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»

Студент гр. 9383 _____ Ноздрин В.Я.

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.

Задание.

Написать тексты исходных .COM и .EXE модулей, которые выводят информацию о системе.

Основные теоретические положения.

Тип IMB PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Ниже приведены коды и типы IMB PC.

1. Тип PC – код FF
2. Тип PC/XT – код FE или FB
3. Тип AT – код FC
4. Тип PS2 модель 30 – код FA
5. Тип PS2 модель 50 или 60 – код FC
6. Тип PS2 модель 80 – код F8
7. Тип PCjr – код FD
8. Тип PC Convertible – код F9

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 – 21-битовый серийный номер пользователя

Выполнение работы.

Шаг 1. Написан текст исходного .COM модуля lab1_com.asm. С помощью компилятора MASM.EXE и компоновщика LINK.EXE был получен «плохой» .EXE модуль. Далее с помощью исполняемого файла BIN2EXE.EXE из «плохого» .EXE модуля был получен «хороший» .COM модуль lab1_com.com.

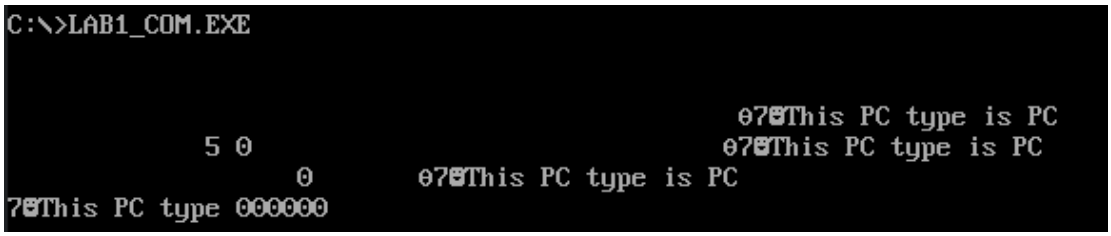

Рис. 1 – Пример работы «плохого» .EXE модуля

Рис. 2 – Пример работы «хорошего» .COM модуля

Шаг 2. Написан текст исходного .EXE модуля lab1_exe.asm. С помощью компилятора MASM.EXE и компоновщика LINK.EXE был получен «хороший» .EXE модуль.

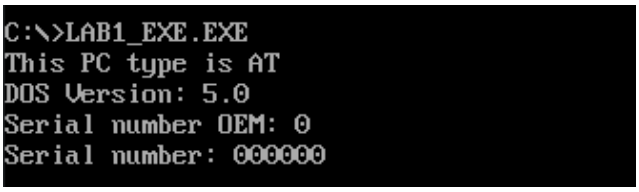

Рис. 3 – Пример работы «хорошего» .EXE модуля

Используемые функции

TETR_TO_HEX – функция, переводящая десятичную цифру в код символа.

BYTE_TO_HEX – функция, переводящая байт в шестнадцатеричной системе счисления в код символа.

WRD_TO_HEX – функция, переводящая шестнадцатеричное число в символьный код.

BYTE_TO_DEC – функция, переводящая байт в шестнадцатеричной системе счисления в символьныый код десятичной системы счисления.

PRINT_STRING – функция, выводящая строку на экран

PRINT_PC_TYPE – функция, выводящая на экран тип IMB PC

PRINT_OS_VERSION – функция, выводящая на экран версию MS DOS и серийный номер.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-програма?

COM-программа должна содержать один сегмент. В нем хранятия данные и код, а стек устанавливается автоматически на последнюю ячейку сегмента.

2) Сколько сегментов должна содержать EXE-програма?

EXE-программа может состоять как из одного так и из большего числа сегментов. Это зависит от выбранной модели памяти.

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

В тексте COM-программы обязательно должна быть директива ORG 100h, резервирующая 256 байт (100h в шестнадцатеричной системе счисления) памяти под разного рода управляющие структуры. Также должна быть директива ASSUME, сопоставляющая сегментные регистры и програмные сегменты. И директива END для завершения программы.

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

В COM-программе нельзя использовать команды типа SEG NAME, где NAME – название сегмента. В COM-программах нет таблицы настройки, с помощью которой в момент запуска программы загрузчик определяет и подставляет адреса сегментов. Также нельзя искользовать процедуры FAR, так как в COM-программах только один сегмент.

