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 _____ Давыдов Д.С.

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.

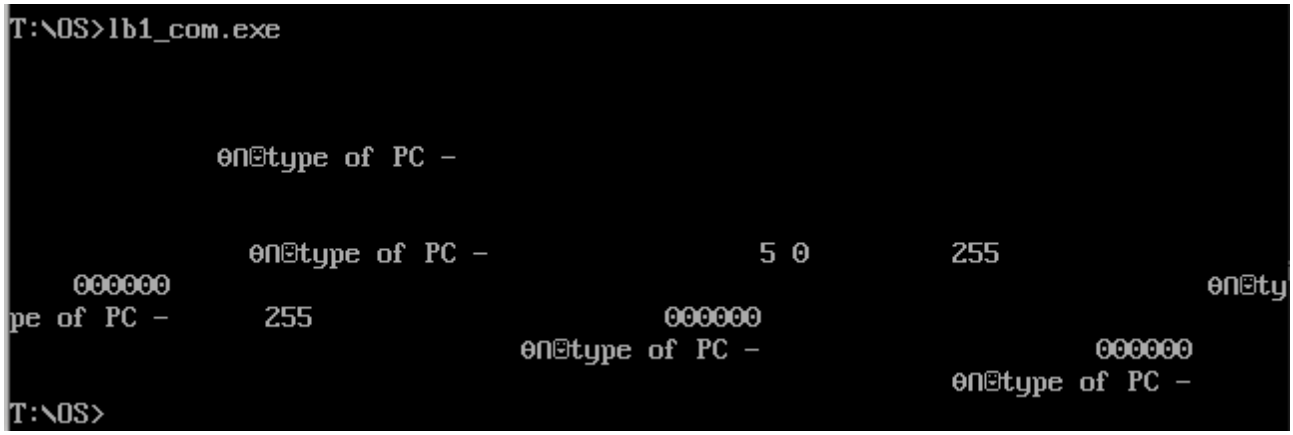
Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

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.

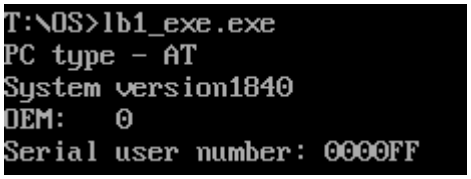
Используя шаблон из методических указаний был написан исходный текст com-модуля. С помощью команд TASM и TLINK был получен «хороший» загрузочный модуль COM и «плохой» модуль EXE.

Результат запуска exe модуля:



Результат запуска com модуля:

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



Разработанный программный код смотреть в приложении А.

Ответы на контрольные вопросы:

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

   Ответ: 1

2) EXE-программа?

   Ответ: любое количество

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

   Ответ: 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?
   Ответ: 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? –
   Ответ: 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?
   Ответ: 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?
   Ответ: 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?
   Ответ: Код с адреса 100h, сегментные регистры указывают на начало префикса программного сегмента после загрузки
2) Что располагается с адреса 0?
   Ответ: Префикс программного сегмента
3) Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Ответ: 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?

   Ответ: 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?

   Ответ: 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?

   Ответ: DS и ES – начало PSP(48DD)

3) Как определяется стек?

   Ответ: В отличие от COM стек определяется при объявлении его сегмента, выделяется указанная память.

4) Как определяется точка входа? –

   Ответ: директивой END с адресом, с которого начинается выполнение программы.

Выводы.

Были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способ их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lr1_com.asm

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

;ДАННЫЕ
pc_type_str DB 'type of PC - ', '$'
pc_type_1 DB 'PC', 0DH, 0AH, '$'
pc_type_2 DB 'PC/XT', 0DH, 0AH, '$'
pc_type_3 DB 'AT', 0DH, 0AH, '$'
pc_type_4 DB 'PS2 model 30', 0DH, 0AH, '$'
pc_type_5 DB 'PS2 model 50 or 60', 0DH, 0AH, '$'
pc_type_6 DB 'PS2 model 80', 0DH, 0AH, '$'
pc_type_7 DB 'PCjr', 0DH, 0AH, '$'
pc_type_8 DB 'PC Convertible', 0DH, 0AH, '$'
unknown_pc_type DB '     error. Unknown', 0DH, 0AH ,'$'
System_version DB 'System version:  .  ',0DH, 0AH, '$'
OEM DB 'OEM:   ', 0DH, 0AH, '$'
user_serial_number DB 'Serial user number:       ',0DH, 0AH, '$'

;Представление 4 бита регистра al в виде цифры 16ой с.с. и
представление её в символьном виде.
TETR_TO_HEX PROC near
      and AL,0Fh
      cmp AL,09
      jbe NEXT
      add AL,07
NEXT:
      add AL,30h ; результат в al
      ret
TETR_TO_HEX ENDP

