**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6381 |  | Лопатина А.С. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2018

**Постановка задачи**

**Цель работы.**

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

**Необходимые сведения для составления программы.**

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

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;

**Сведения о функциях и структурах данных управляющей программы.**

Сведения о функциях:

|  |  |
| --- | --- |
| Название | Назначение |
| TETR\_TO\_HEX | перевод полвины байта из 2-ой СС в 16-ую СС |
| BYTE\_TO\_HEX | перевод байта, расположенного в AL, в 2 символа 16-ой числа в AX |
| WRD\_TO\_HEX | перевод в 16-ую СС 16-ти разрядного числа |
| BYTE\_TO\_DEC | перевод в 10-ую СС |
| OUTPUT\_PROC | вывод на экран |
| DET\_TYPE | получение типа РС |
| DET\_VERSION | получение версии системы |
| DET\_OEM\_NUM | получение серийного номера ОЕМ |
| Название | Назначение |
| DET\_USER\_NUM | получение серийного номера пользователя |

Сведения о структурах данных:

|  |  |  |
| --- | --- | --- |
| Название | Тип | Назначение |
| TYPE\_PC | db | Тип PC |
| VERSION | db | Версия системы |
| OEM\_NUM | db | Серийный номер OEM |
| USER\_NUM | db | Серийный номер пользователя |
| PC | db | PC |
| PC\_XT | db | PC/XT |
| AT | db | AT |
| PC2\_30 | db | PC2 модель 30 |
| PC2\_50 | db | PC2 модель 50 |
| PC2\_80 | db | PC2 модель 80 |
| PCjr | db | PCjr |
| PC\_Conv | db | PC Convertible |

**Последовательность действий, выполняемых программой.**

Программа определяет тип PC, версию системы, серийный номер OEM и серийный номер пользователя, затем сохраняет их в соответствующие переменные выводит полученные значения на экран.

**Результат работы программы.**

1. «Хороший» .COM модуль

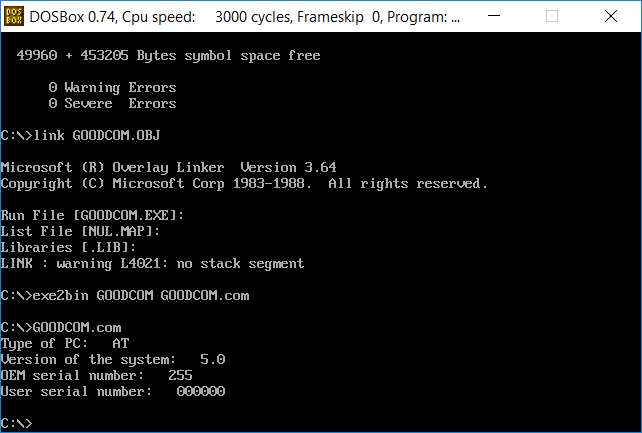


рис.1. «Хороший» .COM модуль

1. «Плохой» .EXE модуль

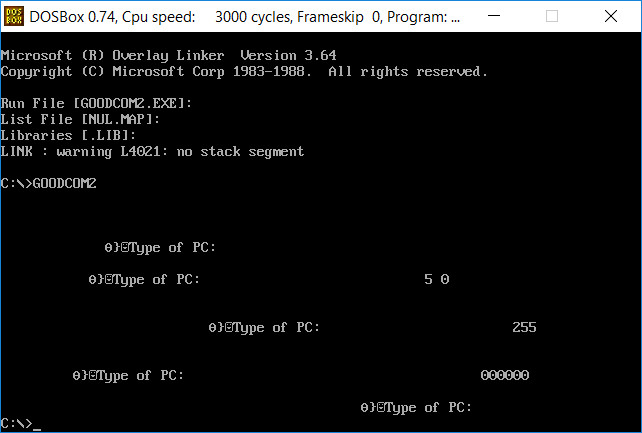


рис.2. «Плохой» .EXE модуль

1. «Хороший» .EXE модуль

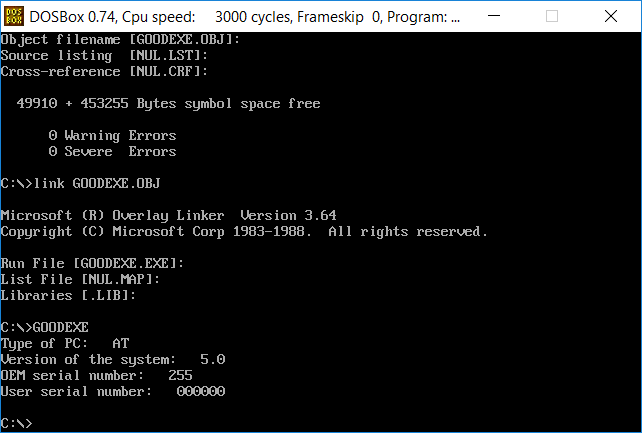


рис.3. «Хороший» .EXE модуль

**Описание результатов исследования проблем, поставленных в лабораторной работе**

**Отличия исходных текстов COM и EXE программ.**

1. Сколько сегментов должна содержать COM программа?

