МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе № 1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студент гр. _____ Аверина О.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Постановка задачи.

1. Требуется написать текст исходного .COM модуля, который определяет тип PC и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели.
2. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx – номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.
3. Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.
4. Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить, сравнить исходные тексты для .COM и .EXE модулей и ответить на вопросы.
5. Запустить FAR и открыть файл загрузочного модуля .COM и файл плохого .EXE в шестнадцатеричном виде, сравнить с хорошим файлом. Ответить на вопросы.
6. Запустить в отладчике TD.EXE .COM. Ответить на контрольные вопросы. Представить план загрузки модуля .COM в основную память.

7. Открыть в отладчике TD.EXE «хороший» .EXE. Ответить на вопросы.

Выполнение работы.

В качестве основы программы был взят шаблон из методического пособия, в котором содержатся следующие процедуры:

- 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 10-ю с/с
- WRITE_PROC – Печать символов

Были объявлены строки для вывода информации:

- TYPE_PC db 'Type: PC',0DH,0AH,'$'
- TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
- TYPE_AT db 'Type: AT',0DH,0AH,'$'
- TYPE_PS2_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
- TYPE_PS2_50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
- TYPE_PS2_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
- TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
- TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'
- VERSION_OS db 'Version DOS: . ',0DH,0AH,'$'
- OEM db 'Serial number OEM: ',0DH,0AH,'$'
- USER_NUMBER db 'User serial number: H $'

Были создана процедура для определения типа ПК PC_WRITE в соответствии с таблицей:

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

А также функция для определения характеристик ОС VERSION_OS_proc:

- номер основной версии системы
- номер модификации;
- серийный номер OEM;
- серийный номер пользователя.

В результате выполнения программы были получены следующие значения:

Рисунок 1 – «хороший» .COM модуль

Рисунок 2 – «плохой» .EXE модуль

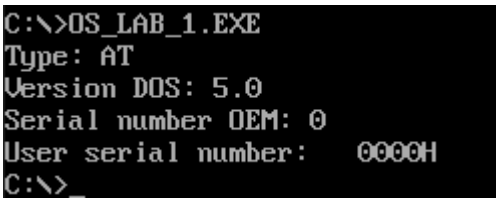

Рисунок 3 – «хороший» .EXE модуль

Ответы на вопросы.

Отличия исходный текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

В .COM файле все данные, код и стек располагаются в одном сегменте.

2) EXE – программа?

EXE-программа должна содержать не менее одного сегмента. Каждый сегмент определяется отдельно.

3) Какие директивы обязательно должны быть в тексте COM-программе?

В COM-файле обязательно должна быть директива ORG 100h, чтобы зарезервировать память для PSP. Также требуется директива ASSUME, где прописывается, что сегменты кода и данных указывают на один сегмент программы.

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

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>, так как в .COM-программе отсутствует таблица настроек (содержит описание адресов, которые зависят от размещения загрузочного модуля в ОП).

Отличия форматов файлов .COM и.EXE программ

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

В .COM файле все данные, код и стек располагаются в одном сегменте и не могут превышать 64 килобайта. При загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы после нее. При вызове COM-файла в стек помещаются сегментный адрес и ноль. Вид COM – файла представлен на рисунке 4.

Рис. 4 – «хороший» .COM – файл

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

В «плохом» EXE - файле данные и код располагаются в одном сегменте, что для EXE - файла некорректно, так как код и данные должны быть разделены на отдельные сегменты. Код располагается с адреса 300h, а с адреса 0h идёт таблица настроек. Вид «плохого» EXE – файла представлен на рисунке 5.

Рис. 5 – «плохой» .EXE – файл

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

В EXE-программе код, данные и стек поделены на сегменты. Заголовок состоит из форматированной части, содержащей сигнатуру и данные, необходимые для загрузки EXE-файла, и таблицы для настройки адресов. В отличии от «плохого» EXE в «хорошем» EXE присутствуют три сегмента: сегмент кода, сегмент данных и сегмент стека, а «плохой» EXE содержит один сегмент, совмещающий код и данные. Вид «хорошего» EXE – файла представлен на рисунке 6.

Рис. 6 – «хороший» EXE – файл

Загрузка COM – модуля в память

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

1. Система выделяет свободный сегмент памяти и заносит его адрес во все сегментные регистры (CS, DS, ES и SS).
2. В первые 256 байт этого сегмента записывается PSP.
3. Непосредственно за ним загружается содержимое COM-файла без изменений.
4. Указатель стека (регистр SP) устанавливается на конец сегмента.
5. В стек записывается 0000h (адрес возврата для команды ret).
6. Управление передаётся по адресу CS:0100h, где находится первый байт исполняемого файла.