Отличия форматов файлов COM и EXE модулей

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

COM файл содержит код, данные и стек в одном сегменте. Код располагается с вдреса 0 (см рис. 4).

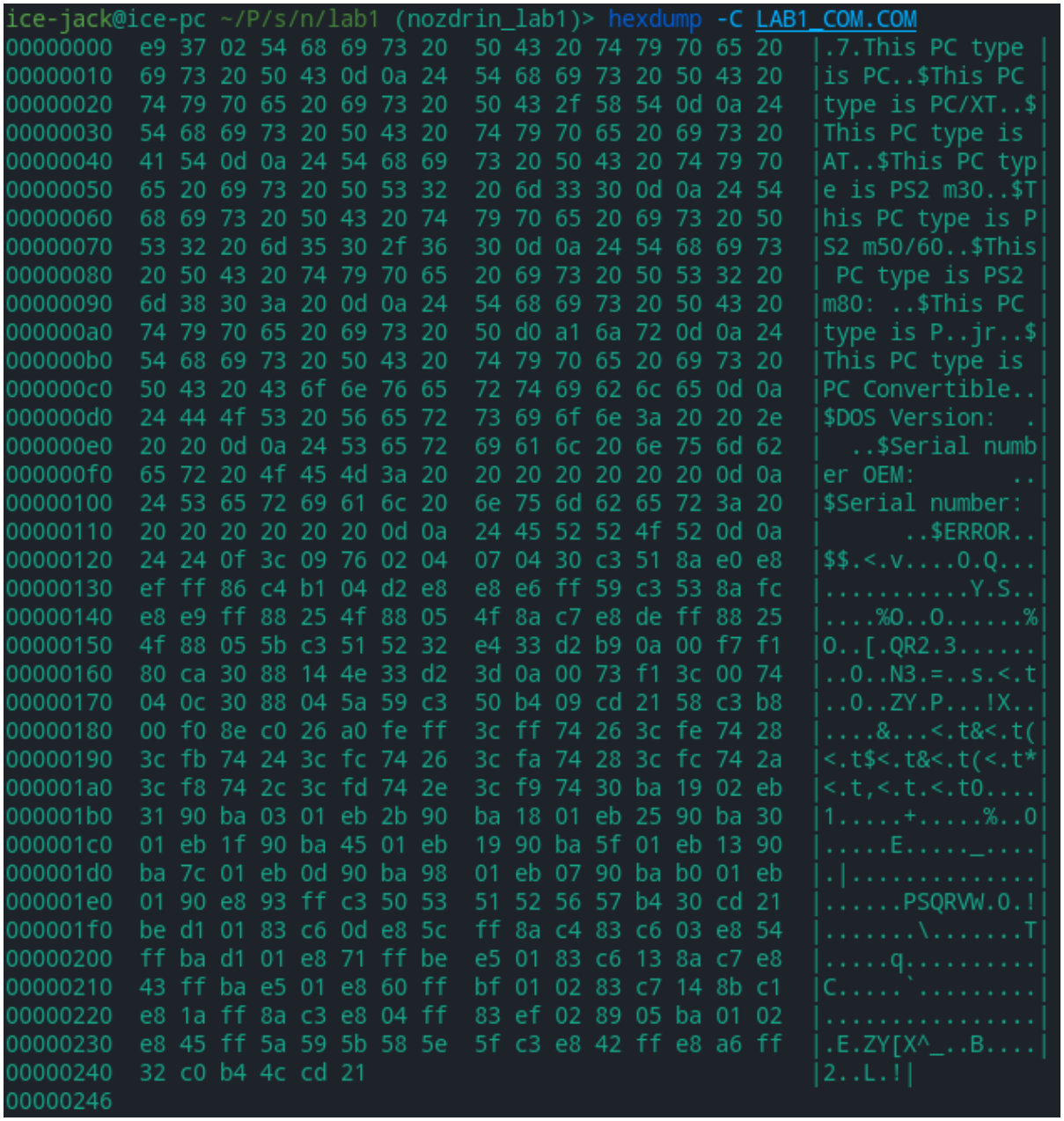

Рис. 4 – шестнадцатеричное представление файла lab1_com.com

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

«Плохой» EXE модуль содержит код, данные и стек в одном сегменте. Код располагается с вдреса 300h (см рис. 5). С адреса 0 располагается управляющая информация загрузчика, заголовок и таблица настроек (см рис. 5).