Ответ: СОМ программа должна содержать один сегмент;

1. EXE программа?

Ответ: ЕХЕ программа может содержать 1 и более сегментов. Эта программа предусматривает содержание сегмента стека, кода и данных.

1. Какие директивы должны обязательно быть в тексте COM программы?

Ответ: в программе обязательно должна быть директива ORG 100h, которая устанавливает значение программного счетчика в 100h, так как при загрузке СОМ-файла в память DOS занимает первые 256 байт блоком данных PSP и располагает код программы после этого блока.

Также в тексте COM-программы обязательно должна быть директива ASSUME, которая указывает, в значение какого сегмента установлен данный сегментный регистр. В данном случае она указывает, что сегмент кода и данных установлен в сегмент TESTPC. Если закомментировать ASSUME и скомпилировать полученный код, то получим ошибку, которая сообщает об отсутствии или недоступности CS, то есть программа не сможет обнаружить начало сегмента кода.

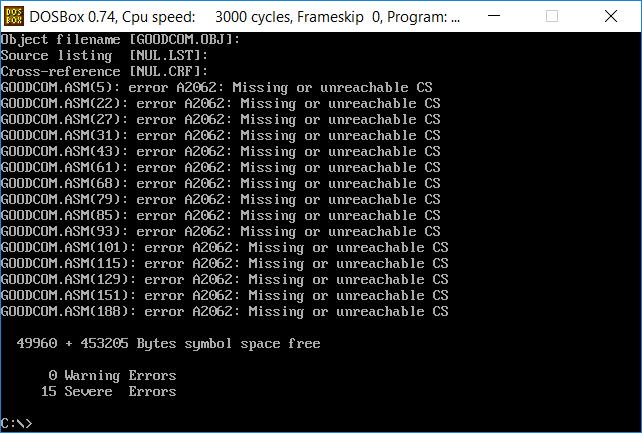


рис 4. Программа без директивы ASSUME

1. Все ли форматы команд можно использовать в COM программе?

Ответ: не все форматы команд можно использовать в СОМ программе, так как в СОМ-программе нет таблицы разметки Relocation Table, вследствие чего ей неоткуда получить информацию об адресе сегмента. А так как в СОМ-программе все сегментные регистры определяются в момент запуска на основе информации о положении полей адресов в файле из Relocation Table, то нельзя использовать команды вида mov <регистр>, seg <имя сегмента>;

**Отличия форматов файлов COM и EXE модулей.**

Запускаем Far и открываем в нём .СОМ модуль, «плохой» .ЕХЕ модуль и «хороший» .ЕХЕ модуль в шестнадцатеричном виде.