Код располагается сразу после PSP по адресу 100h.

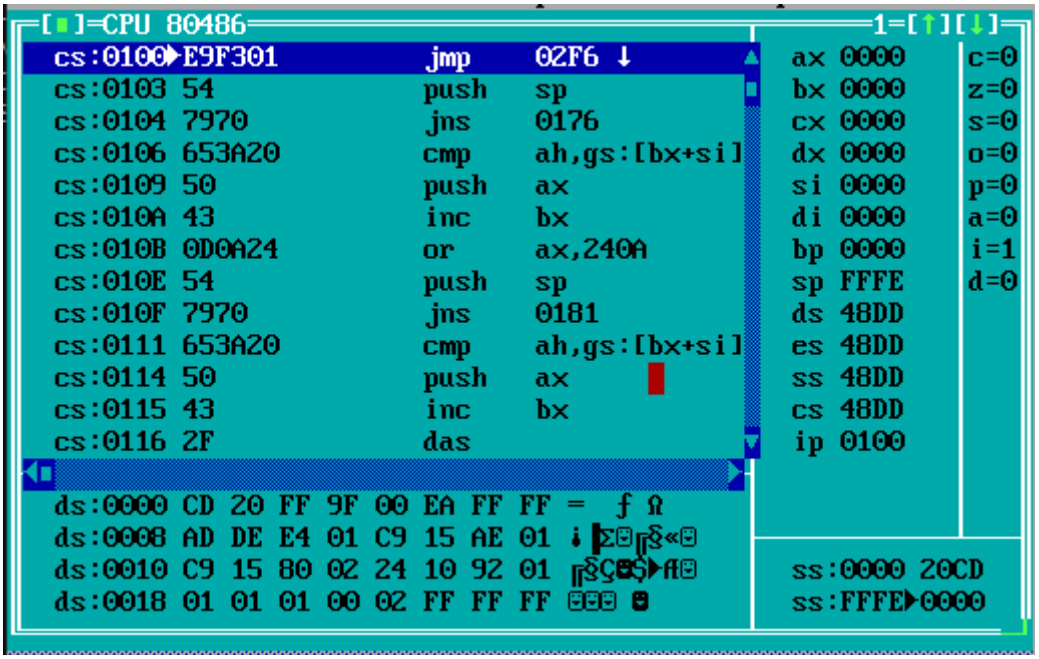

Рис. 7 – COM-файл в отладчике TD

2) Что располагается с адреса 0?

Сегмент PSP.

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

Все сегменты равны нулю. Все он указывают на PSP.

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

В SP содержится адрес FFFEh, т.е. на начало стека в момент загрузки модуля. В com-модулях стек растёт от старших адресов к младшим. SS – на начало.

Загрузка «хорошего» EXE модуля в основную память

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

DS и ES указывают на начало префикса программного сегмента, SS– на начало сегмента стека, CS – на начало сегмента команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END. Вид «хорошего» EXE – файла в отладчике TD представлен на рисунке 8.