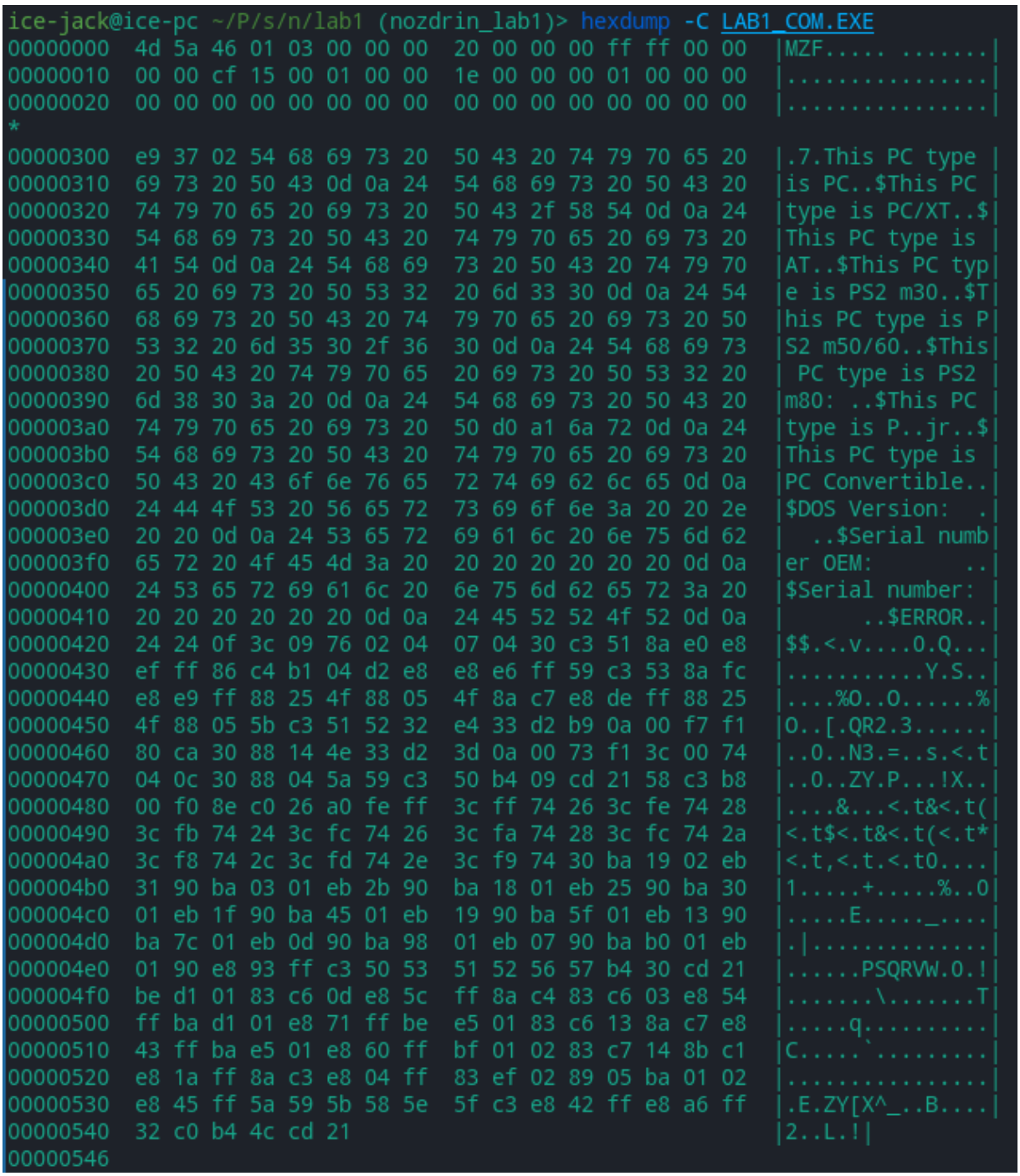

Рис. 5 – шестнадцатеричное представление файла lab1_com.exe

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

«Хороший» EXE файл содержит код, данные и стек в отдельных сегментах. Это отличает его от «плохого» EXE модуля в котором они хранятся в одном сегменте. В «хорошем» EXE файле код начинается с адреса 400h после заголовка с таблицей настроек (200h) и стека (200h) (см. рис.6). PSP на диске не хранится, память для него выделяется при запуске исполняемого файла.