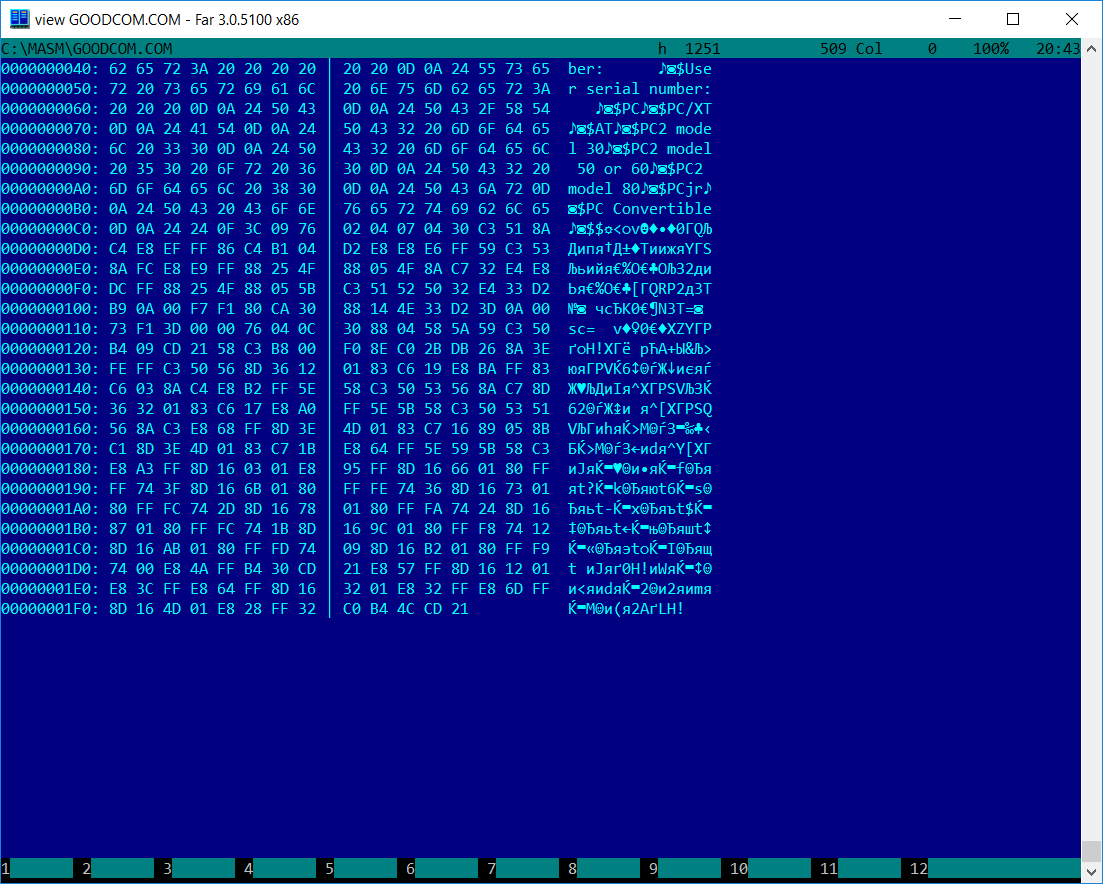


рис.5. «Хороший» .COM модуль в Far

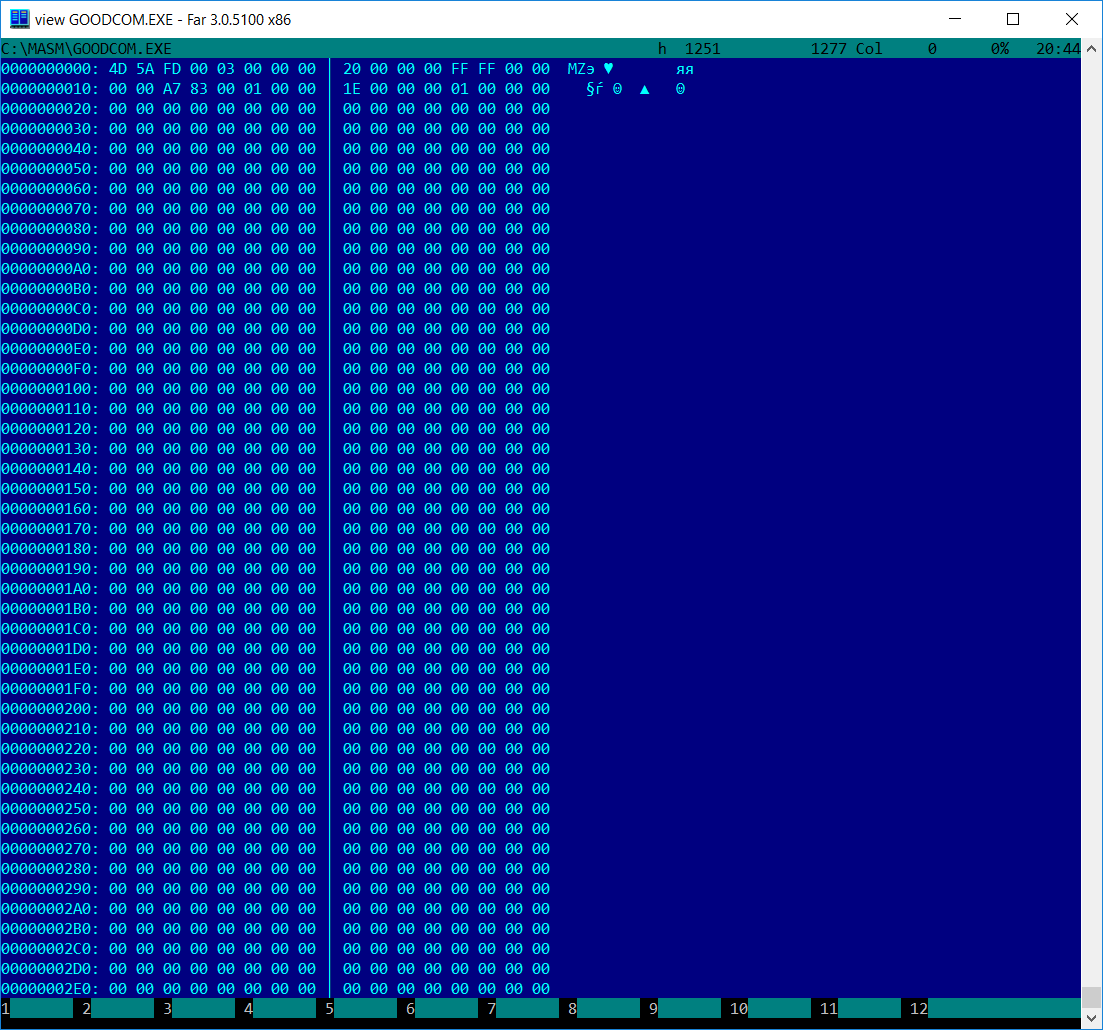