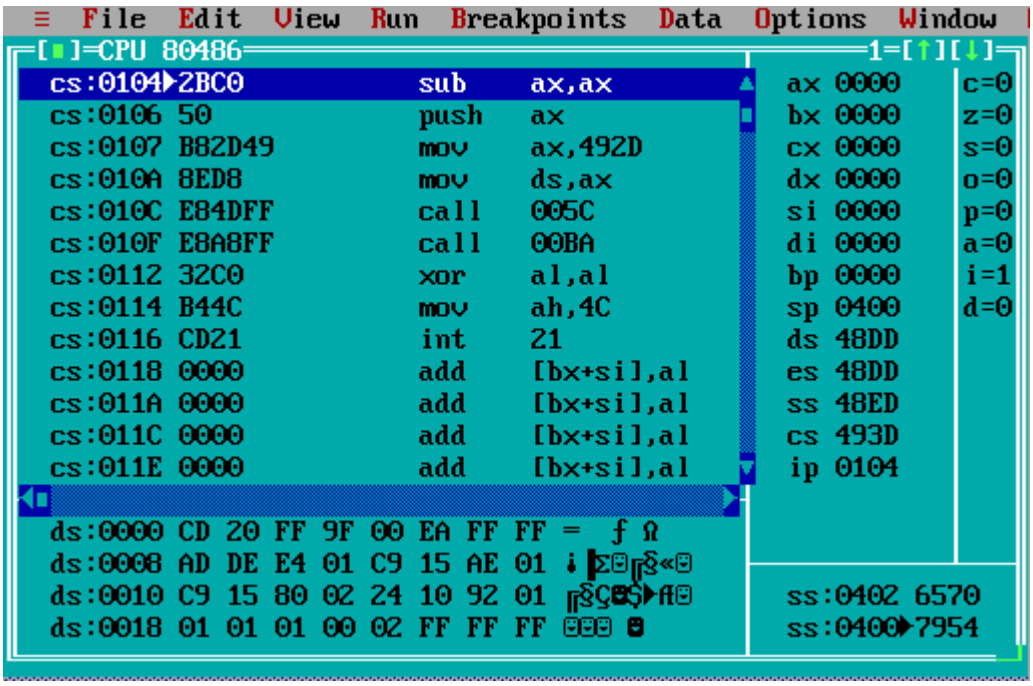

Рис 8 - COM-файл в отладчике TD

2) На что указывают регистры DS и ES?

DS и ES указывают на начало префикса программного сегмента.

3) Как определяется стек?

Стек определяется с помощью директивы .stack, после которой задаётся размер стека. При исполнении регистр SS указывает на начало сегмента стека, а SP на конца стека(его смещение).

4) Как определяется точка входа?

Точка входа указывается после директивы END, по которой программа переходит при запуске.

Выводы.

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А (OS_1_COM.asm)

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
;Данные
TYPE_PC db  'Type: PC',0DH,0AH,'$'
TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
TYPE_AT db  'Type: AT',0DH,0AH,'$'
TYPE_PS2_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
TYPE_PS2_50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
TYPE_PS2_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

VERSION_OS db 'Version DOS:  .  ',0DH,0AH,'$'
OEM db  'Serial number OEM:  ',0DH,0AH,'$'
USER_NUMBER db  'User serial number:       H $'
;Процедуры
;-----------------------------------------------------
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
;байт в AL переводится в два символа шест. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;в AL старшая цифра
pop CX ;в AH младшая
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
call BYTE_TO_HEX
mov [DI],AH
dec DI
mov [DI],AL
pop BX
ret
WRD_TO_HEX ENDP
;--------------------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля младшей цифры
     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
;------------------------------
WRITE_PROC PROC near
     mov AH,09h
     int 21h
     ret
WRITE_PROC ENDP

PC_WRITE PROC near

     mov ax, 0f000h ; получаем номер модели
```

```
     mov es, ax
     mov al, es:[0fffeh]

     cmp al, 0ffh ; начинаем стравнивать
     je pc
     cmp al, 0feh
     je pc_xt
     cmp al, 0fbh
     je pc_xt
     cmp al, 0fch
     je pc_at
     cmp al, 0fah
     je pc_ps2_m30
     cmp al, 0f8h
     je pc_ps2_m80
     cmp al, 0fdh
     je pc_jr
     cmp al, 0f9h
     je pc_conv
     xor dh, dh
     mov dl, al
     jmp write

pc:
          mov dx, offset TYPE_PC
          jmp write
pc_xt:
          mov dx, offset TYPE_PC_XT
          jmp write
pc_at:
          mov dx, offset TYPE_AT
          jmp write
pc_ps2_m30:
          mov dx, offset TYPE_PS2_30
          jmp write
pc_ps2_m80:
          mov dx, offset TYPE_PS2_80
          jmp write
pc_jr:
          mov dx, offset TYPE_PC_JR
          jmp write
pc_conv:
          mov dx, offset TYPE_PC_CONV
          jmp write
write:
          call WRITE_PROC
     ret
```

```
PC_WRITE ENDP

VERSION_OS_proc PROC near
     MOV AH,30h
     INT 21h

     push ax

     mov si, offset VERSION_OS
     add si, 13   ; сдвиг в строке
     call BYTE_TO_DEC
     pop ax
     mov al, ah
     add si, 3
     call BYTE_TO_DEC
     mov dx, offset VERSION_OS ; сдвиг для печати
     call WRITE_PROC

     mov si, offset OEM
     add si, 19
     mov al, bl
     call BYTE_TO_DEC
     mov dx, offset OEM
     call WRITE_PROC

     mov di, offset USER_NUMBER
     add di, 25
     mov ax, cx
     call WRD_TO_HEX
     mov al, bl
     call BYTE_TO_DEC
     sub di, 2 ; what is it
     mov [di], ax
     mov dx, offset USER_NUMBER
     call WRITE_PROC

     ret

VERSION_OS_proc ENDP

BEGIN:

call PC_WRITE
call VERSION_OS_proc
xor AL,AL
mov AH,4Ch
```

```
     int 21H
     TESTPC ENDS
END START ;
```

ПРИЛОЖЕНИЕ Б (OS_1_exe.asm)