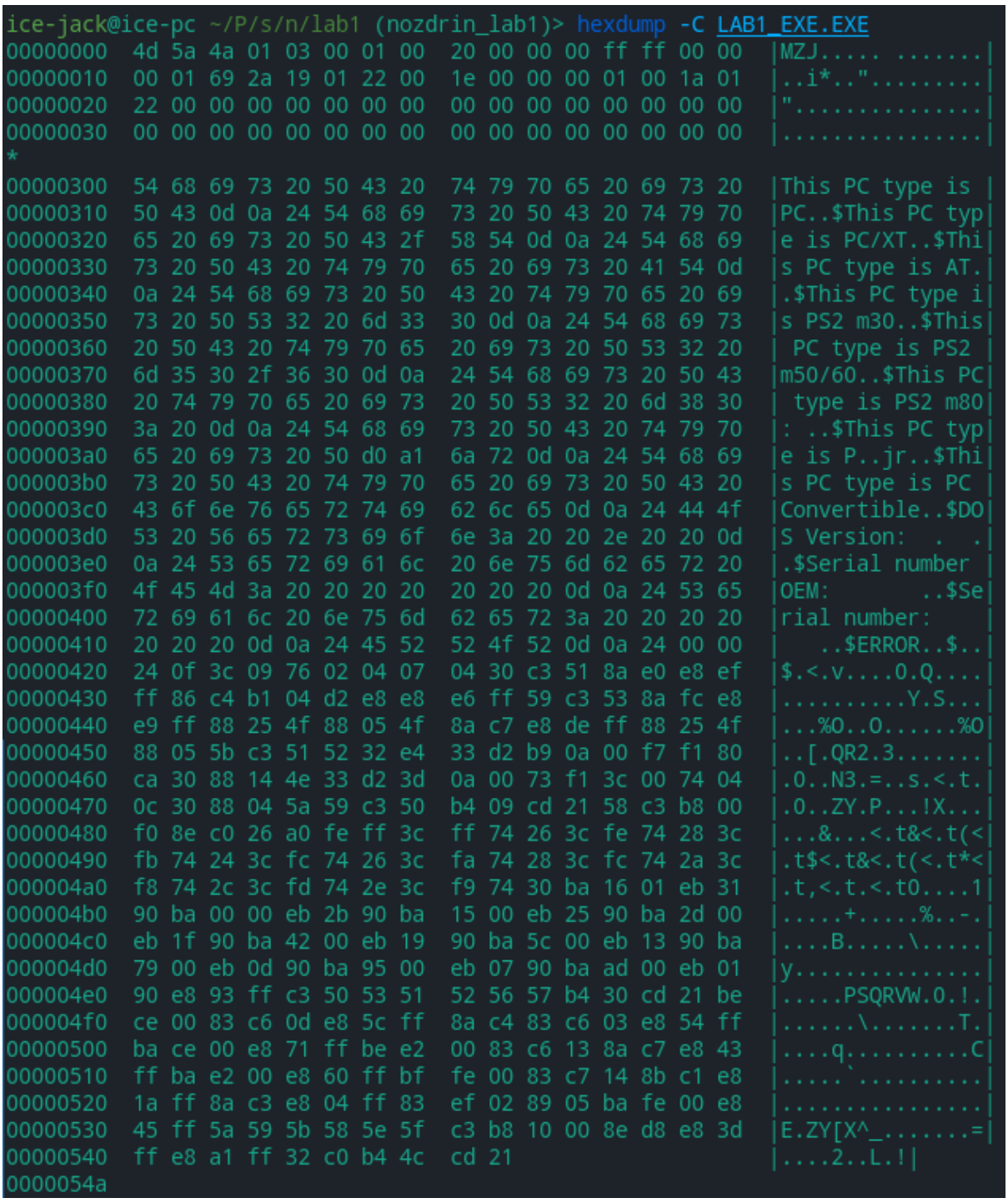

Рис. 6 – шестнадцатеричное представление файла lab1_exe.exe

Загрузка COM модуля в основную память

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

Первые 256 байт (100h) занимает блок PSP, далее, с адреса 100h располагается располагается код.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP размером в 256 байт (100h).

