

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур заголовочных модулей

Студент гр. 7381

Павлов А.П.

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 загрузки в основную память.

Основные теоретические положения.

Тип IBM PC можно узнать, обратившись к предпоследнему байту ROM BIOS и сопоставив 16-тиричный код и тип в таблице. Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H, входным параметром которой является номер функции в AH.

Выполнение работы.

На основе шаблона, приведенного в методических указаниях, был написан текст исходного .COM модуля, который определял тип PC и версию системы. Был получен “хороший” .COM модуль и “плохой” .EXE модуль. Результаты работы программ представлены на рисунках 1–2.

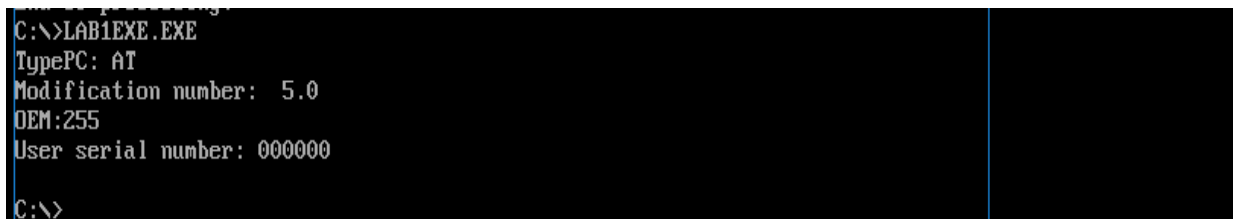
```
Z:\>c:
C:\>LAB1COM.COM
TypePC: AT
Modification number: 5.0
OEM:255
User serial number: 000000
C:\>_
```

Рисунок 1 – результат работы “хорошего” .COM модуля.

```
C:\>LAB1COM.EXE
Modification number: 5.0 255 000000
255 Modification number: 000000
umber: 000000
Modification number:
```

Рисунок 2 – результат работы “плохо” .EXE модуля.

Затем был переписан и отлажен исходный .COM модуль для того, чтобы получить “хороший” .EXE модуль. Результат работы предстален на рисунке 3.



```
C:\>LAB1EXE.EXE
TypePC: AT
Modification number: 5.0
OEM:255
User serial number: 000000
C:\>
```

Рисунок 3 – результат работы “хорошо” .EXE модуля.

Выводы.

Исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки в основную память. Реализована программа на языке ассемблера позволяющая определить тип IBM PC и тип системы.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

Ответ: COM-программа должна содержать один сегмент кода, в котором находятся данные и код.

2. EXE- программа?

Ответ: EXE-программа может содержать несколько программных сегментов, включая сегмент стека, сегмент данных и сегмент кода.

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

Ответ: `assume`-директива, сообщающая транслятору, о том какому сегментному регистру соответствует какой сегмент. Директива `org 100h` сообщает компилятору, что всю адресацию нужно сместить на 256 байт, где будет располагаться PSP.

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

Ответ: нет, так как отсутствует таблица настроек (Relocation table), в которой находится соответствие фактических адресов сегментов и абсолютных ссылок на сегменты, следовательно, нельзя использовать команды, которые используют адрес сегмента и дальнюю адресацию.

Отличия форматов файлов COM и EXE модулей

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

Ответ: COM-файл состоит из команд, процедур и данных. Код начинается с адреса 0h.

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

Ответ: В «плохом» файле EXE данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h идёт таблица настроек.

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

Ответ: 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 код, данные и стек разделены по сегментам.

Загрузка COM модуля в основную память.

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

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

2. Что располагается с адреса 0?

Ответ: префикс программного сегмента PSP.

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

Ответ: Все сегментные регистры указывают на начало PSP.

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

Ответ: Сегмент стека создается в COM-файлах автоматически. Указатель стека устанавливается на конец сегмента и имеет адрес FFFFh.

Загрузка «хорошего» EXE модуля в основную память.

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

Ответ: 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, определяется значение сегментных регистров. DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента кода. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

2. На что указывают регистры DS и ES?

Ответ: Начало PSP.

3. Как определяется стек?

Ответ: 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.

4. Как определяется точка входа?