рис.6. «Плохой» .EXE модуль в Far

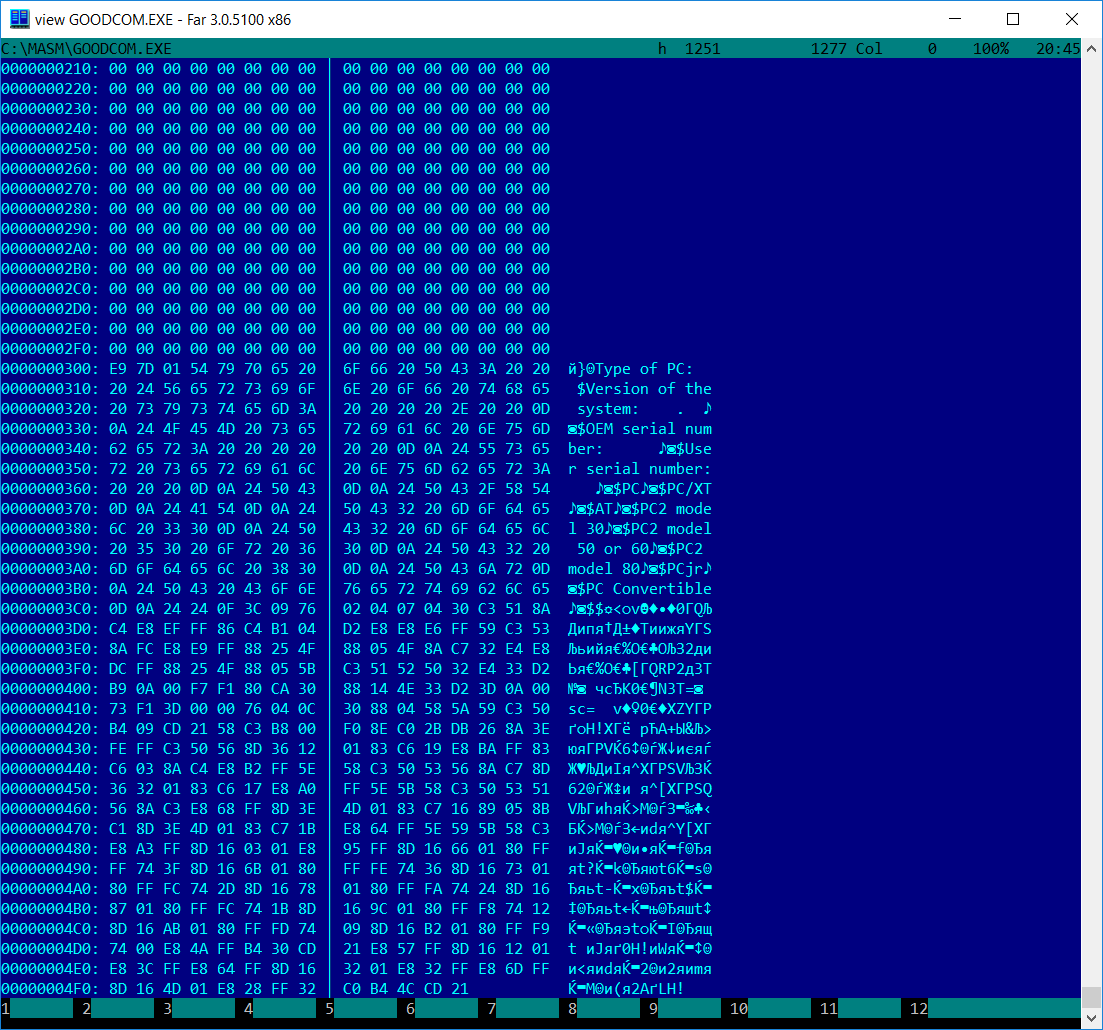


рис.7. «Плохой» .EXE модуль в Far (продолжение)

