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.

Студентка гр. 9383 _____ Лысова А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

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

Задание.

Написать тексты исходных .COM и .EXE модулей, которые определяют 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 название модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа, и выводится на экран соответствующего сообщения.

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AH и AL формировать тестовую строку в формате xx.yy, где xx — номер основной версии, а yy — номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран.

Основные теоретические положения.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа:

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**

Ассемблерная программа должна читать содержимое этого байта, переводить двоичный код в символьную строку, содержащую запись шестнадцатиричного числа и выводить на экран эту строку.

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

MOV AH,30h

INT 21h

Выходными параметрами являются:

AL - номер основной версии. Если 0, то < 2.0

AH - номер модификации

BH - серийный номер OEM (Original Equipment Manufacturer)

BL:CX - 24-битовый серийный номер пользователя.

Выполнение работы.

1. Был написан текст исходного загрузочного .COM модуля, выполняющего поставленную задачу. После чего были получены «плохой» .EXE файл и «хороший» .COM файл, благодаря компиляции и использованию exe2bin.exe.

2. Был переписан текст исходного модуля так, чтобы получить «хороший» загрузочный .EXE модуль.

3. Модули были загружены в отладчик TD.EXE и проанализированы в сравнении друг с другом, после чего были сделаны выводы и приведены ответы на контрольные вопросы.

Используемые функции.

- TETR_TO_HEX — шаблонная функция, которая переводит десятичное число в код символа.
- BYTE_TO_HEX — шаблонная функция, которая переводит байт в шестнадцатеричной СС в код символа.
- WRD_TO_HEX — шаблонная функция, которая переводит шестнадцатеричное число в символьный код.
- BYTE_TO_DEC — шаблонная функция, которая переводит байт в шестнадцатеричной СС в символьный код десятичной СС.
- PRINT_STRING — функция, которая печатает строку на экран.
- PRINT_TYPE — функция, которая печатает тип PC.
- VERSION_DOS — функция, которая печатает версию MS DOS на экран.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Один сегмент, в котором хранятся и данные и сам код.

2) EXE-программа?

По-разному. Но в такой программе сегменты стека, данных и кода различны.

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

В тексте COM-программы обязательно должна быть директива **ORG 100h**, которая устанавливает значение программного счетчика в 100h, так как при загрузке COM-файла в память DOS занимает первые 256 байт, что в шестнадцатеричной СС пишется как 100h, под разного рода управляющие структуры. Поэтому вся адресация кода дожна начинаться со смещением на 256 байт. И директива **ASSUME,** которая сопоставляет сегментные регистры и программные сегменты. Это позволяет компилятору корректно связывать имена, определенные в сегментах, а так же следить за их смещением, вернее относительно чего их смещение происходит. А так же директива **END,** которая позволяет завершить программу.

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

В COM-программе нельзя использовать команды типа seg NAME, где NAME это название сегмента, потому что в COM-программах нет таблицы настройки. А так же нельзя использовать процедуры с переходом far, так как сегмент в такой программе всего лишь один.

Отличия форматов файлов .COM и .EXE модулей.

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

Структура COM файла состоит из команд и процедур, а также непосредственно данных. Получается, что стек, данные и код находятся в пределах одного сегмента. Код располагается с нулевого адреса.

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

Так как «плохой» EXE файл списан с COM файла, то здесь тоже все предполагаемые сегменты находятся внутри одного общего. Но код и данные будут начинаться с адреса 300h, это связано с тем, что с адреса 0h здесь лежит заголовок и таблица настроек.

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

В «хорошем» EXE файле уже исправлено объединение всех сегментов, теперь они разнесены по разным: сегмент стека, сегмент данных и сегмент кода, в отличии от «плохого». Также отсутствие обязательной для COM файлов директивы ORG 100h приводит к тому, что память под управляющие структуры (PSP) не выделяется.

Загрузка COM модуля в основную память.

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

Для загрузки модуля COM выделялось 256 байт, которые он и занимает блоком данных PSP, располагая код программы после этого блока. Следовательно, код будет располагаться с адреса 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP, занимая 100h байт.

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

Все сегментные регистры (CS, SS, DS, ES) указывают на начало блока PSP .

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

Автоматически. SS (сегментный регистр стека) указывает на начало PSP(0h), а SP(указатель на стек) указывает на последнюю доступную ячейку памяти в сегменте, это становится началом стека (FFFFh).

Загрузка «хорошего» EXE модуля в основную память.

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

«Хороший» EXE модуль загружается, считывая при этом заголовок EXE и выполняя перемещение адресов сегментов. Регистры DS и ES устанавливаются в начало PSP. SS устанавливается на начало сегмента стека. CS устанавливается на начало сегмента кода. В IP загружается смещение точки входа в программу. PSP строится загрузчиком и содержит системную информацию. А управление передается загруженной задаче по адресу, указанному в заголовке.