Ответ: С помощью директивы END. Она указывает метку, в которую переходит программа при запуске.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ТЕКСТ .COM МОДУЛЯ

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

ModifNum	db	'Modification number: . ', 0dh, 0ah, '\$'
----------	----	---

OEM	db	'OEM: ', 0dh, 0ah, '\$'
-----	----	-----------------------------

UserSerialNum	db	'User serial number: ', 0dh, 0ah, '\$'
---------------	----	--

Type_PC_Other	db	2 dup ('?'), 'h\$'
---------------	----	--------------------

Type_PC	db	'TypePC: PC', 0DH, 0AH, '\$'
---------	----	------------------------------

Type_PC_XT	db	'TypePC: PC/XT', 0DH, 0AH, '\$'
------------	----	---------------------------------

Type_AT	db	'TypePC: AT', 0DH, 0AH, '\$'
---------	----	------------------------------

Type_PS2_30	db	'TypePC: PC2 model 30', 0DH, 0AH, '\$'
-------------	----	--

Type_PS2_50	db	'TypePC: PC2 model 50 or 60', 0DH, 0AH, '\$'
-------------	----	--

Type_PS2_80	db	'TypePC: PC2 model 80', 0DH, 0AH, '\$'
-------------	----	--

Type_PCjr	db	'TypePC: PCjr', 0DH, 0AH, '\$'
-----------	----	--------------------------------

Type_PC_Conv	db	'TypePC: PC Convertible', 0DH, 0AH, '\$'
--------------	----	--

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

```

```

PRINT PROC  near
        push ax
        mov  ah,09h
        int     21h
        pop  ax
        ret

```

PRINT ENDP

```

MOD_PC     PROC  near
        push ax
        push si

```

```

        mov     si, offset ModifNum
        add     si, 22
        call    BYTE_TO_DEC
        add     si, 3
        mov     al, ah
        call    BYTE_TO_DEC
        pop     si
        pop     ax
        ret

MOD_PC   ENDP

OEM_PC   PROC near
        push    ax
        push    bx
        push    si
        mov     al,bh
        lea     si, OEM
        add     si, 6
        call    BYTE_TO_DEC
        pop     si
        pop     bx
        pop     ax
        ret

OEM_PC   ENDP

SER_PC   PROC near
        push    ax
        push    bx
        push    cx
        push    si
        mov     al,bl

```

```

        call BYTE_TO_HEX
        lea     di,UserSerialNum
        add     di,20
        mov     [di],ax
        mov     ax,cx
        lea     di,UserSerialNum
        add     di,25
        call WRD_TO_HEX
        pop     si
        pop     cx
        pop     bx
        pop     ax
        ret
SER_PC   ENDP

BEGIN:   mov     bx, 0F000h
        mov     es, bx
        mov     ax, es:[0FFFEh]
        ;PC
        cmp     al, 0FFh
        je      _PC
        ;PC/XT
        cmp     al, 0FEh
        je      _PC_XT
        cmp     al, 0FBh
        je      _PC_XT
        ;AT
        cmp     al, 0FCh
        je      _AT
        ;PS2 model 30
        cmp     al, 0FAh
        je      _PS2_30
        ;PS2 model 80
        cmp     al, 0F8h

```

```

        je          _PS2_80
        ;PCjr
        cmp         al, 0FDh
        je          _PCjr
        ;PC Convertible
        cmp  al, 0F9h
        je          _PC_Conv
        ;unknown type
        call  BYTE_TO_HEX
        lea  di, Type_PC_Other
        mov  [di], ax
        lea  dx, Type_PC_Other
        jmp  _EndPC

_PC: lea  dx, Type_PC
     jmp  _EndPC
_PC_XT: lea  dx, Type_PC_XT
     jmp  _EndPC
_AT: lea  dx, Type_AT
     jmp  _EndPC
_PS2_30:lea  dx, Type_PS2_30
     jmp  _EndPC
_PS2_80:lea  dx, Type_PS2_80
     jmp  _EndPC
_PCjr: lea  dx, Type_PCjr
     jmp  _EndPC
_PC_Conv:lea  dx, Type_PC_Conv