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

Сегментные регистры CS, SS, DS, ES – кода, стека, данных и дополнительный, все указывают на начало блока PSP.

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

Стек в .COM модулях определяется автоматически. Сегментный регистр стека (регистр SS) устанавливается на начало блока PSP, а указатель на стек (регистр SP) – на последнюю доступную ячейку памяти в сегменте. То есть адреса с 0h по FFFFh.

Загрузка «хорошего» EXE модуля в основную память

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

При загрузке «хорошего» EXE модуля считвается заголовок EXE, выполняется перемещение адресов сегментов, регистры DS и ES устанавливаются в начало PSP, в IP загружается смежение точки входа в программу.

2) На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало PSP.

3) Как определяется стек?

Стек определяется при объявлении сегмента стека, где необходимо указать сколько памяти будет выделено. Регистры SS и SP указывают на начало и конец стека соответственно.

4) Как определяется точка входа?

Точка входа определяется параметром директивы END в конце программы

Выводы.

В процессе выполнения лабораторной работы были изучены структуры и отличительные черты и способы загрузки в основную память загрузочных модулей типа .COM и .EXE.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: lab1_com.asm

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN
; ДАННЫЕ
TYPE_PC                          DB    'This  PC  type  is
PC'        ,0DH,0AH,'$'
TYPE_PC_XT                      DB    'This  PC  type  is
PC/XT'     ,0DH,0AH,'$'
TYPE_AT                          DB    'This  PC  type  is
AT'        ,0DH,0AH,'$'
TYPE_PS2_M30                DB    'This  PC  type  is  PS2
m30'   ,0DH,0AH,'$'
TYPE_PS2_M50_M60          DB     'This  PC  type  is  PS2
m50/60',0DH,0AH,'$'
TYPE_PS2_M80             DB   'This  PC  type  is  PS2  m80:
' ,0DH,0AH,'$'
TYPE_PC_jr                       DB    'This  PC  type  is
PCjr'      ,0DH,0AH,'$'
TYPE_PC_CNV                  DB    'This  PC  type  is  PC
Convertible',0DH,0AH,'$'
DOS_VERSION                      DB    'DOS  Version:    .
'         ,0DH,0AH,'$'
SERIAL_NUMBER_OEM          DB       'Serial   number   OEM:
' ,0DH,0AH,'$'
USER_NUMBER                          DB     'Serial  number:
'     ,0DH,0AH,'$'
ERROR                                                    DB
'ERROR'                   ,0DH,0AH,'$'
```

```
     ; 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
     ; Перевод в 16 с/с 16-ти разрядного числа
     ; В AX - число, DI - адрес последнего символа
         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 10 с/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
    ;-------------------------------------------------------------
------------------
    ;-------------------------------------------------------------
------------------
    ;-------------------------------------------------------------
------------------
    PRINT_STRING PROC near
        PUSH    AX
        MOV     AH,09h
        INT     21h
        pop     AX
        RET
    PRINT_STRING endp
    ;-------------------------------------------------------------
------------------
    PRINT_PC_TYPE PROC near
    ; тип IBM PC хранится в байте по адресу 0F000:0FFFEh
    ; в предпоследнем байте ROM BIOS
        MOV     AX, 0F000h
        MOV     ES, AX
        MOV     AL, ES:[0FFFEh]
    ;-------------------------------------------------------------
------------------
        cmp     AL, 0ffh
        je      print_pc
        cmp     AL, 0feh
        je      print_pc_xt
        cmp     AL, 0fbh
        je      print_pc_xt
        cmp     AL, 0fch
```