```
     STACK SEGMENT STACK
          DW 200H DUP(?) ; заполнен мусором
     STACK ENDS

     ;Данные
     DATA SEGMENT
     TYPE_PC db  'Type: PC',0DH,0AH,'$'
     TYPE_PC_XT db 'Type: PC/XT',0DH,0AH,'$'
     TYPE_AT db  'Type: AT',0DH,0AH,'$'
     TYPE_PS2_30 db 'Type: PS2 модель 30',0DH,0AH,'$'
     TYPE_PS2_50_60 db 'Type: PS2 модель 50 или 60',0DH,0AH,'$'
     TYPE_PS2_80 db 'Type: PS2 модель 80',0DH,0AH,'$'
     TYPE_PC_JR db 'Type: PCjr',0DH,0AH,'$'
     TYPE_PC_CONV db 'Type: PC Convertible',0DH,0AH,'$'

     VERSION_OS db 'Version DOS:  .  ',0DH,0AH,'$'
     OEM db  'Serial number OEM:    ',0DH,0AH,'$'
     USER_NUMBER db  'User serial number:       H $'
     DATA ENDS
     ;Процедуры
     ;-------------------------------------------------
     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
     ;байт в AL переводится в два символа шест. числа в AX
     push CX
     mov AH,AL
     call TETR_TO_HEX
     xchg AL,AH
     mov CL,4
     shr AL,CL
     call TETR_TO_HEX ;в AL старшая цифра
     pop CX ;в AH младшая
     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
     call BYTE_TO_HEX
     mov [DI],AH
     dec DI
     mov [DI],AL
     pop BX
     ret
WRD_TO_HEX ENDP
;--------------------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля младшей цифры
     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
;--------------------------------
WRITE_PROC PROC near
     mov AH,09h
     int 21h
     ret
WRITE_PROC ENDP

PC_WRITE PROC near

     mov ax, 0f000h ; получаем номер модели
     mov es, ax
     mov al, es:[0fffeh]

     cmp al, 0ffh ; начинаем стравнивать
     je pc
     cmp al, 0feh
     je pc_xt
     cmp al, 0fbh
     je pc_xt
     cmp al, 0fch
     je pc_at
     cmp al, 0fah
     je pc_ps2_m30
     cmp al, 0f8h
     je pc_ps2_m80
     cmp al, 0fdh
     je pc_jr
     cmp al, 0f9h
     je pc_conv
     xor dh, dh
     mov dl, al
     jmp write

pc:
          mov dx, offset TYPE_PC
          jmp write
pc_xt:
          mov dx, offset TYPE_PC_XT
          jmp write
pc_at:
          mov dx, offset TYPE_AT
```

```
          jmp write
pc_ps2_m30:
          mov dx, offset TYPE_PS2_30
          jmp write
pc_ps2_m80:
          mov dx, offset TYPE_PS2_80
          jmp write
pc_jr:
          mov dx, offset TYPE_PC_JR
          jmp write
pc_conv:
          mov dx, offset TYPE_PC_CONV
          jmp write
write:
          call WRITE_PROC
     ret
PC_WRITE ENDP



VERSION_OS_proc PROC near
     MOV AH,30h
     INT 21h

     push ax

     mov si, offset VERSION_OS
     add si, 13   ; сдвиг в строке
     call BYTE_TO_DEC
     pop ax
     mov al, ah
     add si, 3
     call BYTE_TO_DEC
     mov dx, offset VERSION_OS ; сдвиг для печати
     call WRITE_PROC

     mov si, offset OEM
     add si, 19
     mov al, bl
     call BYTE_TO_DEC
     mov dx, offset OEM
     call WRITE_PROC

     mov di, offset USER_NUMBER
     add di, 25
     mov ax, cx
```

```
          call WRD_TO_HEX
          mov al, bl
          call BYTE_TO_DEC
          sub di, 2
          mov [di], ax
          mov dx, offset USER_NUMBER
          call WRITE_PROC

          ret

     VERSION_OS_proc ENDP

     TYPE_OS PROC near
          sub ax,ax;
          push  AX
          mov ax, DATA
          mov ds, ax

          call PC_WRITE
          call VERSION_OS_proc

          xor AL,AL
          mov AH,4Ch
          int 21H
     TYPE_OS ENDP
     CODE ENDS
END  TYPE_OS
```