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. 9381 _____ Игнашов В.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Задание.

«Истина познается в сравнении», как говорили древние. К счастью, у нас есть возможность исследовать в одной системе два различных формата загрузочных модулей, сравнить их и лучше понять как система программирования и управляющая программа обращаются с ними. Система программирования включает компилятор с языка ассемблер (часто называется, просто, ассемблер), который изготавливает объектные модули. Компоновщик (Linker) по совокупности объектных модулей, изготавливает загрузочный модуль, а также, функция ядра – загрузчик, которая помещает программу в основную память и запускает на выполнение. Все эти компоненты согласованно работают для изготовления и выполнения загрузочных модулей разного типа. Для выполнения лабораторной работы сначала нужно изготовить загрузочные модули.

Шаг 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.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

Используя шаблон из методических указаний был написан исходный текст com-модуля. С помощью программных пакетов TASM и TLINK был

получен «хороший» загрузочный модуль COM и с помощью MASM и LINK был получен плохой загрузочный модуль EXE.

```
F:\>tasm lr1.asm
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   lr1.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  473k


F:\>tlink /t lr1.obj
Turbo Link  Version 5.1 Copyright (c) 1992 Borland International

F:\>lr1.com
PC Type: FC
Modification number: 5.0
OEM: 255
Serial Number: 000000
```

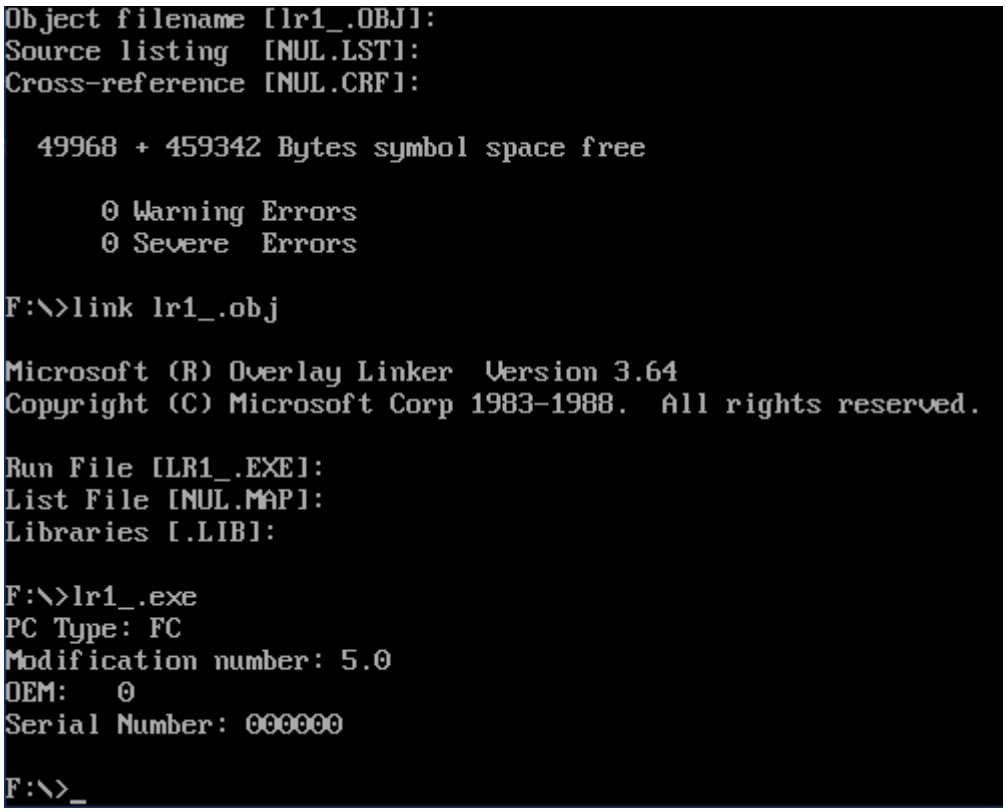
lr1.com

```
Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Run File [LR1.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

F:\>lr1.exe
        FC                      5 0        255                 000000


           0ó PC Type:
                  5 0        255                 000000


0ó PC Type:
    255                000000

                                                0ó PC Type:
             000000

                                        0ó PC Type:
```

lr1.exe

Также был переписан код для exe-модуля, который работает идентично com-модулю – «хороший» exe-модуль

lr1_.exe



Ответы на контрольные вопросы:

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа? – **1**
2) EXE-программа? – **любое количество**
3) Какие директивы должны обязательно быть в тексте COM-программы? – **Директива ASSUME – сообщение транслятору о том, какие сегменты использовать в соответствии с какими регистрами, т.к. сегменты сами по себе равноправны. Директива ORG 100h, задающая смещение для всех адресов программы для префикса программного сегмента**
4) Все ли форматы команд можно использовать в COM-программе? - **В COM-программе можно использовать только near-переходы из-за наличия только 1 сегмента. Команды с адресом сегмента также нельзя использовать, тк он до загрузки неизвестен.**

Отличия форматов файлов COM и EXE модулей

1) Какова структура файла COM? С какого адреса располагается код? – **Состоит из команд, функций и данных. С 0 адреса**

2) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с 0 адреса? – **Данные и код содержатся в одном сегменте. Код – с адреса 300h. С 0 адреса – информация о том, что это EXE-файл**
3) Какова структура файла «хорошего» EXE? Чем он отличается от «плохого» EXE файла? – **В хорошем exe-файле стек и код разделены по сегментам и не имеется директивы ORG 100h(под префикс программного сегмента) – код начинается с адреса 200h**

