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		Габов Е. С.
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 21h-го прерывания. Входной параметр:

mov ah, 30h

int 21h

Выходные параметры:

AL – номер основной версии

AH – номер модификации

BH – серийный номер OEM

BL:CX – 24-битовый серийный номер пользователя

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 (Good_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: обязательно один сегмент, но можно больше. В частности предусматриваются отдельные сегменты для кода, данных и стека.

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

Ответ: Директива ORG 100h, которая задает смещение для всех адресов программы на 256 байт для префикса программного сегмента (PSP). Также необходима директива ASSUME, с помощью директивы ASSUME ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами. 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).

Все ли форматы команд можно использовать в COM-программе?

Ответ: Нет, не все. COM-программа подразумевает наличие только одного сегмента, а значит, можно использовать только near-переходы, так как в far-переходах подразумевается использование нескольких сегментов. Так же нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен, т.к. в COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, потому что подобные адреса в нём запрещены.

Отличия форматов файлов COM и EXE модулей.

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

Ответ: Файл COM состоит из команд, процедур и данных, используемых в программе. Код начинается с нулевого 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 0 адреса располагается управляющая информация для загрузчика.

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. 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 данные, стек и код 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. Автоматически определяется стэк.

2. Что располагается с адреса 0?

Ответ: PSP.

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

Ответ: Все сегментные указывают на 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 (стек размещается в конце 64 Кб блока). Следовательно, он занимает оставшуюся память и растет «навстречу» программе.

Загрузка «хорошего» EXE модуля в основную память.

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

Ответ: Сначала генерируется PSP, определяется длина тела загрузочного модуля, определяется начальный сегмент. Затем загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, к полю каждого сегмента прибавляется сегментный адрес начального сегмента и определяются значения сегментных регистров. DS и ES указывают на начало сегмента PSP (119C), CS указывает на начало сегмента кода (11AC), а SS – на начало сегмента стека (11CD).

2. На что указывают регистры DS и ES?

Ответ: Начало сегмента 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.

Как определяется точка входа?

Ответ: Смещение точки входа в программу загружается в указатель команд IP. Адрес, с которого начинается выполнение программы, определяется операндом директивы END, который называется точкой входа.
END <процедура> - процедура, с которой следует начинать программу.

ПРИЛОжение а

исходный текст .Com модуля

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

PC_TYPE db ' type is:  ', '$'
PC db 'PC', 0dh, 0ah, '$'
PCXT db 'PC/XT', 0dh, 0ah, '$'
AT db 'AT', 0dh, 0ah, '$'
Ps2_30 db 'Ps2_30', 0dh, 0ah, '$'
Ps2_50_60 db 'Ps2_50_60', 0dh, 0ah, '$'
Ps2_80 db 'ps2_80', 0dh, 0ah, '$'
PCjr db 'PCjr', 0dh, 0ah, '$'
PC_Convertible db 'PC Conventible', 0dh, 0ah, '$'
V_NUMBER db 'VERSION NUMBER:              ', 0dh, 0ah, '$'
M_NUMBER db 'MODIFICATION NUMBER:         ', 0dh, 0ah, '$'
S_NUMBER db 'SERIES NUMBER:               ', 0dh, 0ah, '$'
USER_NUMBER db 'USER NUMBER:              ', 0dh, 0ah, '$'
DEFAULT db 'DONT EQUAL ANYONE             ', 0dh, 0ah, '$'


PRINT PROC NEAR ; print by offset which contain in dx
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT 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
        call BYTE_TO_HEX
        mov [DI],AH
```

```
        dec DI
        mov [DI],AL
        pop BX
        ret
WRD_TO_HEX ENDP

TETR_TO_HEX     PROC  near ; only half part of al convert to ASII
        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 ; al convert to ASII
        push    CX
        mov     AH,AL
        call    TETR_TO_HEX
        xchg    AL,AH
        mov     CL,4
        shr     AL,CL
        call    TETR_TO_HEX
        pop     CX
        ret
BYTE_TO_HEX     ENDP

BYTE_TO_DEC     PROC  near
        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


DEFINE_PCTYPE PROC NEAR
        push es
        mov bx, 0F000h
        mov es, bx
        sub bx, bx
        mov bh, es:[0FFFEh] ; bh contain type of PC
        pop es
        ret
DEFINE_PCTYPE ENDP


DefineVersion PROC NEAR ;AL
        push ax
        push si
        mov si, offset V_NUMBER
        add si, 10h
        call BYTE_TO_DEC
        pop si
        pop ax
        ret
DefineVersion ENDP

DefineModification PROC NEAR ;AH
        push ax
        push si
        mov si, offset M_NUMBER
        add si, 17h
        mov al, ah
        call BYTE_TO_DEC
        pop si
        pop ax
        ret
```

```
DefineModification ENDP

DefineOEM PROC NEAR ;BH
        push ax
        push bx
        push si
        mov si, offset S_NUMBER
        add si, 11h
        mov al, bh
        call BYTE_TO_DEC
        pop si
        pop bx
        pop ax
        ret
DefineOEM ENDP

DefineUNumber PROC NEAR ;BL:CX
        push bx
        push cx
        push di
        push ax
        mov si, offset USER_NUMBER
        add si, 13
        mov ax, cx
        call WRD_TO_HEX
        mov al, bl
        call BYTE_TO_HEX
        mov si, offset USER_NUMBER
        add si, 14
        mov [si], ax
        pop ax
        pop di
        pop cx
        pop bx
        ret
DefineUNumber ENDP

BEGIN:
        call DEFINE_PCTYPE
        push dx
        mov dx, offset PC_TYPE
        call PRINT
        pop dx
```

```
        mov dx, offset PC
        cmp bh, 0FFh
        je      PrintType

        mov dx, offset PCXT
        cmp bh, 0FEh
        je      PrintType

        mov dx, offset AT
        cmp bh, 0FCh
        je      PrintType

        mov dx, offset Ps2_30
        cmp bh, 0FAh
        je      PrintType

        mov dx, offset Ps2_50_60
        cmp bh, 0FCh
        je      PrintType

        mov dx, offset Ps2_80
        cmp bh, 0F8h
        je      PrintType

        mov dx, offset PCjr
        cmp bh, 0FDh
        je      PrintType

        mov dx, offset PC_Convertible
        cmp bh, 0F9h
        je      PrintType

        mov al, bh
        call BYTE_TO_HEX
        mov dx, offset DEFAULT

PrintType:
        call PRINT
        call DefineVersion
        call DefineModification
        call DefineOEM
        call DefineUNumber
```

```
        mov dx, offset V_NUMBER
        call PRINT
        mov dx, offset M_NUMBER
        call PRINT
        mov dx, offset S_NUMBER
        call PRINT
        mov dx, offset USER_NUMBER
        call PRINT

        xor     al, al
        mov     ah, 4Ch
        int     21h
TESTPC  ENDS

END   START
```