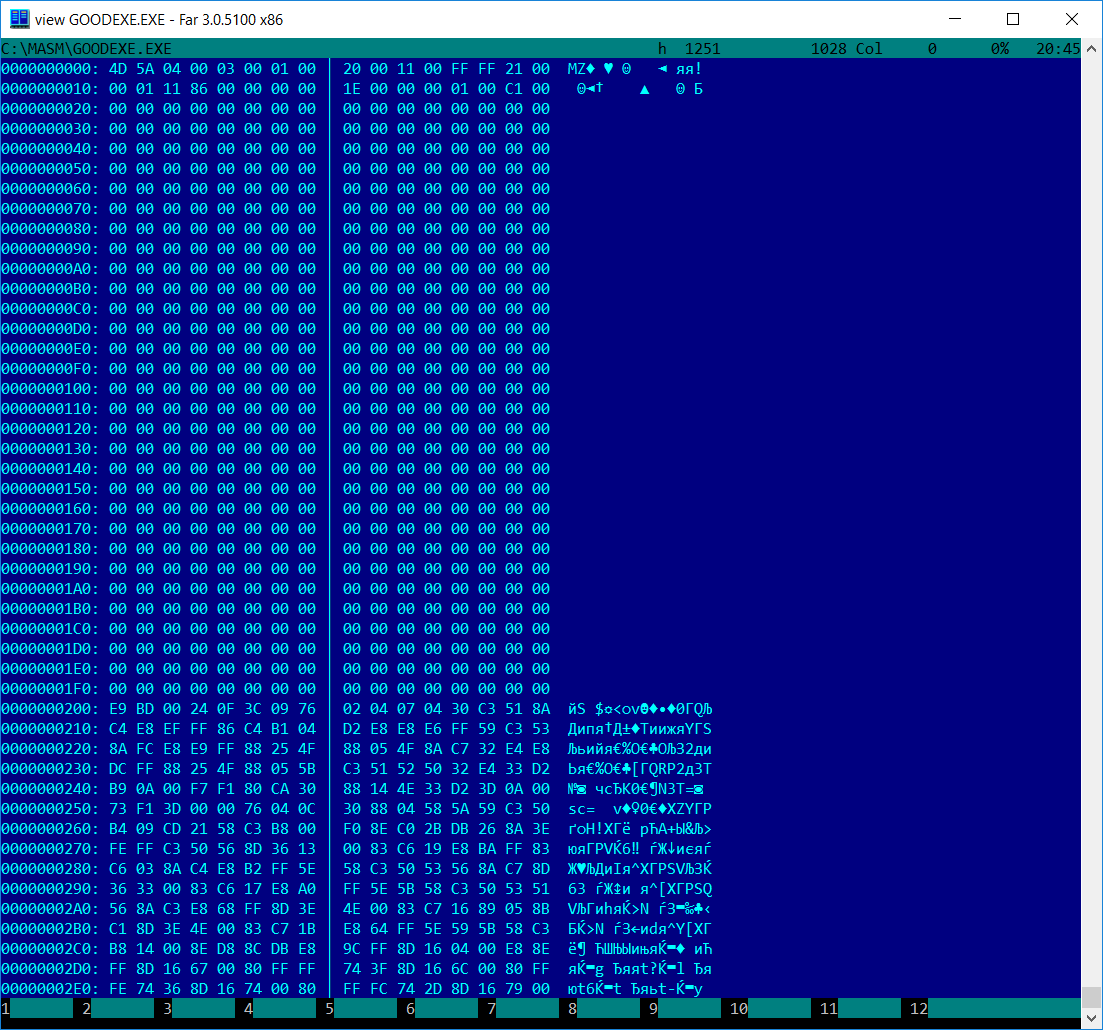


рис.8. «хороший» .EXE модуль в Far

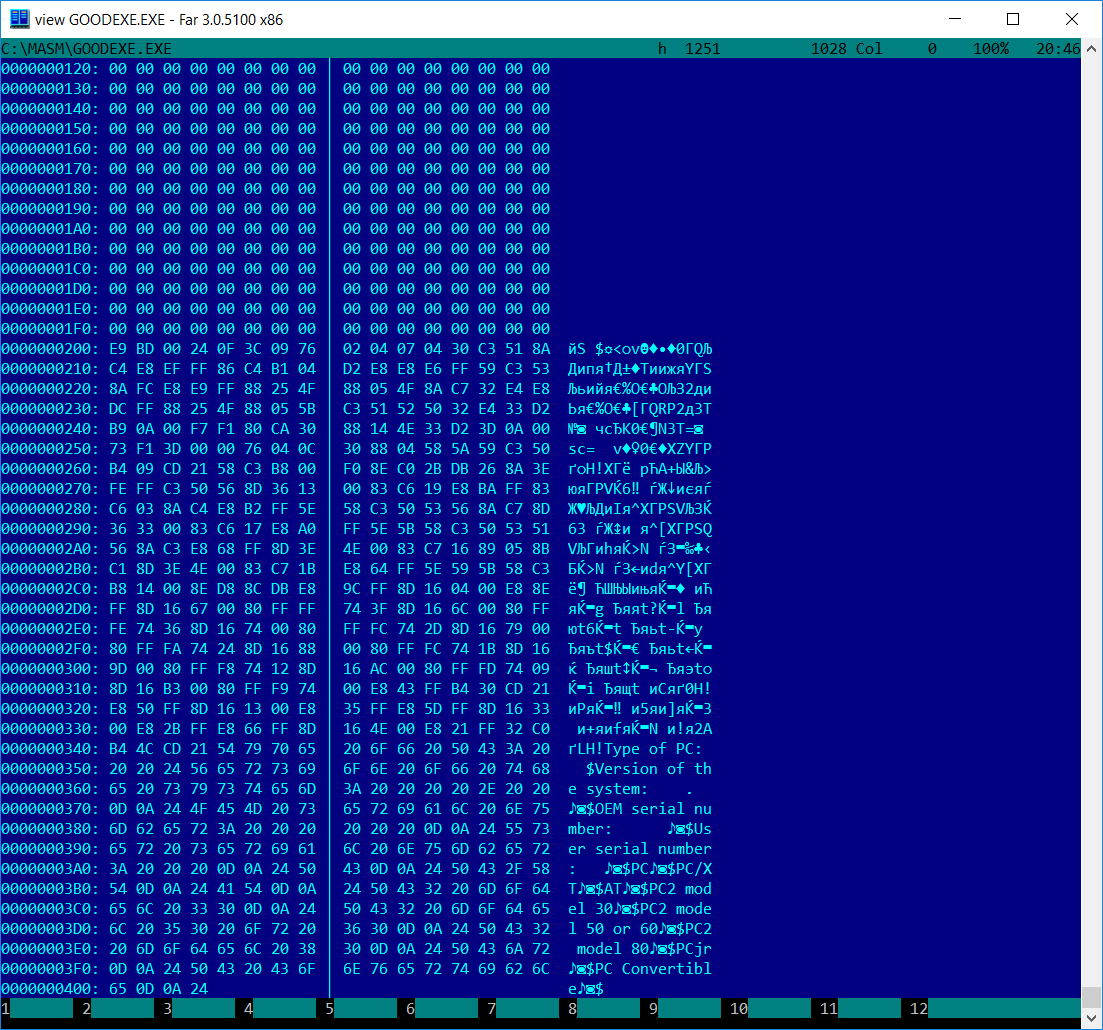


рис.9. «Хороший» .EXE модуль в Far (продолжение)

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

Ответ: файл COM состоит из данных, используемых в программе, и команд. Код располагается с нулевого адреса;

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

Ответ: в файле «плохого» EXE в одном сегменте содержатся данные и код. Код располагается с адреса 300h, так как в нем присутствует начальный блок размером не менее 200h и еще 100h резервирует директива ORG 100h. С 0 адреса располагается таблица разметки;

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

Ответ: в файле «хорошего» EXE содержится сегмент стека, сегмент данных и сегмент кода. Отличие от «плохого» в наличии 3 сегментов. Кроме того, код начинается с 200h, а не с 300h, как в «плохом» ЕХЕ-файле, так как в этом файле отсутствует директива ORG 100h, резервирующая пространство для заголовка;