;Представление al как два числа в 16-ой с.с. и перемещение их в ax
BYTE_TO_HEX PROC near
      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

; перевод в 16 с.с 16-ти разрядного числа
; в AX - число, а в DI - адрес последнего символа
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
        call BYTE_TO_HEX
        mov [DI],AH
        dec DI
        mov [DI],AL
        pop BX
        ret
    WRD_TO_HEX ENDP

    ; перевод в 10 с/с. SI - адрес поля младшей цифры
    BYTE_TO_DEC PROC near
        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

    WRITE PROC NEAR
        push ax
        mov   AH, 9
       int   21h  ; функция DOS по прерыванию
        pop ax
       ret
    WRITE ENDP

    ; получение типа PC
    GET_PC_TYPE PROC NEAR
        push ax
        push dx
        push es
        mov ax, 0F000h
        mov es, ax
        mov al, es:[0FFFEh]
        mov dx, offset pc_type_str
        call WRITE

        cmp al, 0FFh ; распознавание типа PC по специльной технической
таблице
        je pc_type_1_case
        cmp al, 0FEh
        je pc_type_2_case
        cmp al, 0FBh
        je pc_type_2_case
```

```
            cmp al, 0FCh
            je pc_type_3_case
            cmp al, 0FAh
            je pc_type_4_case
            cmp al, 0FCh
            je pc_type_5_case
            cmp al, 0F8h
            je pc_type_6_case
            cmp al, 0FDh
            je pc_type_7_case
            cmp al, 0F9h
            je pc_type_8_case
            jmp unknown_pc_type_case

      pc_type_1_case:
            mov dx, offset pc_type_1 ; загрузка в зависимости от значения
al смещения нужной строки
            jmp final_step
      pc_type_2_case:
            mov dx, offset pc_type_2
            jmp final_step
      pc_type_3_case:
            mov dx, offset pc_type_3
            jmp final_step
      pc_type_4_case:
            mov dx, offset pc_type_4
            jmp final_step
      pc_type_5_case:
            mov dx, offset pc_type_5
            jmp final_step
      pc_type_6_case:
            mov dx, offset pc_type_6
            jmp final_step
      pc_type_7_case:
            mov dx, offset pc_type_7
            jmp final_step
      pc_type_8_case:
            mov dx, offset pc_type_8
            jmp final_step

      unknown_pc_type_case:
            mov dx, offset unknown_pc_type
            push ax
            call BYTE_TO_HEX
            mov si, dx
            mov [si], al
            inc si
            mov [si], ah ; в начало сообщения записывается код ошибки в
16-ричном виде
            pop ax
      final_step:
            call write
      end_of_proc:
            pop es
            pop dx
            pop ax
            ret
      GET_PC_TYPE ENDP

      GET_VERSRION PROC NEAR
```

```
            push ax
            push dx
            MOV AH,30h
            INT 21h
            ;write version number

      ;Сначала надо обработать al, а потом ah и записать в конец
System_version
                  push ax
                  push si
                  lea si, System_version
                  add si, 16
                  call BYTE_TO_DEC
                  add si, 3
                  mov al, ah
                  call BYTE_TO_DEC
                  pop si
                  pop ax
      ;OEM
            mov al, bh
            lea si, OEM
            add si, 7
            call BYTE_TO_DEC

      ;user_serial_number
            mov al, bl
            call BYTE_TO_HEX ;
            lea di, user_serial_number
            add di, 20
            mov [di], ax
            mov ax, cx
            lea di, user_serial_number
            add di, 25
            call WRD_TO_HEX

      version_:
            mov dx, offset System_version
            call WRITE
      get_OEM:
            mov dx, offset OEM
            call write
      get_user_serial_number:
            mov dx, offset user_serial_number
            call write
      end_of_proc_2:
            pop dx
            pop ax
            ret
      GET_VERSRION ENDP

      MAIN: ;функция main
            call GET_PC_TYPE
            call GET_VERSRION
      ;выход в ДОС
            xor AL,AL
            mov AH,4Ch
            int 21H

