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. 9382 _____ Кодуков А.В.

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.

Выполнение работы:

Функции:

- WRITE – печать
- DOSVER – определение версии системы
- PCTYPE – определение типа PC

Утилита последовательно составляет строки и выводит на экран:

версию DOS

1. номер версии
2. номер модификации
3. серийный номер OEM
4. серийный номер пользователя

тип PC

Контрольные вопросы по лабораторной работе №1

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?
   Один сегмент команд
2) EXE-программа?
   Несколько сегментов (>= 1)
3) Какие директивы должны обязательно быть в тексте COM-программы?
   org 100h
4) Все ли форматы команд можно использовать в COM-программе?
   Нельзя обращаться к сегментам, там как сегментные регистры не определяются на компиляции

Отличия форматов файлов COM и EXE модулей

1) Какова структура файла COM? С какого адреса располагается код?
    - PSP – 256 байт
    - Код и данные (с адреса 100h)
2) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?

    Начало:PSP и пропуск 100h, код и данные начинаются с 300h

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

    Разделены сегменты данных, стека и кода, нет пропсука 100h

Загрузка COM модуля в основную память

1) Какой формат загрузки модуля COM? С какого адреса располагается код?
    - Определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы
    - Создаётся блок памяти для PSP и программы
    - Загружается COM-файл с адреса 100h
    - Сегментные регистры CS, DS, ES, SS устанавливаются на начало PSP (0h)
    - Регистр SP устанавливается на конец PSP (FFh)
    - В стек записывается значение 0000
    - В регистр IP записывается значение 100h
2) Что располагается с адреса 0?

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

    Все регистры указывают на начало PSP
4) Как определяется стек? Какую область памяти он занимает? Какие адреса?

Значение регистра SP устанавливается так, чтобы он указывал на последнюю доступную в сегменте ячейку памяти. Таким образом программа занимает начало, а стек - конец сегмента

Загрузка «хорошего» EXE модуля в основную память

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

   DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

2) На что указывают регистры DS и ES?

   DS и ES указывают на начало PSP

3) Как определяется стек?

   Стековый сегмент определен в программе

4) Как определяется точка входа?

   директивой end

Вывод:

В ходе работы был изучен принцип работы исполняемого файла .com, а также выявлены его отличия от .exe файла с точки зрения работы, выделения памяти и исходного кода.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

bad.asm

```
TESTPC     SEGMENT
           ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
               org 100h
START:     JMP     BEGIN
; ДАННЫЕ
VERSION db "DOS version:  $"
MODNUM db 13, 10, "Modification number:  $"
SERIAL db 13, 10, "Serial number:       $" ; 16 symbols
OEM db 13, 10, "OEM:     $"; 6 sybmols

TYPESTR db 13, 10, "PC type: $"
TYPEPC db "PC$"
TYPEPCXT db "PC/XT$"
TYPEAT db "AT$"
TYPEPS2M30 db "PS2 (30 model)$"
TYPEPS2M5060 db "PS2 (50 or 60 model)$"
TYPEPS2M80 db "PS2 (80 model)$"
TYPEPCJR db "PC jr$"
TYPEPCCONV db "PC Convertible$"

;ПРОЦЕДУРЫ
;------------------------------
WRITE PROC near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
WRITE ENDP
;------------------------------
DOSVER PROC near
      mov ah, 30h
      int 21h
      ; al - version number
      ; ah - modification number
      ; bh - OEM serial number
      ; bl:cx - user serial number
      push ax

      mov si, offset VERSION
      add si, 12
      call BYTE_TO_DEC
      mov dx, offset VERSION
      call WRITE

      mov si, offset MODNUM
      add si, 23
      pop ax
      mov al, ah
      call BYTE_TO_DEC
      mov dx, offset MODNUM
      call WRITE

      mov si, offset OEM
      add si, 9
      mov al, bh
      call BYTE_TO_DEC
      mov dx, offset OEM
```

```
      call WRITE

      mov di, offset SERIAL
      add di, 22
      mov ax, cx
      call WRD_TO_HEX
      mov al, bl
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset SERIAL
      call WRITE
      ret
DOSVER ENDP
;-------------------------------
PCTYPE PROC near
      mov ax, 0f000h
      mov es, ax
      mov al, es:[0fffeh]
      mov dx, offset TYPESTR
      call WRITE
      cmp al, 0ffh
      jz pc
      cmp al, 0feh
      jz pcxt
      cmp al, 0fbh
      jz pcxt
      cmp al, 0fch
      jz pcat
      cmp al, 0fah
      jz pcps2m30
      cmp al, 0f8h
      jz pcps2m80
      cmp al, 0fdh
      jz pcjr
      cmp al, 0f9h
      jz pcconv
      pc:
            mov dx, offset TYPEPC
            jmp writestring
      pcxt:
            mov dx, offset TYPEPCXT
            jmp writestring
      pcat:
            mov dx, offset TYPEAT
            jmp writestring
      pcps2m30:
            mov dx, offset TYPEPS2M30
            jmp writestring
      pcps2m5060:
            mov dx, offset TYPEPS2M5060
            jmp writestring
      pcps2m80:
            mov dx, offset TYPEPS2M80
            jmp writestring
      pcjr:
            mov dx, offset TYPEPCJR
            jmp writestring
      pcconv:
            mov dx, offset TYPEPCCONV
            jmp writestring
      writestring:
            call WRITE
```

```
      ret
PCTYPE ENDP
;-------------------------------
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_1:     pop      DX
           pop      CX
           ret
BYTE_TO_DEC    ENDP
;--------------------------------
; КОД
BEGIN:
           call DOSVER
         call PCTYPE
         mov ah, 10h
         int 16h
           ; Выход в DOS
           xor     AL,AL
           mov     AH,4Ch
           int     21H

TESTPC    ENDS
END       START     ;конец модуля, START - точка входа
```

