

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ЛАБОРАТОРНАЯ РАБОТА № 1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей.

Студентка гр. 7381

Алясова А.Н.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

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

Ход работы.

1) На основе шаблона, приведенного в методических указаниях, написали текст исходного .COM модуля, который определяет тип PC и версию системы. Получили “хороший” .COM модуль и “плохой” .EXE модуль. Результаты работы программ представлен ниже.

Файл com.asm

Получили com.com

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

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

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

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

Получили com.exe

```
C:\>masm com.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [com.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

    49978 + 457282 Bytes symbol space free

    0 Warning Errors
    0 Severe Errors

C:\>link com.obj

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

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

```

C:\>com.exe
FC
5 0 255 000000

0¼ PC Type:
5 0 255 000000

0¼ PC Type:
255 000000

0¼ PC Type:
000000

0¼ PC Type:

```

2) Написали текст, построили и отладили исходный .EXE модуль, который выполняет те же функции, что и модуль .COM. Таким образом, получили “хороший” .EXE модуль. Результат работы программы представлен ниже.

Файл exe.asm

Получили exe.exe

```

C:\>masm exe.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [exe.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

49978 + 457282 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\>link exe.obj

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

Run File [EXE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\>exe.exe
PC Type: FC
Modification number: 5.0
OEM: 0
Serial Number: 000000

```

Функции программ

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

Ответы на контрольные вопросы:

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один сегмент, в котором находятся код и данные.

2. EXE программа?

EXE-программа предполагает отдельные сегменты для кода, данных и стека.

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

Директива `ORG 100h`, которая задает смещение для всех адресов программы на 256 байт для префикса программного сегмента (PSP).

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

Нет, только команды типа `Tiny` (вызовы `near`). Загрузить адрес сегмента не представляется возможным (так как в `.COM`-программах отсутствуют таблицы настроек), значит нельзя использовать команды, связанные с адресом сегмента.

Отличия форматов файлов COM и EXE модулей

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

`.COM`-файл состоит из команд, процедур и данных, используемых в программе. Код начинается с 0 адреса.

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

В файле EXE содержится информация для загрузчика, данные и код. С 0 адреса располагается таблица настроек (информация для загрузчика). Код начинается с адреса `300h`.

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

В отличие от плохого, хороший EXE-файл не содержит директивы `ORG 100h` (которая выделяет память под PSP), поэтому код начинается с адреса `200h`.

EXE-файл состоит из информации для загрузчика, сегмента стека, сегмент данных и сегмент кода. В плохом был 1 сегмент.

Загрузка COM модуля в основную память

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

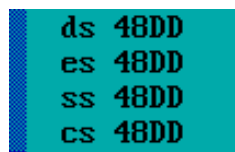
После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.

2. Что располагается с 0 адреса?

С адреса 0 располагается префикс программного сегмента (PSP).

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

Сегментные регистры имеют значения, которые соответствуют сегменту, в который модуль был помещен управляющей программой. Все они указывают на один и тот же сегмент памяти, поэтому все регистры имеют значения 48DD. Они указывают на PSP.



```
ds 48DD
es 48DD
ss 48DD
cs 48DD
```

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

Стек создается автоматически, указатель стека в конце сегмента. Он занимает оставшуюся память и адреса изменяются от больших к меньшим, то есть от FFFh к 0000h.

Загрузка «хорошего» EXE модуля в память

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

Сначала создается PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. 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. DS и ES указывают на начало PSP (48DD), CS – на начало сегмента команд (4932), а SS – на начало сегмента стека (48ED).

