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 _____ Фоминенко А.Н.

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.

Последовательность действий программы.

1) Программа получает информацию о типе пк из предпоследнего байта ROM BIOS по адресу 0F000:0FFFEh.
2) В зависимости от значения байта определяется тип, далее он выводится на экран с помощью процедуры PRINT. В случае, если значение не совпадает с табличными, то выводится значение предпоследнего байта ROM BIOS.
3) Программа узнаёт версию ОС, серийные номера OEM и пользователя с помощью вызова функции 30h прерывания 21h. Данные заносятся в соответствующие строки и выводятся на экран.
4) Программа завершает свою работу.

Ход работы.

1) На основе шаблона, приведённого в методических указаниях, был написан текст исходного .COM модуля. Далее с помощью транслятора masm.exe и компоновщика link.exe был скомпилирован плохой .EXE модуль. При помощи exe2bin.exe по плохому .EXE модулю был построен хороший .COM модуль.
2) Был написан текст исходного .EXE модуля. При помощи транслятора masm.exe и компоновщика link.exe был скомпилирован хороший .EXE модуль.
3) Проведено сравнение исходных текстов lab1_com.asm и lab1_exe.asm.

4) При помощи программы FAR были открыты в шестнадцатеричном виде файлы загрузочных модулей lab1_com.com, lab1_com.exe и lab1_exe.exe.
5) Был исследован загрузочный модуль .COM при помощи отладчика.
6) Был исследован загрузочный модуль .EXE при помощи отладчика.

Функции программ.

Названия функций	Описание
TETR_TO_HEX	Перевод десятичной цифры в код символа.
BYTE_TO_HEX	Перевод байта 16-ной с.с. в символьный код.
WRD_TO_HEX	Перевод слова 16-ной с.с. в символьный код.
BYTE_TO_DEC	Перевод байта 16-ной с.с. в символьный код в 10-ной с.с.
PRINT	Вывод строки.

Ответы на контрольные вопросы.

1) Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

   Один сегмент.

2. EXE-программа?

Любое количество сегментов. Сегменты кода, данных и стека находятся отдельно.

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

ORG 100H - устанавливает значение программного счётчика в 100h, так как при загрузке COM- файла в память DOS занимает первые 256 байтов блоком данных PSP и располагает код программы после этого блока. Все программы, которые компилируются в файлы типа COM должны начинаться с этой директивы.

Директива ASSUME указывает ассемблеру, с каким сегментом или группой сегментом связан тот или иной сегментный регистр. Она не изменяет значений сегментных регистров, а только позволяет ассемблеру проверять допустимость ссылок и самостоятельно вставлять префиксы переопределения сегментов при необходимости. Без данной директивы программа не скомпилируется, так как ассемблер не будет знать, относительно чего происходит смещение меток.

Директива END, которой завершается любая программа на ассемблере.

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

Нет. Так как в COM программах в DOS отсутствует таблица настроек, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, нельзя использовать команды, связанные с адресом сегмента.

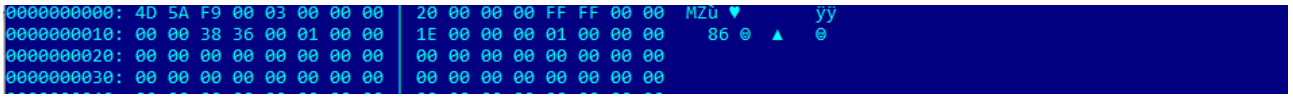
2) Отличия форматов файлов COM и EXE модулей.

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

В COM файле код, данные и стек располагаются в одном сегменте. Код, как и данные начинается с адреса 0h. Файл в 16-ом виде:

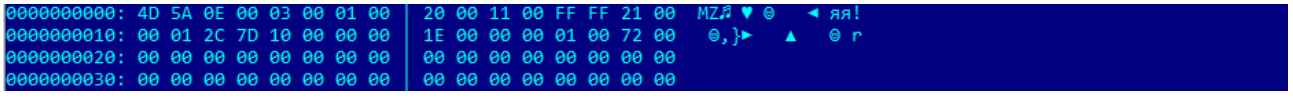
2. Какова структура файла плохого EXE? С какого адреса располагается код? Что располагается с адреса 0?

