**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. 9383 |  | Орлов Д.С. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2021

**Постановка задачи**

Шаг 1. Написать текст исходного .COM модуля, который определяет тип PC и версию системы. Резульаттом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 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».

Шаг 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.

Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».

Шаг 7. Оформить отчет в соотсветствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Таблица 1. Используемые процедуры

|  |  |
| --- | --- |
| Процедура | Описание |
| TETR\_TO\_HEX | Перевод десятичной цифры в код символа ASCII |
| BYTE\_TO\_HEX | Перевод байта в AL в два символа шестн. числа AX |
| BYTE\_TO\_WRD | Запись AL в строку DEC\_NUMBER |
| PC\_TYPE | Вывод типа ПК на экран |
| OS\_VERSION | Вывод версии ОС на экран |

**Выполнение шагов**

**Шаг 1.** В процессе выполнения лабораторной работы была разработана программа для модуля .COM, исходный код которой приведен в приложении А. После отладки был получен «плохой» .EXE модуль, а из него был получен «хороший» .COM модуль.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1. Пример работы модуля .COM

Также, был создан «плохой» .EXE, который работает неверно:

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2. Пример работы «плохого» .EXE

**Шаг 2.** В процессе выполнения лабораторной работы была разработана программа для «хорошего» модуля .EXE, исходный код которой приведен в приложении Б.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 3. Пример работы «хорошего» .EXE

**Шаг 3.** Ответим на вопросы по теме «Отличия исходных текстов .EXE и .COM программ».

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент.

2. EXE-программа?

Такая программа должна содержать не менее одного сегмента (один и более).

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

В COM программе обязательно должна быть прописана директива org 100h, т. к. первые 100h байт занимает PSP. Это позволит сместить все относительные адреса на 100h байт. Также необходимо прописать (ASSUME CS:MYSEGMENT, DS:MYSEGMENT, ES:NOTHING, SS:NOTHING), чтобы сегмент данных и сегмент кода указывали на один и тот же сегмент.

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

Нельзя использовать команды вида <mov register, seg segname>. Т. к. в COM-программе отсутствует таблица настроек.

**Шаг 4.** Шестнадцатеричное представление:Изображение выглядит как текст

Автоматически созданное описание

Рис. 4. Шестнадцатеричное представление модуля .COM

Шестнадцатеричное представление «плохого» модуля .EXE:

Рис. 5. Шестнадцатеричное представление «плохого» модуля .EXE

Изображение выглядит как текст

Автоматически созданное описание Рис. 6. Шестнадцатеричное представление «хорошего» модуля .EXE

Отличия .COM от «плохого» .EXE в том, что в .COM программа идет без заголовков. В «плохом» .EXE есть неправильный заголовок, потому что он создавался по COM-программе.

В «хорошем» .EXE заголовок корректный, также EXE-файл стал немного больше в размерах.

**Отличия форматов файлов COM и EXE модулей**

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

COM-файл содержит данные и машинные программы. Код располагается по адресу 0h, но при загрузке модуля устанавливается смещение 100h.

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

«Плохой» EXE содержит EXE-маркер (байты MZ), заголовок и таблицу настроек. В заголовке содержится различная служебная информация для загрузчика. В таблице настроек содержатся записи в формате сегмент: смещение. К смещениям в загрузочном модуле, на которые указывают значения в таблице, после загрузки программы в память будет прибавлен сегментный адрес. Код и данные располагаются в одном сегменте. Код располагается с 300h байта. С адреса 2 располагаются заголовок и таблица настроек после него.

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

В «хорошем» EXE код, данные и стек разделены на сегменты. С адреса 2 в «хорошем» EXE располагается заголовок, а после него идет таблица настроек. Код начинается с 400h, т.к. размер заголовка и таблицы — 200h, и размер стека — 200h.

**Шаг 5.**

**Загрузка COM-модуля в основную память**

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

ОС ищет подходящее по размеру место в оперативной памяти для COM-модуля. Далее она помещает в это место PSP, а по смещению в 100h помещает считанный с диска модуль.

2. Что располагается с адреса 0?

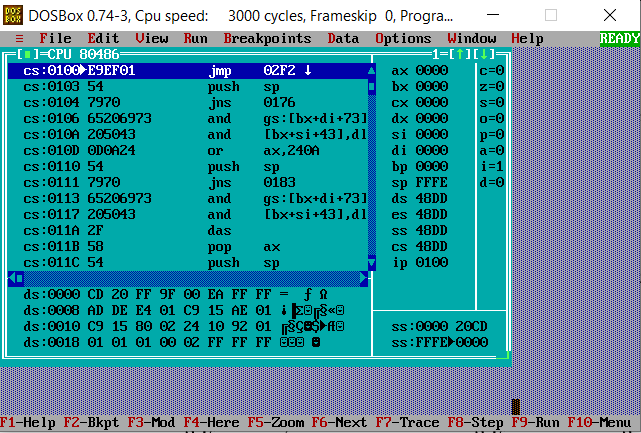
С адреса 0 располагается PSP размером в 100h байт.