```
ds 48DD
es 48DD
ss 48ED
cs 4932
```

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

3. Как определяется стек?

Стек определяется при объявлении сегмента стека, в котором указывается, сколько памяти необходимо выделить.

4. Как определяется точка входа?

С помощью директивы END, операндом которой является адрес, с которого начинается выполнение программы.

ПРИЛОЖЕНИЕ А

КОДЫ ИСХОДНЫХ ПРОГРАММ

com.asm

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

;ДАННЫЕ
PC_Type      db    'PC Type:  ', 0dh, 0ah, '$'
Mod_number   db    'Modification number:  .  ', 0dh, 0ah, '$'
OEM          db    'OEM:  ', 0dh, 0ah, '$'
S_number     db    'Serial Number:      ', 0dh, 0ah, '$'

;ПРОЦЕДУРЫ
;-----
;печатать строки
PRINT_STRING PROC near
    mov     ah, 09h
    int     21h
    ret
PRINT_STRING ENDP

;-----
;перевод десятичной цифры в код символа
TETR_TO_HEX  PROC near
    and     al, 0fh ;логическое умножение всех пар битов
    cmp     al, 09
    jbe     NEXT ;Переход если ниже или равно
    add     al, 07
NEXT: add     al, 30h
    ret
TETR_TO_HEX  ENDP

;-----
;перевод байта 16 с.с в символьный код
;байт в AL переводится в два символа шестнадцетиричного числа в AX
BYTE_TO_HEX  PROC near
    push    cx
    mov     al, ah
    call    TETR_TO_HEX
    xchg    al, ah
```



```

        mov     cl, 4
        shr     al, cl ;логический сдвиг вправо
        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
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX      ENDP

```

```

;-----
;перевод байта 16 с.с в символьный код 10 с.с
;si - адрес поля младшей цифры

```

```

BYTE_TO_DEC      PROC near
        push    cx
        push    dx
        push    ax
        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      ax, 00h
                jbe      end_l
                or        al, 30h
                mov      [si], al
end_l:          pop      ax
                pop      dx
                pop      cx
                ret
BYTE_TO_DEC      ENDP

```

;-----

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 на количество символов
pop  ax
pop  bx
pop  es

mov  ah, 30h
int  21h

;Mod_numb
push ax
push si
lea  si, Mod_numb
add  si, 21
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

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

;Вывод
lea   dx, PC_Type
call  PRINT_STRING
lea   dx, Mod_numb
call  PRINT_STRING
lea   dx, OEM
call  PRINT_STRING
lea   dx, S_numb
call  PRINT_STRING

;Выход в dos
xor   al, al
mov  ah, 4ch
int   21h
ret

TESTPC  ENDS
END  START

```

exe.asm

```
DOSSEG
AStack    SEGMENT  STACK
           DW 512 DUP(?)
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
```

```
;-----
```

```
;печать строки
```

```
PRINT_STRING PROC near
           mov  ah, 09h
           int  21h
           ret
PRINT_STRING ENDP
```

```
;-----
```

```
;перевод десятичной цифры в код символа
```

```
TETR_TO_HEX PROC near
           and  al, 0fh ;логическое умножение всех пар битов
           cmp  al, 09
           jbe  NEXT ;Переход если ниже или равно
           add  al, 07
NEXT: add    al, 30h
           ret
TETR_TO_HEX ENDP
```

```
;-----
```

```
;перевод байта 16 с.с в символьный код
```

```
;байт в AL переводится в 2 символа шестнадцетиричного числа в AX
```

```

BYTE_TO_HEX          PROC near
    push cx
    mov     al, ah
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl ;логический сдвиг вправо
    call    TETR_TO_HEX
    pop     cx
    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
    xor     ah, ah
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    pop     bx
    ret
WRD_TO_HEX          ENDP

```

```

;-----
;перевод байта 16 с.с в символьный код 10 с.с
;si - адрес поля младшей цифры
BYTE_TO_DEC          PROC near
    push cx
    push dx
    push ax
    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     ax, 00h
        jbe     end_1
        or      al, 30h
        mov     [si], al
end_1:  pop     ax
        pop     dx
        pop     cx
        ret
BYTE_TO_DEC      ENDP

```

```

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 на количество символов
        pop     ax
        pop     bx
        pop     es

        mov     ah, 30h
        int     21h

```

```

;Mod_numb
push ax
push si
lea      si, Mod_numb
add      si, 21
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

;S_Numb
mov  al, bl
call BYTE_TO_HEX
lea  di, S_numb
add  di, 15
mov  [di], ax
mov  ax, cx
lea  di, S_numb
add  di, 20
call WRD_TO_HEX

lea  dx, PC_Type
call PRINT_STRING
lea  dx, Mod_numb
call PRINT_STRING
lea  dx, OEM
call PRINT_STRING
lea  dx, S_numb
call PRINT_STRING

;выход в dos
xor  al, al
mov  ah, 4ch
int  21h
ret

```

```
CODE ENDS  
      END    main
```