В плохом EXE файле код, данные и стек располагаются в одном сегменте. Данные и код начинаются с адреса 300h. С адреса 0h находится управляющая информация для загрузчика, которая содержит заголовок и таблицу настройки адресов. Это можно проверить, открыв файл в 16-ом виде:



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

В хорошем EXE-файле код, данные и стек находятся в различных сегментах, в отличие от плохого, где они находятся в одном сегменте. Также изменён порядок расположения сегментов в памяти, и стек имеет другой размер - 256 байт(100h). В отличии от плохого, хороший EXE-файл не содержит директивы ORG 100h, которая выделяет память под PSP. Поэтому код начинается с адреса 210h. Файл в 16-ом виде:



3) Загрузка COM модуля в основную память.

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

   При загрузке COM файла в память DOS занимает первые 256 байт блоком данных PSP. Код располагается с адреса 100h. После загрузки COM-программы в память сегментные регистры указывают на начало PSP.

2. Что располагается с адреса 0?

   С адреса 0 располагается префикс программного сегмента PSP.

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

   Регистры DS, ES, CS, SS имеют одно значение 119C, так как указывают на один и тот же блок PSP.

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

Стек создаётся автоматически. Регистр SS указывает на начало блока PSP(0h), а SP указывает на конец модуля (FFF5h). То есть стек располагается между адресами SS:0000h и SS:FFF5h и заполняется с конца модуля в сторону уменьшения адресов.

4) Загрузка хорошего EXE модуля в основную память.

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

Сначала создаётся PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Далее загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память. К полю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров. В результате DS и ES указывают на начало PSP(119C), CS - на начало сегмента команд (11AC), а SS - на начало сегмента стека (11CD). Управление передаётся загруженной задаче по адресу, указанному в заголовке.

2. На что указывают регистры DS и ES?

Данные регистры указывают на начало блока PSP.

3. Как определяется стек?

Регистр SS указывает на начало сегмента стека, а SS:SP - на конец сегмента стека.

4. Как определяется точка входа?

Точка входа определяется параметром после директивы END, в качестве

которого нужно передать метку, с которой программа начнёт выполнение команд.

Вывод.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1_com.asm

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN
; Данные
Type_PC          db 'PC type - PC',0DH,0AH,'$'
Type_XT          db 'PC type - XT',0DH,0AH,'$'
Type_AT          db 'PC type - AT',0DH,0AH,'$'
Type_PS_30       db 'PC type - PS2 model 30',0DH,0AH,'$'
Type_PS_80       db 'PC type - PS2 model 80',0DH,0AH,'$'
Type_PCjr        db 'PC type - PCjr',0DH,0AH,'$'
Type_PCConvert   db 'PC type - PC Convertible',0DH,0AH,'$'
Type_Unknown     db '  ', 0DH, 0AH, '$'

OS_Version_Num   db 'OS version number -   .  ',0DH,0AH,'$'
Num_OEM          db 'OEM number -   ',0DH,0AH,'$'
User_series_num  db '24-bit user series 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 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 цифра
            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  PROC NEAR    ; вывод строки на экран
      push ax
      mov ah, 9h
      int 21H
      pop ax
      ret
PRINT ENDP
;--------------------------------
; КОД
BEGIN:
      ;получаем информацию о типе пк
      mov ax, 0F000h
      mov es, ax
```

```
	mov al, es:[0FFFEh]
	;сравниваем полученное значение с табличными значениями
	cmp al, 0FFh
	je  type_1
	cmp al, 0FEh
	je  type_1
	cmp al, 0FBh
	je  type_2
	cmp al, 0FAh
	je  type_3
	cmp al, 0FCh
	je  type_4
	cmp al, 0F8h
	je  type_5
	cmp al, 0FDh
	je  type_6
	cmp al, 0F9h
	je  type_7
	jmp print_type
;выбираем строку соответствующую типу пк
type_1:
	mov dx, offset Type_PC
	jmp  print_type
type_2:
	mov dx, offset Type_XT
	jmp  print_type
type_3:
	mov dx, offset Type_AT
	jmp  print_type
type_4:
	mov dx, offset Type_PS_30
	jmp  print_type
type_5:
	mov dx, offset Type_PS_80
	jmp  print_type
type_6:
	mov dx, offset Type_PCjr
	jmp  print_type
type_7:
	mov dx, offset Type_PCConvert
	jmp  print_type
unknown:
	call BYTE_TO_HEX
	mov bx, offset Type_Unknown
	mov [bx], al
	mov [bx+1], ah
	mov dx, bx
print_type:
	;печатаем тип пк
	call PRINT
	;узнаём информацию о версии ОС и серийные номера OEM и
```

