$'
PC_XT db 'PC TYPE: PC/XT',0DH,0AH,'$'
AT db 'PC TYPE: AT',0DH,0AH,'$'
PS2_30 db 'PC TYPE: PS2 model 30',0DH,0AH,'$'
;PS2_50_60 db 'PC TYPE: PS2 model 50 or 60',0DH,0AH,'$'
PS2_80 db 'PC TYPE: PS2 model 80',0DH,0AH,'$'
PCjr db 'PC TYPE: PCjr',0DH,0AH,'$'
PC_Convertible db 'PC TYPE: PC Convertible',0DH,0AH,'$'
PC_Unknown db 'PC TYPE:  ',0DH,0AH,'$'
```

```
VERSIONS db 'Version MS-DOS:  .  ',0DH,0AH,'$'
SERIAL_NUMBER db  'Serial number OEM:   ',0DH,0AH,'$'
USER_NUMBER db  'User serial number:       H $'

DATA ENDS


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

;ПРОЦЕДУРЫ
;-------------------------------------------------

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
; 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,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 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 10с/с, 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
;--------------------------------

print_dx PROC near
    mov AH,09h
    int 21h
    ret
print_dx ENDP

PRINT_PC_TYPE PROC near
    mov ax, 0F000h
    mov es, ax
    mov ah, es:[0FFFEh]

pc_l:
    cmp ah, 0FFh
    jne pc_xt_l_1
    mov dx, offset PC
    JMP final

pc_xt_l_1:
    cmp ah, 0FEh
    jne pc_xt_l_2
    mov dx, offset PC_XT
    JMP final

pc_xt_l_2:
    cmp ah, 0FBh
    jne at_l
    mov dx, offset PC_XT
    JMP final

at_l:
    cmp ah, 0FCh
    jne ps2_30_l
    mov dx, offset AT
    JMP final

ps2_30_l:
    cmp ah, 0FAh
    jne ps2_80_l
```

```
    mov dx, offset PS2_30
    JMP final

ps2_80_l:
    cmp ah, 0F8h
    jne pcjr_l
    mov dx, offset PS2_80
    JMP final

pcjr_l:
    cmp ah, 0FDh
    jne pc_convertible_l
    mov dx, offset PCjr
    JMP final

pc_convertible_l:
    cmp ah, 0F9h
    jne pc_unknown_l
    mov dx, offset PC_Convertible
    JMP final

pc_unknown_l:
    mov di, offset PC_Unknown + 9
    mov al, ah
    call BYTE_TO_HEX
    mov [di], AX
    mov dx, offset PC_Unknown

final:
    call print_dx
    ret
PRINT_PC_TYPE ENDP

PRINT_VER_OS PROC near
    mov ah,30h
    int 21h
    push ax

    mov si,offset VERSIONS
    add si,16
    call BYTE_TO_DEC
    pop ax
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    mov dx,offset VERSIONS
    call print_dx

    mov si,offset SERIAL_NUMBER
    add si,21
    mov al,bh
    call BYTE_TO_DEC
    mov dx ,offset SERIAL_NUMBER
    call print_dx

    mov di, offset USER_NUMBER
     add di, 25
     mov ax, cx
     call WRD_TO_HEX
     mov al, bl
```

```
      call BYTE_TO_HEX
      sub di,2
      mov [di], ax
      mov dx, offset USER_NUMBER
      call print_dx
      ret

final_2:
   ret
PRINT_VER_OS ENDP

; КОД
main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX
    call PRINT_PC_TYPE
    call PRINT_VER_OS
; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
main ENDP
CODE ENDS
    END main ;конец модуля, START - точка входа
```