2) На что указывают регистры DS и ES?

На начало PSP блока.

3) Как определяется стек?

Стек определяется при объявлении сегмента стека, в котором необходимо указать, сколько памяти будет выделено.

4) Как определяется точка входа?

Параметром после директивы END в конце программы.

Выводы.

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.
ИСХОДНЫЙ КОД.

Название файла: com.asm

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

SYSTEM_VERSION db '     SYSTEM VERSION:  .  ',0DH, 0AH, '$'
OEM db '    OEM SERIAL NUMBER:    ',0DH, 0AH, '$'
SERIAL_NUMBER db '      USER SERIAL NUMBER:             ',0DH, 0AH, '$'
VERSION db 'VERSION MS DOS: ', 0DH, 0AH, '$'

TYPE_PC db 'TYPE OF IBM PC:   PC', 0DH, 0AH, '$'
TYPE_PC_XT db 'TYPE OF IBM PC:      PC/XT', 0DH, 0AH, '$'
TYPE_AT db 'TYPE OF IBM PC:   AT', 0DH, 0AH, '$'
TYPE_PS2 db 'TYPE OF IBM PC:  PS2 model 30', 0DH, 0AH, '$'
TYPE_PS2_80 db 'TYPE OF IBM PC:     PS2 model 80', 0DH, 0AH, '$'
TYPE_PCjr db 'TYPE OF IBM PC: PCjr', 0DH, 0AH, '$'
TYPE_PC_conv db 'TYPE OF IBM PC:    PC Convertible', 0DH, 0AH, '$'

ERROR db 'Error: The received byte does not match any type! ', 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 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

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
      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_1
```

```
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

PRINT_STRING      PROC near
      push AX
      mov ah, 09h
      int 21h
      pop AX
      ret
PRINT_STRING      ENDP

PRINT_TYPE  PROC near
      push AX
      push ES
      push DX

      mov AX, 0F000h
      mov ES, AX
      mov AL, ES:[0FFFEh]

      cmp al, 0FFh
      je PC

      cmp al, 0FEh
      je PC_XT

      cmp al, 0FBh
      je PC_XT

      cmp al, 0FCh
      je AT

      cmp al, 0FAh
      je PS2_30

      cmp al, 0F8h
      je PS2_80
```

```
      cmp al, 0FDh
      je PCjr

      cmp al, 0F9h
      je PC_Conv

PC:
      mov DX, offset TYPE_PC
      jmp END_PRINT

PC_XT:
      mov DX, offset TYPE_PC_XT
      jmp END_PRINT

AT:
      mov DX, offset TYPE_AT
      jmp END_PRINT

PS2_30:
      mov DX, offset TYPE_PS2
      jmp END_PRINT

PS2_80:
      mov DX, offset TYPE_PS2_80
      jmp END_PRINT

PCjr:
      mov DX, offset TYPE_PCjr
      jmp END_PRINT

PC_Conv:
      mov DX, offset TYPE_PC_Conv
      jmp END_PRINT

      mov DX, offset ERROR

END_PRINT:
      call PRINT_STRING
      pop DX
      pop ES
      pop AX
```

```
      ret

PRINT_TYPE  ENDP

VERSION_DOS PROC near
      push AX
      push DX

      mov AH, 30h
      int 21h

      push AX
      push SI

      lea SI, SYSTEM_VERSION
      add SI, 17
      call BYTE_TO_DEC
      add SI, 3
      mov AL, AH
      call BYTE_TO_DEC

      mov DX, offset SYSTEM_VERSION
      call PRINT_STRING

      pop SI
      pop AX

      ;---------------------------------------
      push AX
      push SI

      mov AL, BH
      lea SI, OEM
      add SI, 22
      call BYTE_TO_DEC

      mov DX, offset OEM
      call PRINT_STRING

      pop SI
      pop AX
```

```
      ;---------------------------------------
      push AX
      push DI

      mov AL, BL
      call BYTE_TO_HEX
      lea DI, SERIAL_NUMBER
      add DI, 21
      mov [DI], AX
      mov AX, CX
      lea DI, SERIAL_NUMBER
      add DI, 25
      call WRD_TO_HEX

      mov DX, offset SERIAL_NUMBER
      call PRINT_STRING

      pop DI
      pop AX
      ;---------------------------------------

      pop DX
      pop AX
      ret

VERSION_DOS ENDP


BEGIN:
      xor AX, AX

      call PRINT_TYPE
      mov DX, offset VERSION
      call PRINT_STRING
      call VERSION_DOS

      xor AL, AL
      mov AH, 4Ch
      int 21h

TESTPC      ENDS
      END START
```

Название файла: exe.asm

```
ASTACK SEGMENT STACK
      db 256 DUP(?)
ASTACK ENDS

DATA SEGMENT

      SYSTEM_VERSION db '     SYSTEM VERSION:  .   ',0DH, 0AH, '$'
      OEM db '    OEM SERIAL NUMBER:    ',0DH, 0AH, '$'
      SERIAL_NUMBER db '      USER SERIAL NUMBER:             ',0DH, 0AH, '$'
      VERSION db 'VERSION MS DOS: ', 0DH, 0AH, '$'