Загрузка COM модуля в основную память

1) Какой формат загрузки модуля COM? С какого адреса располагается код? – **Код с адреса 100h, сегментные регистры указывают на начало префикса программного сегмента после загрузки**
2) Что располагается с адреса 0? – **Префикс программного сегмента**
3) Какие значения имеют сегментные регистры? На какие области памяти они указывают? – **Сегментные регистры имеют значения, соответствующие сегменту, в который был помещен модуль. 48DD(PSP), так как один и тот же сегмент памяти**
4) Как определяется стек? Какую область памяти он занимает? Какие адреса? – **Указатель стека в конце сегмента, занимает неиспользованную память, адреса меняются от FFFEh к 0000h**

Загрузка «хорошего» EXE модуля в основную память

1) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры? – **Создается префикс программного сегмента, в начальный сегмент записывается загрузочный модуль, таблица настройки – в рабочую память. Каждый сегмент прибавляет сегментный адрес начального сегмента данных. Далее определяются значения сегментных регистров. DS и ES – начало PSP(48DD), CS – Начало команд(4932). SS – начало стека(48ED)**
2) На что указывают регистры DS и ES? - **DS и ES – начало PSP(48DD)**
3) Как определяется стек? - **В отличие от COM стек определяется при объявлении его сегмента, выделяется указанная память.**

4) Как определяется точка входа? – **директивой END с адресом, с которого начинается выполнение программы.**

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lr1.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_Type               db    'PC Type:  ', 0dh, 0ah,'$'
Mod_numb        db    'Modification number:  .  ', 0dh, 0ah,'$'
OEM                   db    'OEM:   ', 0dh, 0ah, '$'
S_numb          db    'Serial Number:       ', 0dh, 0ah, '$'
;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
BEGIN:
;PC_Type
      push es
      push bx
      push ax
      mov bx, 0F000h
      mov es, bx
      mov ax, es:[0FFFEh]
      mov ah, al
      call BYTE_TO_HEX
      lea bx, PC_Type
      mov [bx+9], ax; смещение по колву символов, записали в PC_Type
по адресу
      pop ax
      pop bx
      pop es

      mov ah, 30h; Воспользуемся функцией получения информации о MS
DOS
      int 21h

;Mod_numb
      push ax
      push si
      lea si, Mod_numb; в si адрес Mod_numb
      add si, 21; Сместимся на 21 символ
      call BYTE_TO_DEC; al - Basic version number
      add si, 3; Еще на три
      mov al, ah
      call BYTE_TO_DEC; al - Modification number
      pop si
      pop ax

;OEM
      mov al, bh
      lea si, OEM
      add si, 7
```

```
      call BYTE_TO_DEC; al - OEM number

;S_numb
      mov al, bl
      call BYTE_TO_HEX; al - 24b number
      lea di, S_numb
      add di, 15
      mov [di], ax
      mov ax, cx
      lea di, S_numb
      add di, 20
      call WRD_TO_HEX

;Output
      mov AH,09h
      lea DX, PC_Type
      int 21h
      lea DX, Mod_numb
      int 21h
      lea DX, OEM
      int 21h
      lea DX, S_numb
      int 21h

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

Название файла: lr1_.asm

```
AStack SEGMENT STACK
AStack ENDS

DATA SEGMENT
PC_Type               db    'PC Type:  ', 0dh, 0ah,'$'
Mod_numb        db    'Modification number:  .  ', 0dh, 0ah,'$'
OEM                   db    'OEM:   ', 0dh, 0ah, '$'
S_numb         db     'Serial Number:       ', 0dh, 0ah, '$'
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
;--------------------------------
; КОД
main:
      push ds
      sub ax, ax
      push ax
      mov ax, DATA
      mov ds, ax
;PC_Type
      push es
      push bx
```

```
            push ax
            mov bx, 0F000h
            mov es, bx
            mov ax, es:[0FFFEh]
            mov ah, al
            call BYTE_TO_HEX
            lea bx, PC_Type
            mov [bx+9], ax; смещение по колву символов, записали в PC_Type
по адресу
            pop ax
            pop bx
            pop es

            mov ah, 30h; Воспользуемся функцией получения информации о MS
DOS
            int 21h

      ;Mod_numb
            push ax
            push si
            lea si, Mod_numb; в si адрес Mod_numb
            add si, 21; Сместимся на 21 символ
            call BYTE_TO_DEC; al - Basic version number
            add si, 3; Еще на три
            mov al, ah
            call BYTE_TO_DEC; al - Modification number
            pop si
            pop ax

      ;OEM
            mov al, bh
            lea si, OEM
            add si, 7
            call BYTE_TO_DEC; al - OEM number

      ;S_numb
            mov al, bl
            call BYTE_TO_HEX; al - 24b number
            lea di, S_numb
            add di, 15
            mov [di], ax
            mov ax, cx
            lea di, S_numb
            add di, 20
            call WRD_TO_HEX

      ;Output
            mov AH,09h
            lea DX, PC_Type
            int 21h
            lea DX, Mod_numb
            int 21h
            lea DX, OEM
            int 21h
            lea DX, S_numb
            int 21h

      ; Выход в DOS
            xor AL,AL
            mov AH,4Ch
```

```
      int 21H
CODE ENDS
      END main
```