

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. 7381

Минуллин М.А.

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.

Написан текст исходного .COM модуля, который определяет тип ПК и версию системы. За основу взят шаблон, приведённый в разделе «Основные сведения». Ассемблерная программа читает содержимое последнего байта ROM BIOS, сравнивая коды по таблице, определять тип ПК и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран в виде соответствующего сообщения («UKWN»). Затем определяется версия системы. Ассемблерная программа по значениям регистров AL и AH формирует текстовую строку в формате XX.YY, где XX – номер основной версии, а YY – номер модификации в десятичной системе счисления, формирует строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводят на экран.

Модуль отлажен. Получен «хороший» .COM модуль, а также «плохой» .EXE модуль из того же исходного текста. Результаты выполнения программ представлены на рис. 1, 2.

Написан текст исходного .EXE модуля, выполняющий те же самые функции, что и .COM модуль. Модуль также отлажен, полученный файл является «хорошим» .EXE модулем.

Исходные тексты «хороших» .COM и .EXE просмотрены. Ниже даны ответы на контрольные вопросы по темам «Отличия исходных текстов .COM и .EXE программ» и «Отличия форматов файлов .COM и .EXE модулей».

Используя отладчик TD.EXE и .COM программу, найдены ответы на вопросы по теме «Загрузка .COM модуля в основную память», которые представлены ниже.

Необходимые сведения.

Тип ПК хранится в байте по адресу 0F000:0FFFFh в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

PC	FF
PC/XT	FE, FB
AT	FC
PS2/30	FA
PS2/50-60	FC
PS2/80	F8
PCjr	FD
PC Convertible	F9

Для определения версии MS DOS следует воспользоваться функцией 30h прерывания 21h. Входным параметром является номер функции в АН.

Выходными параметрами являются:

1. AL – номер основной версии. Если 0, то версия < 2.0.
2. AH – номер модификации.
3. BH – серийный номер OEM (Original Equipment Manufacturer).
4. BL:CX – 24-битовый серийный номер пользователя.

```
C:\>TASM.EXE 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:  471k

C:\>TLINK.EXE /t COM.OBJ
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>COM.COM
PC Type: AT
Modify number: 5.0
OEM Code: 255
User Serial Number: 000000
```

Рисунок 1 – результат работы «хорошего» .COM модуля.

```

C:\>TLINK.EXE COM.OBJ
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
Warning: No stack

C:\>COM.EXE

                                0>PC Type: Unknown
                                000000
5 0                                255
0>PC Type: Unknown
                                000000
255                                0>PC
C Type: Unknown
                                000000
                                0>PC Type: Unknown

```

Рисунок 2 – результат работы «плохого» .EXE модуля.

```

C:\>TASM.EXE EXE.ASM
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file:   EXE.ASM
Error messages:   None
Warning messages: None
Passes:           1
Remaining memory: 471k

C:\>TLINK.EXE EXE.OBJ
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>EXE.EXE
PC Type: AT
Modify number: 5.0
DEM Code: 0
User Serial Number: 000000

```

Рисунок 3 – результат работы «хорошего» .EXE модуля.

Отличия исходных текстов .COM и .EXE программ.

В: Сколько сегментов должна содержать .COM-программа?

О: .COM-программа содержит всего один сегмент, в котором вместе находятся и данные и код.

В: Сколько сегментов должна содержать .EXE-программа?

О: .EXE-программа может содержать произвольное число сегментов данных, но в целом предполагается наличие обязательных сегментов стека и кода.

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

О: Обязательным являются директивы `ORG 100h`, которая задаёт смещение всех адресов программы в 256 байт (связано с расположением программы в памяти и префикса программного сегмента), а также директива `ASSUME`, используемая для того, чтобы указать соответствия между сегментными регистрами и программными сегментами.

В: Все ли форматы команд можно использовать в `.COM`-программе?

О: Поскольку в `.COM` программе предполагается использование только одного сегмента, то разрешено использовать только `near`-переходы (в пределах одного сегмента). Использование `far`-переходов запрещено (они предполагают переходы между сегментами). Запрещено использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки не известен (в `.COM`-программах в `DOS` не содержится таблица настройки, содержащая описание адресов, зависящих от размещения загрузочного модуля).

Отличия форматов файлов `.COM` и `.EXE` модулей.

В: Какова структура `.COM`-файла? С какого адреса располагается код?

О: `.COM`-файл состоит из команд, процедур и данных, используемых в программе. Сам код начинается с нулевого адреса.

В: Какова структура «плохого» `.EXE`-файла? С какого адреса располагается код? Что располагается с нулевого адреса?

В `.EXE`-файле данные и код содержатся в одном сегменте. С нулевого адреса располагается подпись компоновщика, указывающая, что файл является `.EXE`-файлом. Это что-то типа дескриптора. Код начинается с адреса `300h`.

В: Какова структура «хорошего» `.EXE`-файла? Чем он отличается от «плохого» `.EXE`-файла?

О: В отличие от «плохого», «хороший» `.EXE`-файл не содержит директивы `ORG 100h` (которая выделяет память под `PSP`), поэтому код начинается с адреса `200h`. В «хорошем» `.EXE`-файле данные, стек и код разделены по сегментам.