      TYPE_PC db 'TYPE OF IBM PC:   PC', 0DH, 0AH, '$'
      TYPE_PC_XT db 'TYPE OF IBM PC:      PC/XT', 0DH, 0AH, '$'
      TYPE_AT db 'TYPE OF IBM PC:   AT', 0DH, 0AH, '$'
      TYPE_PS2 db 'TYPE OF IBM PC:  PS2 model 30', 0DH, 0AH, '$'
      TYPE_PS2_80 db 'TYPE OF IBM PC:     PS2 model 80', 0DH, 0AH, '$'
      TYPE_PCjr db 'TYPE OF IBM PC: PCjr', 0DH, 0AH, '$'
      TYPE_PC_conv db 'TYPE OF IBM PC:    PC Convertible', 0DH, 0AH, '$'

      ERROR db 'Error: The received byte does not match any type! ', 0DH, 0AH,
'$'

DATA ENDS

TESTPC      SEGMENT
      ASSUME CS:TESTPC, 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 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

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
      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_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

PRINT_STRING      PROC near
      push AX
      mov ah, 09h
      int 21h
      pop AX
      ret
PRINT_STRING      ENDP

PRINT_TYPE  PROC near
      push AX
      push ES
      push DX

      mov AX, 0F000h
      mov ES, AX
      mov AL, ES:[0FFFEh]

      cmp al, 0FFh
      je PC

      cmp al, 0FEh
      je PC_XT

      cmp al, 0FBh
      je PC_XT

      cmp al, 0FCh
      je AT

      cmp al, 0FAh
```

```
      je PS2_30

      cmp al, 0F8h
      je PS2_80

      cmp al, 0FDh
      je PCjr

      cmp al, 0F9h
      je PC_Conv

PC:
      mov DX, offset TYPE_PC
      jmp END_PRINT

PC_XT:
      mov DX, offset TYPE_PC_XT
      jmp END_PRINT

AT:
      mov DX, offset TYPE_AT
      jmp END_PRINT

PS2_30:
      mov DX, offset TYPE_PS2
      jmp END_PRINT

PS2_80:
      mov DX, offset TYPE_PS2_80
      jmp END_PRINT

PCjr:
      mov DX, offset TYPE_PCjr
      jmp END_PRINT

PC_Conv:
      mov DX, offset TYPE_PC_Conv
      jmp END_PRINT

      mov DX, offset ERROR

END_PRINT:
```

```
      call PRINT_STRING
      pop DX
      pop ES
      pop AX
      ret

PRINT_TYPE  ENDP

VERSION_DOS PROC near
      push AX
      push DX

      mov AH, 30h
      int 21h

      push AX
      push SI

      lea SI, SYSTEM_VERSION
      add SI, 17
      call BYTE_TO_DEC
      add SI, 3
      mov AL, AH
      call BYTE_TO_DEC

      mov DX, offset SYSTEM_VERSION
      call PRINT_STRING

      pop SI
      pop AX


      ;---------------------------------------
      push AX
      push SI

      mov AL, BH
      lea SI, OEM
      add SI, 22
      call BYTE_TO_DEC

      mov DX, offset OEM
```

```
        call PRINT_STRING

        pop SI
        pop AX
        ;---------------------------------------
        push AX
        push DI

        mov AL, BL
        call BYTE_TO_HEX
        lea DI, SERIAL_NUMBER
        add DI, 21
        mov [DI], AX
        mov AX, CX
        lea DI, SERIAL_NUMBER
        add DI, 25
        call WRD_TO_HEX

        mov DX, offset SERIAL_NUMBER
        call PRINT_STRING

        pop DI
        pop AX
        ;---------------------------------------

        pop DX
        pop AX
        ret

VERSION_DOS ENDP


BEGIN PROC far

        sub AX, AX
        mov AX, DATA
        mov DS, AX

        call PRINT_TYPE
        mov DX, offset VERSION
        call PRINT_STRING
        call VERSION_DOS
```

```
      xor AL, AL
      mov AH, 4Ch
      int 21h
      ret

BEGIN ENDP
TESTPC      ENDS
      END BEGIN
```