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 _____ Дорох С.В.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург
2018

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.

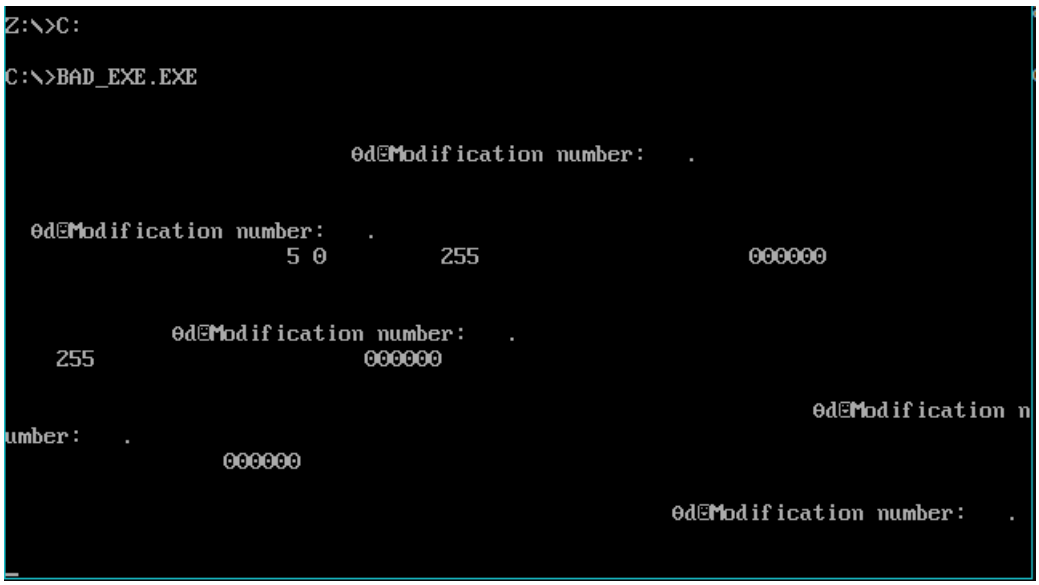
Тип IBM PC можно узнать обратившись к предпоследнему байту ROM BIOS и сопоставив 16-тиричный код и тип в таблице. Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H, входным параметром является номер функции в AH.

Выполнение работы.

Написан текст исходного .COM модуля, который определяет тип PC и версию системы. Для решения поставленной задачи был использован шаблон ассемблерного текста с функциями управляющей программы и процедурами перевода двоичных кодов в символы шестадцатеричных чисел и десятичное число из раздела “общие сведения” методический указаний. Для того, чтобы узнать тип IBM PC программа обращается к предпоследнему байту ROM BIOS. Далее полученное значение сравнивается с таблицей. Для определения версии MS DOS используется функция 30h 21h-го прерывания. И в соответствие с полученными данными в регистрах.

Написан текст исходного .EXE модуля с тем же функционалом.

1. Результат работы «плохого» .EXE модуля (BAD_EXE.exe):



2. Результат работы «хорошего» .COM модуля (LAB_COM.com):

3. Результат работы «хорошего» .EXE модуля (GOOD_EXE.exe):

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: Один сегмент – сегмент кода

2. EXE-программа?

Ответ: 3 сегмента (сегмент стека, сегмент кода и сегмент данных).

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

Ответ: В случае комментирования ORG 100h линковщик жалуется на ошибку с текстом «Connot generate COM file: invalid initial entry point address», это связано с тем, что когда загрузочный модуль попадает в оперативную память, то в начале программы резервируется 100h байт под PSP. В случае комментирования ASSUME, компилятор выводит ошибку «Near jump or call to different CS», это связано с тем, что необходимо привязать сегментный регистр CS к моему сегменту(TESTPC).

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

Ответ: Нельзя использовать команды, которые используют адреса сегментов.

Отличия форматов файлов COM и EXE модулей.

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

Ответ: COM-файл содержит данные и машинные команды. 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 «плохом» EXE всего один сегмент и для данных и для кода. В «хорошем» есть разбиение на сегменты, также присутствует стэк.

Также, в «хорошем» EXE файле код располагается с адреса 200h, а в «плохом» - с адреса 300h.

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: 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 генерируется 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: 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 сегмента PSP.

3. Как определяется стек?

Ответ: Регистрам SS и SP присваиваются значения, указанные в заголовке, а затем 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,'$'
OEM        db    'OEM:     ',0dh,0ah,'$'
SerialNum  db    'Version number:        ',0dh,0ah,'$'

str_type   db 'PC type is ','$'
str_PC           db 'PC',0DH,0AH,'$'
str_PC_XT  db 'PC/XT',0DH,0AH,'$'
str_AT           db 'AT',0DH,0AH,'$'
str_PC2_30       db 'PC2 model 30',0DH,0AH,'$'
str_PC2_50       db 'PC2 model 50 or 60',0DH,0AH,'$'
str_PC2_80       db 'PC2 model 80',0DH,0AH,'$'
str_PCjr   db 'PCjr',0DH,0AH,'$'
str_PC_Conv db '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 ; в AL старшая цифра
            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



TYPE_PC PROC NEAR

            mov ax, 0F000h
            mov es, ax
            sub bx, bx
            mov bh, es:[0FFFEh]
            ret