Загрузка .COM модуля в основную память.

В: Какой формат загрузки .COM-модуля? С какого адреса располагается код?

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

В: Что располагается с нулевого адреса?

О: С нулевого адреса располагается префикс программного сегмента (PSP).

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

В: Как определяется стек? Какую область памяти он занимает? Какие адреса:

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

Загрузка «хорошего» .EXE модуля в основную память.

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

О: Сначала создаётся PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, к модулю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров. DS и ES указывают на начало PSP (42DD), CS – на начало сегмента команд (4932), а SS – на начало сегмента стека (48ED).

В: На что указывают регистры DS и ES?

О: Изначально регистры DS и ES указывают на начало сегмента PSP.

В: Как определяется стек?

О: Стек определяется при объявлении сегмента стека, в котором указывается, сколько памяти необходимо выделить. В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

В: Как определяется точка входа?

О: Точка входа определяется с помощью директивы END, операндом которой является адрес, с которого начинается выполнения программы.

Выводы.

В ходы выполнения лабораторной работы было проведено исследование различий в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Были даны ответы на предложенные в лабораторной работе вопросы. Итоговым результатом лабораторной работы стали 2 текста программы для .COM и .EXE модулей, выполняющих требования задания.

ПРИЛОЖЕНИЕ А

ТЕКСТ ИСХОДНОГО ФАЙЛА .COM МОДУЛЯ

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

TypePC_Unknown db 'PC Type: Unknown', 0dh, 0ah, '$'
TypePC_PC      db 'PC Type: PC', 0dh, 0ah, '$'
TypePC_PCXT    db 'PC Type: PC/XT', 0dh, 0ah, '$'
TypePC_AT      db 'PC Type: AT', 0dh, 0ah, '$'
TypePC_230PS   db 'PC Type: PC2 model 30', 0dh, 0ah, '$'
TypePC_280PS   db 'PC Type: PS2 model 80', 0dh, 0ah, '$'
TypePC_PCjr    db 'PC Type: PCjr', 0dh, 0ah, '$'
TypePC_PCC     db 'PC Type: PC Convertible', 0dh, 0ah, '$'

ModifyNumber   db 'Modify number:  . ', 0dh, 0ah, '$'
OEM_Code       db 'OEM Code:  ', 0dh, 0ah, '$'
UserSN         db 'User Serial Number:  ', 0dh, 0ah, '$'

PRINT_STRING PROC near
    mov     ah, 09h
    int     21h
    ret
PRINT_STRING 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
    push    cx
    mov     al, ah
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
```



```

        shr     al, cl
        call    TETR_TO_HEX
        pop     cx
        ret
BYTE_TO_HEX      ENDP

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

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:

; здесь определяем тип ПК

```
    push    es
    push    bx
    push    ax
    mov     bx, 0F000h
    mov     es, bx
```

; предпоследний байт ROM BIOS

```
    mov     ax, es:[0FFFEh]
```

; PC

```
    cmp     al, 0FFh
```

```
    je      MVPC
```

; PC/XT

```
    cmp     al, 0FEh
```

```
    je      MVPCXT
```

```
    cmp     al, 0FBh
```

```
    je      MVPCXT
```

; AT

```
    cmp     al, 0FCh
```

```
    je      MVAT
```

; PS2/30

```
    cmp     al, 0FAh
```

```
    je      MV230PS
```

; коды для AT и PS2/50-60 совпадают, поэтому не обрабатываем
(ну или что делать?)

; PS2/80

```
    cmp     al, 0F8h
```

```
    je      MV280PS
```

; PCjr

```
    cmp     al, 0FDh
```

```
    je      MVPCjr
```

; PC Convertible

```
    cmp     al, 0F9h
```

```
    je      MVPCC
```

; PC Unknown

```
    lea     dx, TypePC_Unknown
```

```
    jmp     MVEND
```

MVPC:

```

        lea     dx, TypePC_PC
        jmp     MVEND
MVPCTX:
        lea     dx, TypePC_PCXT
        jmp     MVEND
MVAT:
        lea     dx, TypePC_AT
        jmp     MVEND
MV230PS:
        lea     dx, TypePC_230PS
        jmp     MVEND
MV280PS:
        lea     dx, TypePC_280PS
        jmp     MVEND
MVPCjr:
        lea     dx, TypePC_PCjr
        jmp     MVEND
MVPCC:
        lea     dx, TypePC_PCC
MVEND:
        call    PRINT_STRING
        pop     ax
        pop     bx
        pop     es

; здесь определяем версию системы
mov     ah, 30h
int     21h

push    ax
push    si
lea     si, ModifyNumber
add     si, 16
call    BYTE_TO_DEC
add     si, 3
mov     al, ah
call    BYTE_TO_DEC
pop     si
pop     ax

; здесь определяем серийный номер OEM
mov     al, bh
lea     si, OEM_Code
add     si, 12
call    BYTE_TO_DEC