**Загрузка COM модуля в основную память.**

Откроем отладчик TD.EXE и загрузим .COM.

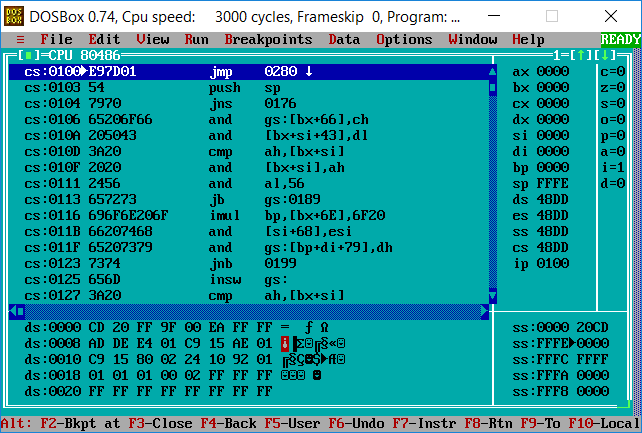


рис.10 .COM модуль в TD

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

Ответ: после загрузки COM-программы в память сегментные регистры указывают на начало PSP. Порядок загрузки: PSP, данные и код, стек. Код располагается с адреса 100h;

1. Что располагается с 0 адреса?

Ответ: с адреса 0 располагается PSP.

1. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Ответ: сегментные регистры имеют значение 48DDh. Они указывают на начало PSP.

1. Как определяется стек? Какую область памяти он занимает? Какие адреса?

Ответ: стек генерируется автоматически, указатель стека в конце сегмента. Он будет занимать оставшуюся память. Стек растет от больших адресов к меньшим, то есть от FFFEh к 0000h.

**Загрузка «хорошего» EXE модуля в память.**

Откроем отладчик TD.EXE и загрузим «хороший» .ЕХЕ.