      TESTPC ENDS
               END START
```

```
        end
```

Название файла: lr1_exe.asm

```
DOSSEG
.model small
.stack 100h

;ДАННЫЕ
.data
pc_type_string DB 'PC type - ', '$'
pc_type_1 DB 'PC', 0DH, 0AH, '$'
pc_type_2 DB 'PC/XT', 0DH, 0AH, '$'
pc_type_3 DB 'AT', 0DH, 0AH, '$'
pc_type_4 DB 'PS2 model 30', 0DH, 0AH, '$'
pc_type_5 DB 'PS2 model 50 or 60', 0DH, 0AH, '$'
pc_type_6 DB 'PS2 model 80', 0DH, 0AH, '$'
pc_type_7 DB 'PCjr', 0DH, 0AH, '$'
pc_type_8 DB 'PC Convertible', 0DH, 0AH, '$'
unknown_pc_type DB '     error. Unknown', 0DH, 0AH ,'$'
System_version DB 'System version:  .  ',0DH, 0AH, '$'
OEM DB 'OEM:   ', 0DH, 0AH, '$'
user_serial_number DB 'Serial user number:       ',0DH, 0AH, '$'


.code
START:
     jmp BEGIN
;Представление 4 бита регистра al в виде цифры 16ой с.с. и
представление её в символьном виде.
TETR_TO_HEX PROC near
     and AL,0Fh
     cmp AL,09
     jbe NEXT
     add AL,07
NEXT:
     add AL,30h ; результат в al
     ret
TETR_TO_HEX ENDP

;Представление al как два числа в 16-ой с.с. и перемещение их в ax
BYTE_TO_HEX PROC near
     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

; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
      call BYTE_TO_HEX
      mov [DI],AH
      dec DI
      mov [DI],AL
      pop BX
      ret
WRD_TO_HEX ENDP

; перевод в 10 с/с. SI - адрес поля младшей цифры
BYTE_TO_DEC PROC near
      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

WRITE PROC NEAR
      push ax
      mov   AH, 9
     int   21h  ; Вызов функции DOS по прерыванию
      pop ax
     ret
WRITE ENDP

GET_PC_TYPE PROC NEAR
      push ax
      push dx
      push es
      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEh]
      mov dx, offset pc_type_string
      call WRITE

      cmp al, 0FFh
      je pc_type_1_case
      cmp al, 0FEh
      je pc_type_2_case
      cmp al, 0FBh
      je pc_type_2_case
```

```
      cmp al, 0FCh
      je pc_type_3_case
      cmp al, 0FAh
      je pc_type_4_case
      cmp al, 0FCh
      je pc_type_5_case
      cmp al, 0F8h
      je pc_type_6_case
      cmp al, 0FDh
      je pc_type_7_case
      cmp al, 0F9h
      je pc_type_8_case
      jmp unknown_pc_type_case

pc_type_1_case:
      mov dx, offset pc_type_1
      jmp final_step
pc_type_2_case:
      mov dx, offset pc_type_2
      jmp final_step
pc_type_3_case:
      mov dx, offset pc_type_3
      jmp final_step
pc_type_4_case:
      mov dx, offset pc_type_4
      jmp final_step
pc_type_5_case:
      mov dx, offset pc_type_5
      jmp final_step
pc_type_6_case:
      mov dx, offset pc_type_6
      jmp final_step
pc_type_7_case:
      mov dx, offset pc_type_7
      jmp final_step
pc_type_8_case:
      mov dx, offset pc_type_8
      jmp final_step

unknown_pc_type_case:
      mov dx, offset unknown_pc_type
      push ax
      call BYTE_TO_HEX
      mov si, dx
      mov [si], al
      inc si
      mov [si], ah
      pop ax
final_step:
      call write
end_of_proc:
      pop es
      pop dx
      pop ax
      ret
GET_PC_TYPE ENDP

GET_VERSRION PROC NEAR
      push ax
      push dx
```

```
        MOV AH,30h
        INT 21

    ;Сначала надо обработать al - xx, а потом ah - yy и записать в
System_version
                push ax
                push si
                lea si, System_version
                add si, 16
                call BYTE_TO_DEC
                add si, 3
                mov al, ah
                call BYTE_TO_DEC
                pop si
                pop ax
    ;OEM
        mov al, bh
        lea si, OEM
        add si, 7
        call BYTE_TO_DEC

    ;get_user_serial_number
        mov al, bl
        call BYTE_TO_HEX
        lea di, user_serial_number
        add di, 20
        mov [di], ax
        mov ax, cx
        lea di, user_serial_number
        add di, 25
        call WRD_TO_HEX

    version_:
        mov dx, offset System_version
        call WRITE
    get_OEM:
        mov dx, offset OEM
        call write
    get_user_serial_number:
        mov dx, offset user_serial_number
        call write
    end_of_proc_2:
        pop dx
        pop ax
        ret
    GET_VERSRION ENDP

    BEGIN:
        mov ax, @data
        mov ds, ax
        call GET_PC_TYPE
        call GET_VERSRION
        ;выход в ДОС
        xor AL,AL
        mov AH,4Ch
        int 21H
    END START
    ; КОНЕЦ МОДУЛЯ
```