```
пользователя
      mov ah, 30h
      int 21h
      mov si, offset OS_Version_Num + 21
      mov dl, ah
      call BYTE_TO_DEC
      mov al, dl
      add si, 3
      call BYTE_TO_DEC
      ; Выводим на экран версию ОС
      mov dx, offset OS_Version_Num
      call PRINT
      mov al, bh
      call BYTE_TO_HEX
      mov di, offset Num_OEM + 13
      mov [di], al
      mov [di+1], ah
      ; Выводим на экран серийный номер OEM
      mov dx, offset Num_OEM
      call PRINT
      mov al, bl
      call BYTE_TO_HEX
      mov di, offset User_series_num + 29
      mov [di], al
      mov [di+1], ah
      mov ax, cx
      add di, 5
      call WRD_TO_HEX
      ; Выводим на экран серийный номер пользователя
      mov dx, offset User_series_num
      call PRINT
; Выход в DOS
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC  ENDS
        END START ; Конец модуля, START - точка входа
```

Имя файла: lab1_exe.asm

```
DOSSEG
.model small
.stack 100h

.data
Type_PC          db 'PC type - PC',0DH,0AH,'$'
Type_XT          db 'PC type - XT',0DH,0AH,'$'
Type_AT          db 'PC type - AT',0DH,0AH,'$'
Type_PS_30       db 'PC type - PS2 model 30',0DH,0AH,'$'
Type_PS_80       db 'PC type - PS2 model 80',0DH,0AH,'$'
```

```
Type_PCjr        db 'PC type - PCjr',0DH,0AH,'$'
Type_PCConvert   db 'PC type - PC Convertible',0DH,0AH,'$'
Type_Unknown     db '  ', 0DH, 0AH, '$'

OS_Version_Num   db 'OS version number -   .  ',0DH,0AH,'$'
Num_OEM          db 'OEM number -   ',0DH,0AH,'$'
User_series_num  db '24-bit user series number -       ',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 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 цифра
            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  PROC NEAR    ; вывод строки на экран
      push ax
      mov ah, 9h
      int 21H
      pop ax
      ret
PRINT ENDP
;-------------------------------
; КОД
BEGIN:
      mov ax, @data
      mov ds, ax
      ;получаем информацию о типе пк
      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEh]
      ;сравниваем полученное значение с табличными значениями
      cmp al, 0FFh
      je  type_1
      cmp al, 0FEh
      je  type_1
      cmp al, 0FBh
      je  type_2
      cmp al, 0FAh
      je  type_3
      cmp al, 0FCh
```

```
      je  type_4
      cmp al, 0F8h
      je  type_5
      cmp al, 0FDh
      je  type_6
      cmp al, 0F9h
      je  type_7
      jmp print_type
;выбираем строку соответствующую типу пк
type_1:
      mov dx, offset Type_PC
      jmp  print_type
type_2:
      mov dx, offset Type_XT
      jmp  print_type
type_3:
      mov dx, offset Type_AT
      jmp  print_type
type_4:
      mov dx, offset Type_PS_30
      jmp  print_type
type_5:
      mov dx, offset Type_PS_80
      jmp  print_type
type_6:
      mov dx, offset Type_PCjr
      jmp  print_type
type_7:
      mov dx, offset Type_PCConvert
      jmp  print_type
unknown:
      call BYTE_TO_HEX
      mov bx, offset Type_Unknown
      mov [bx], al
      mov [bx+1], ah
      mov dx, bx
print_type:
      ;печатаем тип пк
      call PRINT
      ;узнаём информацию о версии ОС и серийные номера OEM и
пользователя
      mov ah, 30h
      int 21h
      mov si, offset OS_Version_Num + 21
      mov dl, ah
      call BYTE_TO_DEC
      mov al, dl
      add si, 3
      call BYTE_TO_DEC
      ; Выводим на экран версию ОС
      mov dx, offset OS_Version_Num
```

```
        call PRINT
        mov al, bh
        call BYTE_TO_HEX
        mov di, offset Num_OEM + 13
        mov [di], al
        mov [di+1], ah
        ; Выводим на экран серийный номер OEM
        mov dx, offset Num_OEM
        call PRINT
        mov al, bl
        call BYTE_TO_HEX
        mov di, offset User_series_num + 29
        mov [di], al
        mov [di+1], ah
        mov ax, cx
        add di, 5
        call WRD_TO_HEX
        ; Выводим на экран серийный номер пользователя
        mov dx, offset User_series_num
        call PRINT
; Выход в DOS
          xor AL,AL
          mov AH,4Ch
          int 21H
END START ; Конец модуля, START - точка входа
```