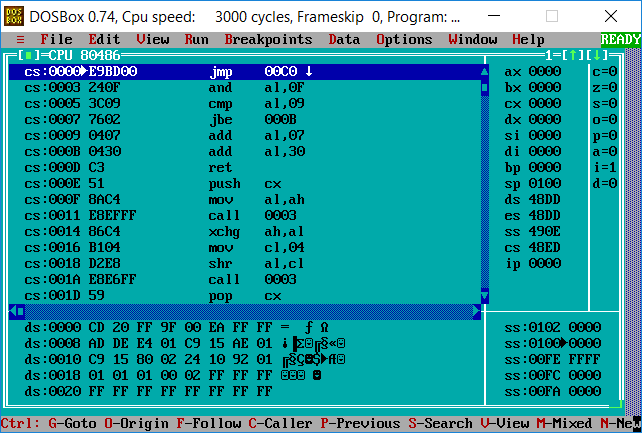


рис.11 «Хороший» .EXE модуль в TD

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

Ответ: вначале строится PSP, стандартная часть заголовка считывается в память, определяется длина тела загрузочного модуля, определяется начальный сегмент, загрузочный модуль считывается в начальный сегмент, таблица разметки считывается в рабочую память, к полю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров.

DS и ES устанавливаются на начало PSP, SS - на начало стека, CS - на начало сегмента кода.

1. На что указывают регистры DS и ES?

Ответ: регистры DS и ES указывают на начало сегмента PSP.

1. Как определяется стек?

Ответ: стек определяется благодаря директиве STACK, а регистрам SS и SP придаются значения, указанные в заголовке, после чего к SS прибавляется сегментный адрес начального сегмента.

1. Как определяется точка входа?

Ответ: точка входа определяется благодаря директиве END;

**Заключение**

В результате выполнения лабораторной работы былb исследованы различия в структурах исходных текстов .COM и .EXE модулей, структур файлов загрузочных модулей и способов их загрузки в основную память.

Приложения

**Код программы** **GOODCOM**.**asm**

TESTPC SEGMENT

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

ORG 100H

START: jmp BEGIN

;Данные

TYPE\_PC db 'Type of PC: ','$'

VERSION db 'Version of the system: . ',0Dh,0Ah,'$'

OEM\_NUM db 'OEM serial number: ',0Dh,0Ah,'$'

USER\_NUM db 'User serial number: ',0Dh,0Ah,'$'

PC db 'PC',0Dh,0Ah,'$'

PC\_XT db 'PC/XT',0Dh,0Ah,'$'

AT db 'AT',0Dh,0Ah,'$'

PC2\_30 db 'PC2 model 30',0Dh,0Ah,'$'

PC2\_50 db 'PC2 model 50 or 60',0Dh,0Ah,'$'

PC2\_80 db 'PC2 model 80',0Dh,0Ah,'$'

PCjr db 'PCjr',0Dh,0Ah,'$'

PC\_Conv db 'PC Convertible',0Dh,0Ah,'$'

;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 al,ah

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

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 ;перевод в 10 с/с, SI - адрес поля младшей цифры

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

;----------------------------

OUTPUT\_PROC PROC NEAR ;Вывод на экран сообщения

push ax

mov ah, 09h

int 21h

pop ax

ret

OUTPUT\_PROC ENDP

;----------------------------

DET\_TYPE PROC near ;Определение типа PC

mov ax, 0F000h

mov es, ax

sub bx, bx

mov bh, es:[0FFFEh]

ret

DET\_TYPE ENDP

;----------------------------

DET\_VERSION PROC near ; Определение версии системы

push ax

push si

lea si,VERSION

add si,19h

call BYTE\_TO\_DEC

add si,3h

mov al,ah

call BYTE\_TO\_DEC

pop si

pop ax

ret

DET\_VERSION ENDP

;-----------------------------

DET\_OEM\_NUM PROC near ; Определение серийного номера ОЕМ

push ax

push bx

push si

mov al,bh

lea si,OEM\_NUM

add si,17h

call BYTE\_TO\_DEC

pop si

pop bx

pop ax

ret

DET\_OEM\_NUM ENDP

;-----------------------------

DET\_USER\_NUM PROC near ;Определение серийного номера пользователя

push ax

push bx

push cx

push si

mov al,bl

call BYTE\_TO\_HEX

lea di,USER\_NUM

add di,22

mov [di],AX

mov ax,cx

lea di,USER\_NUM

add di,27

call WRD\_TO\_HEX

pop si

pop cx

pop bx

pop ax

ret

DET\_USER\_NUM ENDP

;-----------------------------

BEGIN:

;Вызываем функцию определения типа РС и выводим поясняющее сообщение

call DET\_TYPE

lea dx,TYPE\_PC

call OUTPUT\_PROC

;Определяем по предпоследнему биту ROM BIOS тип IBM PC

lea dx, PC

cmp bh, 0FFh

je output

lea dx, PC\_XT

cmp bh, 0FEh

je output

lea dx, AT

cmp bh, 0FCh

je output

lea dx, PC2\_30

cmp bh, 0FAh

je output

lea dx, PC2\_50

cmp bh, 0FCh

je output

lea dx, PC2\_80

cmp bh, 0F8h

je output

lea dx, PCjr

cmp bh, 0FDh

je output

lea dx, PC\_Conv

cmp bh, 0F9h

je output

output:

call OUTPUT\_PROC

;Определяем оставшиеся характеристики и выводим их на экран

mov ah,30h

int 21h

call DET\_VERSION ; Определяем версию системы

lea dx,VERSION

call OUTPUT\_PROC

call DET\_OEM\_NUM ; Определеляем серийный номер ОЕМ

lea dx,OEM\_NUM

call OUTPUT\_PROC

call DET\_USER\_NUM ;Определяем серийный номер пользователя

lea dx,USER\_NUM

call OUTPUT\_PROC

;Выход в DOS

xor al, al

mov ah, 4ch

int 21h

TESTPC ENDS

END START

**Код программы GOODEXE.asm**

.MODEL SMALL

.STACK 100h

.DATA

EOF EQU '$'

;Данные

TYPE\_PC db 'Type of PC: ','$'

VERSION db 'Version of the system: . ',0Dh,0Ah,'$'

OEM\_NUM db 'OEM serial number: ',0Dh,0Ah,'$'

USER\_NUM db 'User serial number: ',0Dh,0Ah,'$'

PC db 'PC',0Dh,0Ah,'$'

PC\_XT db 'PC/XT',0Dh,0Ah,'$'

AT db 'AT',0Dh,0Ah,'$'

PC2\_30 db 'PC2 model 30',0Dh,0Ah,'$'

PC2\_50 db 'PC2 model 50 or 60',0Dh,0Ah,'$'

PC2\_80 db 'PC2 model 80',0Dh,0Ah,'$'

PCjr db 'PCjr',0Dh,0Ah,'$'

PC\_Conv db 'PC Convertible',0Dh,0Ah,'$'

.CODE

START: jmp BEGIN

;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 al,ah

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

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 ;перевод в 10 с/с, SI - адрес поля младшей цифры

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

;----------------------------

OUTPUT\_PROC PROC NEAR ;Вывод на экран сообщения

push ax

mov ah, 09h

int 21h

pop ax

ret

OUTPUT\_PROC ENDP

;----------------------------

DET\_TYPE PROC near ;Определение типа PC

mov ax, 0F000h

mov es, ax

sub bx, bx

mov bh, es:[0FFFEh]

ret

DET\_TYPE ENDP

;----------------------------

DET\_VERSION PROC near ; Определение версии системы

push ax

push si

lea si,VERSION

add si,19h

call BYTE\_TO\_DEC

add si,3h

mov al,ah

call BYTE\_TO\_DEC

pop si

pop ax

ret

DET\_VERSION ENDP

;-----------------------------

DET\_OEM\_NUM PROC near ; Определение серийного номера ОЕМ

push ax

push bx

push si

mov al,bh

lea si,OEM\_NUM

add si,17h

call BYTE\_TO\_DEC

pop si

pop bx

pop ax

ret

DET\_OEM\_NUM ENDP

;-----------------------------

DET\_USER\_NUM PROC near ;Определение серийного номера пользователя

push ax

push bx

push cx

push si

mov al,bl

call BYTE\_TO\_HEX

lea di,USER\_NUM

add di,22

mov [di],AX

mov ax,cx

lea di,USER\_NUM

add di,27

call WRD\_TO\_HEX

pop si

pop cx

pop bx

pop ax

ret

DET\_USER\_NUM ENDP

;-----------------------------

BEGIN:

mov ax, @data

mov ds, ax

mov bx, ds

;Вызываем функцию определения типа РС и выводим поясняющее сообщение

call DET\_TYPE

lea dx,TYPE\_PC

call OUTPUT\_PROC

;Определяем по предпоследнему биту ROM BIOS тип IBM PC

lea dx, PC

cmp bh, 0FFh

je output

lea dx, PC\_XT

cmp bh, 0FEh

je output

lea dx, AT

cmp bh, 0FCh

je output

lea dx, PC2\_30

cmp bh, 0FAh

je output

lea dx, PC2\_50

cmp bh, 0FCh

je output

lea dx, PC2\_80

cmp bh, 0F8h

je output

lea dx, PCjr

cmp bh, 0FDh

je output

lea dx, PC\_Conv

cmp bh, 0F9h

je output

output:

call OUTPUT\_PROC

;Определяем оставшиеся характеристики и выводим их на экран

mov ah,30h

int 21h

call DET\_VERSION ; Определяем версию системы

lea dx,VERSION

call OUTPUT\_PROC

call DET\_OEM\_NUM ; Определеляем серийный номер ОЕМ

lea dx,OEM\_NUM

call OUTPUT\_PROC

call DET\_USER\_NUM ;Определяем серийный номер пользователя

lea dx,USER\_NUM

call OUTPUT\_PROC

;Выход в DOS

xor al, al

mov ah, 4ch

int 21h

END START