_EndPC: call PRINT

        sub  ax, ax
        mov  ah, 30h
        int  21h
        call MOD_PC

```

```

        call    OEM_PC
        call    SER_PC

        ;print results
        lea     dx, ModifNum
        call    PRINT
        lea     dx, OEM
        call    PRINT
    lea         dx, UserSerialNum
        call    PRINT

        mov ax, 4C00h
        int 21h

TESTPC    ENDS
        END    START

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ ТЕКСТ .EXE МОДУЛЯ

```

_STACK      SEGMENT      STACK
             db      512  dup(0)

_STACK      ENDS


_DATA SEGMENT

             ModifNum      db      'Modification number:  .  ',
0dh, 0ah, '$'

             OEM            db      'OEM:  ',0dh,0ah,'$'

             UserSerialNum db      'User serial number:  ',
0dh, 0ah, '$'

             Type_PC_Other db      2 dup ('?'), 'h$'
             Type_PC       db      'TypePC: PC', 0DH, 0AH, '$'
             Type_PC_XT     db      'TypePC:  PC/XT', 0DH,
0AH, '$'

             Type_AT       db      'TypePC: AT', 0DH, 0AH, '$'
             Type_PS2_30    db      'TypePC: PC2 model 30', 0DH,
0AH, '$'

             Type_PS2_50    db      'TypePC: PC2 model 50 or 60',
0DH, 0AH, '$'

             Type_PS2_80    db      'TypePC: PC2 model 80', 0DH,
0AH, '$'

             Type_PCjr      db      'TypePC: PCjr', 0DH, 0AH, '$'
             Type_PC_Conv   db      'TypePC: PC Convertible', 0DH,
0AH, '$'

_DATA      ENDS

_CODE SEGMENT

             ASSUME CS:_CODE, DS:_DATA, ES:NOTHING, SS:_STACK


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

```

```

PRINT PROC near

```

```

        push    ax
        mov     ah,09h
        int     21h

```



```

        pop    ax
        ret

PRINT ENDP

MOD_PC    PROC near
        push  ax
        push  si
        mov   si, offset ModifNum
        add   si, 22
        call  BYTE_TO_DEC
        add   si, 3
        mov   al, ah
        call  BYTE_TO_DEC
        pop   si
        pop   ax
        ret

MOD_PC    ENDP

OEM_PC    PROC near
        push  ax
        push  bx
        push  si
        mov   al, bh
        lea   si, OEM
        add   si, 6
        call  BYTE_TO_DEC
        pop   si
        pop   bx
        pop   ax
        ret

OEM_PC    ENDP

```

```

SER_PC    PROC  near
           push  ax
           push  bx
           push  cx
           push  si
           mov   al,b1
           call  BYTE_TO_HEX
           lea   di,UserSerialNum
           add   di,20
           mov   [di],ax
           mov   ax,cx
           lea   di,UserSerialNum
           add   di,25
           call  WRD_TO_HEX
           pop   si
           pop   cx
           pop   bx
           pop   ax
           ret
SER_PC    ENDP

```

```

MAIN  PROC  near
       mov  ax, _DATA
       mov  ds, ax
       sub  ax, ax
       mov  bx, 0F000h
       mov  es, bx
       mov  ax, es:[0FFFEh]
       ;PC
       cmp  al, 0FFh
       je   _PC
       ;PC/XT
       cmp  al, 0FEh

```

```

        je          _PC_XT
        cmp     al, 0FBh
        je          _PC_XT
        ;AT
        cmp     al, 0FCh
        je          _AT
        ;PS2 model 30
        cmp     al, 0FAh
        je          _PS2_30
        ;PS2 model 80
        cmp     al, 0F8h
        je          _PS2_80
        ;PCjr
        cmp     al, 0FDh
        je          _PCjr
        ;PC Convertible
        cmp     al, 0F9h
        je          _PC_Conv
        ;unknown type
        call    BYTE_TO_HEX
        lea     di, Type_PC_Other
        mov     [di], ax
        lea     dx, Type_PC_Other
        jmp     _EndPC

_PC: lea     dx, Type_PC
        jmp     _EndPC
_PC_XT: lea     dx, Type_PC_XT
        jmp     _EndPC
_AT: lea     dx, Type_AT
        jmp     _EndPC
_PS2_30:lea     dx, Type_PS2_30
        jmp     _EndPC
_PS2_80:lea     dx, Type_PS2_80

```

```

        jmp      _EndPC
_PCjr:   lea     dx, Type_PCjr
        jmp      _EndPC
_PC_Conv:lea     dx, Type_PC_Conv

_EndPC: call PRINT

        sub     ax, ax
        mov     ah, 30h
        int     21h
        call    MOD_PC
        call     OEM_PC
        call    SER_PC

        ;print results
        lea     dx, ModifNum
        call    PRINT
        lea     dx, OEM
        call    PRINT
        lea     dx, UserSerialNum
        call    PRINT

        mov     ax, 4C00h
        int     21h
        ret

MAIN ENDP
_CODE     ENDS
        END     MAIN

```