good.asm

```
.model small
.data
VERSION db "DOS version:  $"
MODNUM db 13, 10, "Modification number:  $"
SERIAL db 13, 10, "Serial number:       $" ; 16 symbols
OEM db 13, 10, "OEM:     $"; 6 sybmols

TYPEPCSTRING db 13, 10, "PC type: $"
TYPEPC db "PC$"
TYPEPCXT db "PC/XT$"
TYPEAT db "AT$"
TYPEPS2M30 db "PS2 (30 model)$"
TYPEPS2M5060 db "PS2 (50 or 60 model)$"
TYPEPS2M80 db "PS2 (80 model)$"
TYPEPCJR db "PC jr$"
TYPEPCCONV db "PC Convertible$"

.stack 100h

.code
;--------------------------------
WRITE PROC near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
WRITE ENDP
;--------------------------------
DOSVER PROC near
      mov ah, 30h
      int 21h
      ; al - version number
      ; ah - modification number
      ; bh - OEM serial number
      ; bl:cx - user serial number
      push ax

      mov si, offset VERSION
      add si, 12
      call BYTE_TO_DEC
```

```
      mov dx, offset VERSION
      call WRITE

      mov si, offset MODNUM
      add si, 23
      pop ax
      mov al, ah
      call BYTE_TO_DEC
      mov dx, offset MODNUM
      call WRITE

      mov si, offset OEM
      add si, 9
      mov al, bh
      call BYTE_TO_DEC
      mov dx, offset OEM
      call WRITE

      mov di, offset SERIAL
      add di, 22
      mov ax, cx
      call WRD_TO_HEX
      mov al, bl
      call BYTE_TO_HEX
      sub di, 2
      mov [di], ax
      mov dx, offset SERIAL
      call WRITE
      ret
DOSVER ENDP
;--------------------------------
PCTYPE PROC near
      mov ax, 0f000h
      mov es, ax
      mov al, es:[0fffeh]
      mov dx, offset TYPEPCSTRING
      call WRITE
      cmp al, 0ffh
      jz pc
      cmp al, 0feh
      jz pcxt
      cmp al, 0fbh
      jz pcxt
      cmp al, 0fch
      jz pcat
      cmp al, 0fah
      jz pcps2m30
      cmp al, 0f8h
      jz pcps2m80
      cmp al, 0fdh
      jz pcjr
      cmp al, 0f9h
      jz pcconv
      pc:
            mov dx, offset TYPEPC
            jmp writestring
      pcxt:
            mov dx, offset TYPEPCXT
            jmp writestring
      pcat:
            mov dx, offset TYPEAT
            jmp writestring
      pcps2m30:
```

```
            mov dx, offset TYPEPS2M30
            jmp writestring
      pcps2m5060:
            mov dx, offset TYPEPS2M5060
            jmp writestring
      pcps2m80:
            mov dx, offset TYPEPS2M80
            jmp writestring
      pcjr:
            mov dx, offset TYPEPCJR
            jmp writestring
      pcconv:
            mov dx, offset TYPEPCCONV
            jmp writestring
      writestring:
            call WRITE
      ret
PCTYPE ENDP
;-------------------------------
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 ;PI AL CЃC,P°CЪC€P°CЏ C†PёC„CЪP°
           pop      CX          ;PI AH PjP»P°PґC€P°CЏ
           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:
          mov ax, @data
          mov ds, ax
          call DOSVER
          call PCTYPE
          mov ah, 10h
          int 16h
            ; Выход в DOS
            xor     AL,AL
            mov     AH,4Ch
            int     21H

END       BEGIN     ;конец модуля
```