```
    je      print_pc_at
    cmp     AL, 0fah
    je      print_pc_ps2_m30
    cmp     AL, 0fch
    je      print_pc_ps2_m50_m60
    cmp     AL, 0f8h
    je      print_pc_ps2_m80
    cmp     AL, 0fdh
    je      print_pc_jr
    cmp     AL, 0f9h
    je      print_pc_cnv
    MOV     DX, offset ERROR
    jmp     print_type
print_pc:
    MOV     DX, offset TYPE_PC
    jmp     print_type
print_pc_xt:
    MOV     DX, offset TYPE_PC_XT
    jmp     print_type
print_pc_at:
    MOV     DX, offset TYPE_AT
    jmp     print_type
print_pc_ps2_m30:
    MOV     DX, offset TYPE_PS2_M30
    jmp     print_type
print_pc_ps2_m50_m60:
    MOV     DX, offset TYPE_PS2_M50_M60
    jmp     print_type
print_pc_ps2_m80:
    MOV     DX, offset TYPE_PS2_M80
    jmp     print_type
print_pc_jr:
    MOV     DX, offset TYPE_PC_jr
    jmp     print_type
```

```
    print_pc_cnv:
        MOV     DX, offset TYPE_PC_CNV
        jmp     print_type
    print_type:
        CALL    PRINT_STRING
        RET
    PRINT_PC_TYPE ENDP
    ;-------------------------------------------------------------
------------------
    PRINT_OS_VERSION PROC near
    ; Сохранение значений регистров
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
    ; Определении версии DOS
        MOV     AH, 30h
        INT     21h
    ; Версия ОС
        MOV     SI, offset DOS_VERSION
        ADD     SI, 13
        CALL    BYTE_TO_DEC
        MOV     AL, AH
        ADD     SI, 3
        CALL    BYTE_TO_DEC
        MOV     DX, offset DOS_VERSION
        CALL    PRINT_STRING
    ; Серийный номер OEM
        MOV SI, offset SERIAL_NUMBER_OEM
        ADD SI, 19
        MOV AL, bh
        CALL BYTE_TO_DEC
```

```
        MOV DX, offset SERIAL_NUMBER_OEM
        CALL PRINT_STRING
    ; Серийный номер пользователя
        MOV DI, offset USER_NUMBER
        ADD DI, 20
        MOV AX, CX
        CALL WRD_TO_HEX
        MOV AL, bl
        CALL BYTE_TO_HEX
        sub DI, 2
        MOV [DI], AX
        MOV DX, offset USER_NUMBER
        CALL PRINT_STRING
    ; Восстановление значений регистров
        pop DX
        pop CX
        pop BX
        pop AX
        pop SI
        pop DI
        RET
    PRINT_OS_VERSION ENDP
    ;-------------------------------------------------------------
------------------
    ; КОД
    BEGIN:
    ; Задание лабораторной работы
        CALL PRINT_PC_TYPE
        CALL PRINT_OS_VERSION
    ; Выход в MS_DOS
        xor     AL,AL
        MOV     AH,4Ch
        INT     21h
```

```
TESTPC  ENDS
        END     START   ; конец модуля, START - точка входа
```

Название файла: lab1_exe.asm

```
AStack    SEGMENT  STACK
          DW 128 DUP(?)
AStack    ENDS

DATA  SEGMENT
; ДАННЫЕ
           TYPE_PC                      DB    'This  PC  type  is
PC'        ,0DH,0AH,'$'
           TYPE_PC_XT                   DB    'This  PC  type  is
PC/XT'     ,0DH,0AH,'$'
           TYPE_AT                      DB    'This  PC  type  is
AT'        ,0DH,0AH,'$'
           TYPE_PS2_M30              DB    'This  PC  type  is  PS2
m30'   ,0DH,0AH,'$'
           TYPE_PS2_M50_M60        DB    'This  PC  type  is  PS2
m50/60',0DH,0AH,'$'
          TYPE_PS2_M80           DB   'This  PC  type  is  PS2  m80:
' ,0DH,0AH,'$'
           TYPE_PC_jr                   DB    'This  PC  type  is
PCjr'      ,0DH,0AH,'$'
           TYPE_PC_CNV               DB    'This  PC  type  is  PC
Convertible',0DH,0AH,'$'
           DOS_VERSION                  DB    'DOS  Version:    .
'         ,0DH,0AH,'$'
              SERIAL_NUMBER_OEM       DB     'Serial  number  OEM:
' ,0DH,0AH,'$'
             USER_NUMBER                  DB    'Serial  number:
'     ,0DH,0AH,'$'
                ERROR                                          DB
'ERROR'                    ,0DH,0AH,'$'
DATA ENDS
```