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

CS, ES, DS и SS указывают на PSP (48DDh), а SP указывает на конец сегмента (FFFEh).

4. Как определяется стек? Какую область он занимает? Какие адреса?

Стек определяется автоматически при ассемблировании и линковке. Стек, теоретически, занимает всю область сегмента. SP указывает на конец стека (FFFEh), а SS указывает на начало стека (48DDh).

 Рис. 7. Интерфейс TD.EXE (.COM)

**Шаг 6.**

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

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

Подготовка загрузки аналогична .COM. EXE загружается по смещению 100h от PSP. В процессе загрузки считывается информация EXE-заголовка и выполняется перемещение адресов сегментов. CS указывает на начало сегмента команд (491Ah). В IP загружается смещение точки входа в программу.

2. На что указывают регистры DS и ES?

На начало сегмента PSP (48DDh).

3. Как определяется стек?

Стек определяется на основе директивы .stack и SP указывает на конец сегмента стека.

4. Как определяется точка входа?

Точка входа определяется с помощью директивы END.

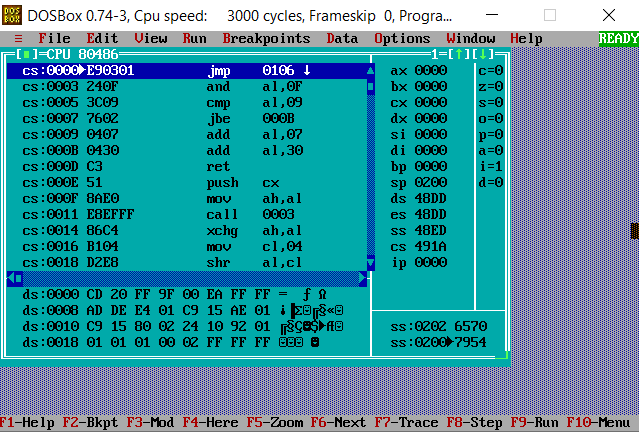
****

Рис. 8. Интерфейс TD.EXE (.EXE)

**Заключение**

В процессе выполнения лабораторной работы были изучены структурные отличия .COM и .EXE модулей и получены навыки работы с Far Manager и отладчиком TD.EXE.

**Приложение А.**

; 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\_T db 'Type is PC',0DH,0AH,'$'

PC\_XT\_T db 'Type is PC/XT',0DH,0AH,'$'

AT\_TY db 'Type is AT',0DH,0AH,'$'

PS30\_T db 'Type is PS2 model 30',0DH,0AH,'$'

PS80\_T db 'Type is PS2 model 80',0DH,0AH,'$'

PCCON\_T db 'Type is PC Convertible',0DH,0AH,'$'

PCjr\_T db 'Type is PCjr',0DH,0AH,'$'

NO\_T db 'ERROR: No type in table: ',0DH,0AH,'$'

VERSION db 'Version: . ',0DH,0AH,'$'

OEM db 'OEM: ',0DH,0AH,'$'

USER db 'User: $'

STRING db 'Значение регистра AX= ',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 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

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

; КОД

PC\_TYPE PROC near

mov AX, 0f000h ;получаем тип пк

mov ES, AX

mov AL, es:[0fffeh]

cmp AL, 0FFh

je pc

cmp AL, 0FEh

je pc\_xt

cmp AL, 0FBh

je pc\_xt

cmp AL, 0FCh

je at\_t

cmp AL, 0FAh

je ps30

cmp AL, 0F8h

je ps80

cmp AL, 0FDh

je pcjr

cmp AL, 0F9h

je pccon

mov dx, offset NO\_T

jmp WRITE\_STRING

pc:

mov dx, offset PC\_T

jmp WRITE\_STRING

pc\_xt:

mov dx, offset PC\_XT\_T

jmp WRITE\_STRING

at\_t:

mov dx, offset AT\_TY

jmp WRITE\_STRING

ps30:

mov dx, offset PS30\_T

jmp WRITE\_STRING

ps80:

mov dx, offset PS80\_T

jmp WRITE\_STRING

pccon:

mov dx, offset PCCON\_T

jmp WRITE\_STRING

pcjr:

mov dx, offset PCjr\_T

jmp WRITE\_STRING\_T

WRITE\_STRING\_T:

call WRITE\_STRING

ret

PC\_TYPE ENDP

OS\_VERSION PROC near

MOV AH, 30h

INT 21h

push AX

mov SI, offset VERSION

add SI, 9

call BYTE\_TO\_DEC

pop AX

mov AL, AH

add SI, 3

call BYTE\_TO\_DEC

mov DX, offset VERSION

call WRITE\_STRING

mov SI, offset OEM

add SI, 5

mov AL, BH

call BYTE\_TO\_DEC

mov DX, offset OEM

call WRITE\_STRING

mov DI, offset USER

add DI, 11

mov AX, CX

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

sub DI, 2

mov [DI], AX

mov DX, offset USER

call WRITE\_STRING

ret

OS\_VERSION ENDP

WRITE\_STRING PROC near

; Вывод строки текста из поля STRING

mov AH,09h

int 21h

ret

WRITE\_STRING ENDP

BEGIN:

call PC\_TYPE

call OS\_VERSION

; Выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START ;конец модуля, START - точка входа

**Приложение Б.**

Astack SEGMENT STACK

DB 512 DUP (?)

Astack ENDS

DATA SEGMENT

; Данные

PC\_T db 'Type is PC',0DH,0AH,'$'

PC\_XT\_T db 'Type is PC/XT',0DH,0AH,'$'

AT\_TY db 'Type is AT',0DH,0AH,'$'

PS30\_T db 'Type is PS2 model 30',0DH,0AH,'$'

PS80\_T db 'Type is PS2 model 80',0DH,0AH,'$'

PCCON\_T db 'Type is PC Convertible',0DH,0AH,'$'

PCjr\_T db 'Type is PCjr',0DH,0AH,'$'

NO\_T db 'ERROR: No type in table: ',0DH,0AH,'$'

VERSION db 'Version: . ',0DH,0AH,'$'

OEM db 'OEM: ',0DH,0AH,'$'

USER db 'User: $'

DATA ENDS

CODE SEGMENT

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

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 шест. 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

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

PC\_TYPE PROC near

mov AX, 0f000h ;получаем тип пк

mov ES, AX

mov AL, es:[0fffeh]

cmp AL, 0FFh

je pc

cmp AL, 0FEh

je pc\_xt

cmp AL, 0FBh

je pc\_xt

cmp AL, 0FCh

je at\_t

cmp AL, 0FAh

je ps30

cmp AL, 0F8h

je ps80

cmp AL, 0FDh

je pcjr

cmp AL, 0F9h

je pccon

mov dx, offset NO\_T

jmp WRITE\_STRING

pc:

mov dx, offset PC\_T

jmp WRITE\_STRING

pc\_xt:

mov dx, offset PC\_XT\_T

jmp WRITE\_STRING

at\_t:

mov dx, offset AT\_TY

jmp WRITE\_STRING

ps30:

mov dx, offset PS30\_T

jmp WRITE\_STRING

ps80:

mov dx, offset PS80\_T

jmp WRITE\_STRING

pccon:

mov dx, offset PCCON\_T

jmp WRITE\_STRING

pcjr:

mov dx, offset PCjr\_T

jmp WRITE\_STRING\_T

WRITE\_STRING\_T:

call WRITE\_STRING

ret

PC\_TYPE ENDP

OS PROC near

;версия

mov ah, 30h

int 21h

push ax

mov si, offset VERSION

add si, 9 ;смещение

call BYTE\_TO\_DEC

pop ax

mov al, ah

add si, 3 ;смещение

call BYTE\_TO\_DEC

mov dx, offset VERSION

call WRITE\_STRING

;серийный номер OEM

mov si, offset OEM

add si, 5 ;смещение

mov al, bh

call BYTE\_TO\_DEC

mov dx, offset OEM

call WRITE\_STRING

;серийный номер пользователя

mov di, offset USER

add di, 11 ;смещение

mov ax, cx

call WRD\_TO\_HEX

mov al, bl

call BYTE\_TO\_HEX

sub di, 2

mov [di], ax

mov dx, offset USER

call WRITE\_STRING

ret

OS ENDP

WRITE\_STRING PROC near

; Вывод строки текста из поля STRING

mov AH,09h

int 21h

ret

WRITE\_STRING ENDP

; Код

BEGIN:

mov AX, DATA

mov DS, AX

call PC\_TYPE

call OS

xor AL,AL

mov AH,4Ch

int 21H

CODE ENDS

END START; конец модуля, START - точка выхода