TYPE_PC 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,SerialNum
            add         di,17
            mov   [di],ax
            mov   ax,cx
            lea         di,SerialNum
            add         di,22
            call  WRD_TO_HEX
            pop         si
            pop         cx
            pop         bx
            pop   ax
            ret
SER_PC      ENDP

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

```
            pop ax
            ret
PRINT ENDP

BEGIN:
            call  TYPE_PC


            mov dx, offset str_type
            call PRINT


            mov dx, offset str_PC_XT
            cmp bh, 0FEh
            je   to_print

            mov dx, offset str_AT
            cmp bh, 0FCh
            je   to_print

            mov dx, offset str_PC2_30
            cmp bh, 0FAh
            je   to_print

            mov dx, offset str_PC2_80
            cmp bh, 0F8h
            je   to_print

            mov dx, offset str_PCjr
            cmp bh, 0FDh
            je   to_print

            mov dx, offset str_PC_Conv
            cmp bh, 0F9h
            je   to_print

            mov dx, offset str_PC2_50
            cmp bh, 0FCh
            je   to_print
```

```
            mov dx, offset str_PC
            cmp bh, 0FFh
            je   to_print

to_print:
            call  PRINT


            mov ah, 30h
            int 21h

            call  MOD_PC
            call    OEM_PC
            call  SER_PC


            lea dx, ModifNum
            call PRINT
            lea dx, OEM
            call PRINT
          lea dx, SerialNum
            call PRINT

            xor        al,al
            mov   ah,3ch
            int        21h
            ret
TESTPC      ENDS
            END   START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ ТЕКСТ .EXE МОДУЛЯ

```
EOL EQU '$'
ASTACK SEGMENT STACK
          DW 512 DUP(?)
ASTACK ENDS


DATA SEGMENT

     STR_TYPE   DB 'PC TYPE IS ','$'
     MODIFNUM   DB   'MODIFICATION NUMBER:   .   ',0DH,0AH,'$'
     OEM        DB   'OEM:    ',0DH,0AH,'$'
     SERIALNUM  DB   'VERSION NUMBER:       ',0DH,0AH,'$'


     STR_PC          DB 'PC',0DH,0AH,'$'
     STR_PC_XT  DB 'PC/XT',0DH,0AH,'$'
     STR_AT          DB 'AT',0DH,0AH,'$'
     STR_PC2_30      DB 'PC2 MODEL 30',0DH,0AH,'$'
     STR_PC2_50      DB 'PC2 MODEL 50 OR 60',0DH,0AH,'$'
     STR_PC2_80      DB 'PC2 MODEL 80',0DH,0AH,'$'
     STR_PCJR   DB 'PCJR',0DH,0AH,'$'
     STR_PC_CONV DB 'PC CONVERTIBLE',0DH,0AH,'$'

DATA ENDS


CODE SEGMENT
          ASSUME CS:CODE, DS:DATA, SS:ASTACK


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

TYPE_PC PROC NEAR

          MOV AX, 0F000H
          MOV ES, AX
```

```
            SUB BX, BX
            MOV BH, ES:[0FFFEH]
            RET

TYPE_PC 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,SERIALNUM
          ADD       DI,17
          MOV  [DI],AX
          MOV  AX,CX
          LEA       DI,SERIALNUM
          ADD       DI,22
          CALL WRD_TO_HEX
          POP       SI
          POP       CX
          POP       BX
          POP  AX
          RET
SER_PC    ENDP


PRINTPROC NEAR
          MOV  AH,09H
          INT       21H
          RET
PRINT     ENDP


MAIN PROC NEAR
```

```
PUSH DS
SUB        AX,AX
PUSH AX
MOV  AX,DATA
MOV  DS,AX
SUB        AX,AX

CALL TYPE_PC

MOV DX, OFFSET STR_TYPE
CALL PRINT

MOV DX, OFFSET STR_PC_XT
CMP BH, 0FEH
JE   TO_PRINT

MOV DX, OFFSET STR_AT
CMP BH, 0FCH
JE   TO_PRINT

MOV DX, OFFSET STR_PC2_30
CMP BH, 0FAH
JE   TO_PRINT


MOV DX, OFFSET STR_PC2_80
CMP BH, 0F8H
JE   TO_PRINT

MOV DX, OFFSET STR_PCJR
CMP BH, 0FDH
JE   TO_PRINT

MOV DX, OFFSET STR_PC_CONV
```

```
            CMP BH, 0F9H
            JE   TO_PRINT

            MOV DX, OFFSET STR_PC2_50
            CMP BH, 0FCH
            JE   TO_PRINT

            MOV DX, OFFSET STR_PC
            CMP BH, 0FFH
            JE   TO_PRINT

TO_PRINT:
     CALL PRINT

            MOV AH, 30H
            INT 21H
            CALL MOD_PC
            CALL    OEM_PC
            CALL SER_PC



            LEA DX, MODIFNUM
            CALL PRINT
            LEA DX, OEM
            CALL PRINT
          LEA DX, SERIALNUM
            CALL PRINT

            XOR       AL,AL
            MOV  AH,4CH
            INT       21H
            RET
MAIN ENDP
```

```
CODE ENDS
            END MAIN
```