```
     CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:AStack

     ; 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
     ; Перевод в 16 с/с 16-ти разрядного числа
```

```
    ; В AX - число, DI - адрес последнего символа
        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 10 с/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
    ;--------------------------------------------------------------
------------------
    ;--------------------------------------------------------------
------------------
    ;--------------------------------------------------------------
------------------
    PRINT_STRING PROC near
        PUSH    AX
        MOV     AH,09h
        INT     21h
        pop     AX
        RET
    PRINT_STRING endp
    ;--------------------------------------------------------------
------------------
    PRINT_PC_TYPE PROC near
    ; тип IBM PC хранится в байте по адресу 0F000:0FFFEh
    ; в предпоследнем байте ROM BIOS
        MOV     AX, 0F000h
        MOV     ES, AX
        MOV     AL, ES:[0FFFEh]
    ;--------------------------------------------------------------
------------------
        cmp     AL, 0ffh
        je      print_pc
        cmp     AL, 0feh
```

```
    je      print_pc_xt
    cmp     AL, 0fbh
    je      print_pc_xt
    cmp     AL, 0fch
    je      print_pc_at
    cmp     AL, 0fah
    je      print_pc_ps2_m30
    cmp     AL, 0fch
    je      print_pc_ps2_m50_m60
    cmp     AL, 0f8h
    je      print_pc_ps2_m80
    cmp     AL, 0fdh
    je      print_pc_jr
    cmp     AL, 0f9h
    je      print_pc_cnv
    MOV     DX, offset ERROR
    jmp     print_type
print_pc:
    MOV     DX, offset TYPE_PC
    jmp     print_type
print_pc_xt:
    MOV     DX, offset TYPE_PC_XT
    jmp     print_type
print_pc_at:
    MOV     DX, offset TYPE_AT
    jmp     print_type
print_pc_ps2_m30:
    MOV     DX, offset TYPE_PS2_M30
    jmp     print_type
print_pc_ps2_m50_m60:
    MOV     DX, offset TYPE_PS2_M50_M60
    jmp     print_type
print_pc_ps2_m80:
    MOV     DX, offset TYPE_PS2_M80
```

```
        jmp     print_type
    print_pc_jr:
        MOV     DX, offset TYPE_PC_jr
        jmp     print_type
    print_pc_cnv:
        MOV     DX, offset TYPE_PC_CNV
        jmp     print_type
    print_type:
        CALL    PRINT_STRING
        RET
    PRINT_PC_TYPE ENDP
    ;-------------------------------------------------------------
------------------
    PRINT_OS_VERSION PROC near
    ; Сохранение значений регистров
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
    ; Определении версии DOS
        MOV     AH, 30h
        INT     21h
    ; Версия ОС
        MOV     SI, offset DOS_VERSION
        ADD     SI, 13
        CALL    BYTE_TO_DEC
        MOV     AL, AH
        ADD     SI, 3
        CALL    BYTE_TO_DEC
        MOV     DX, offset DOS_VERSION
        CALL    PRINT_STRING
    ; Серийный номер OEM
```

```
        MOV SI, offset SERIAL_NUMBER_OEM
        ADD SI, 19
        MOV AL, bh
        CALL BYTE_TO_DEC
        MOV DX, offset SERIAL_NUMBER_OEM
        CALL PRINT_STRING
    ; Серийный номер пользователя
        MOV DI, offset USER_NUMBER
        ADD DI, 20
        MOV AX, CX
        CALL WRD_TO_HEX
        MOV AL, bl
        CALL BYTE_TO_HEX
        sub DI, 2
        MOV [DI], AX
        MOV DX, offset USER_NUMBER
        CALL PRINT_STRING
    ; Восстановление значений регистров
        pop DX
        pop CX
        pop BX
        pop AX
        pop SI
        pop DI
        RET
    PRINT_OS_VERSION ENDP
    ;-------------------------------------------------------------
------------------
    ; КОД
    Main PROC FAR
        MOV     AX, DATA
        mov     DS, AX
    ; Задание лабораторной работы
        CALL PRINT_PC_TYPE
```

```
    CALL PRINT_OS_VERSION
; Выход в MS_DOS
    xor     AL,AL
    MOV     AH,4Ch
    INT     21h
Main ENDP

CODE    ENDS
        END     Main
```