```

```

; здесь определяем серийный номер пользователя
mov     al, bl
call    BYTE_TO_HEX
lea     di, UserSN
add     di, 20
mov     [di], ax
mov     ax, cx
lea     di, UserSN
add     di, 25
call    WRD_TO_HEX

; выводим все определённые данные
lea     dx, ModifyNumber
call    PRINT_STRING
lea     dx, Oem_Code
call    PRINT_STRING
lea     dx, UserSN
call    PRINT_STRING

; выходим из программы
xor     al, al
mov     ah, 4ch
int     21h

ret
TESTPC  ENDS
END     START

```

ПРИЛОЖЕНИЕ Б

ТЕКСТ ИСХОДНОГО ФАЙЛА .EXE модуля

```
AStack      SEGMENT STACK
AStack      ENDS
```

```
DATA        SEGMENT
```

```
TypePC_Unknown db 'PC Type: Unknown', 0dh, 0ah, '$'
TypePC_PC       db 'PC Type: PC', 0dh, 0ah, '$'
TypePC_PCXT     db 'PC Type: PC/XT', 0dh, 0ah, '$'
TypePC_AT       db 'PC Type: AT', 0dh, 0ah, '$'
TypePC_230PS    db 'PC Type: PC2 model 30', 0dh, 0ah, '$'
TypePC_280PS    db 'PC Type: PS2 model 80', 0dh, 0ah, '$'
TypePC_PCjr     db 'PC Type: PCjr', 0dh, 0ah, '$'
TypePC_PCC      db 'PC Type: PC Convertible', 0dh, 0ah, '$'

ModifyNumber    db 'Modify number:  . ', 0dh, 0ah, '$'
OEM_Code        db 'OEM Code:  ', 0dh, 0ah, '$'
UserSN          db 'User Serial Number:  ', 0dh, 0ah, '$'
```

```
DATA        ENDS
```

```
CODE        SEGMENT
            ASSUME CS:CODE, DS:DATA, SS:AStack
```

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

```
PRINT_STRING 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
```

```

        push    cx
        mov     al, ah
        call    TETR_TO_HEX
        xchg    al, ah
        mov     cl, 4
        shr     al, cl
        call    TETR_TO_HEX
        pop     cx
        ret

BYTE_TO_HEX      ENDP

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

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
        xor     ax, ax
        push    ax
        mov     ax, DATA
        mov     ds, ax

```

; здесь определяем тип ПК

```

        push    es
        push    bx
        push    ax
        mov     bx, 0F000h
        mov     es, bx

```

; предпоследний байт ROM BIOS

```

        mov     ax, es:[0FFFEh]

```

; PC

```

        cmp     al, 0FFh

```

```

        je      MVPC

```

; PC/XT

```

        cmp     al, 0FEh

```

```

        je      MVPCXT

```

```

        cmp     al, 0FBh

```

```

        je      MVPCXT

```

; AT

```

        cmp     al, 0FCh

```

```

        je      MVAT

```

; PS2/30

```

        cmp     al, 0FAh

```

```

        je      MV230PS

```

; коды для AT и PS2/50-60 совпадают, поэтому не обрабатываем
(ну или что делать?)

; PS2/80

```

        cmp     al, 0F8h

```

```

        je      MV280PS
        ; PCjr
        cmp     al, 0FDh
        je      MVPCjr
        ; PC Convertible
        cmp     al, 0F9h
        je      MVPCC
        ; PC Unknown
        lea     dx, TypePC_Unknown
        jmp     MVEND
MVPC:
        lea     dx, TypePC_PC
        jmp     MVEND
MVPCXT:
        lea     dx, TypePC_PCXT
        jmp     MVEND
MVAT:
        lea     dx, TypePC_AT
        jmp     MVEND
MV230PS:
        lea     dx, TypePC_230PS
        jmp     MVEND
MV280PS:
        lea     dx, TypePC_280PS
        jmp     MVEND
MVPCjr:
        lea     dx, TypePC_PCjr
        jmp     MVEND
MVPCC:
        lea     dx, TypePC_PCC
MVEND:
        call    PRINT_STRING
        pop     ax
        pop     bx
        pop     es

        ; здесь определяем версию системы
        mov     ah, 30h
        int     21h

        push    ax
        push    si
        lea     si, ModifyNumber
        add     si, 16
        call    BYTE_TO_DEC

```



```

add     si, 3
mov     al, ah
call    BYTE_TO_DEC
pop     si
pop     ax

; здесь определяем серийный номер OEM
mov     al, bh
lea     si, OEM_Code
add     si, 12
call    BYTE_TO_DEC

; здесь определяем серийный номер пользователя
mov     al, bl
call    BYTE_TO_HEX
lea     di, UserSN
add     di, 20
mov     [di], ax
mov     ax, cx
lea     di, UserSN
add     di, 25
call    WRD_TO_HEX

; выводим все определённые данные
lea     dx, ModifyNumber
call    PRINT_STRING
lea     dx, Oem_Code
call    PRINT_STRING
lea     dx, UserSN
call    PRINT_STRING

; выходим из программы
xor     al, al
mov     ah, 4ch
int     21h

ret

CODE     ENDS
        END     Main

```