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 _____ Соседков К.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.

Задание.

Шаг 1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

Для выполнения первого шага работы на языке ассемблера был написан исходный код модуля COM определяющий тип PC и версию системы(Рисунок 1). Кроме этого был получен «плохой» EXE модуль, полученный из исходного кода для COM модуля(Рисунок 2).

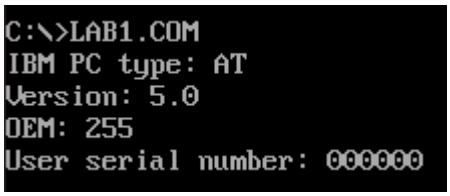

Рисунок 1: результат работы COM

Рисунок 2: результат работы „плохого“ EXE

Для выполнения второго шага был написан исходный код EXE модуля, который выполняет те же функции что и COM модуль(Рисунок 3).

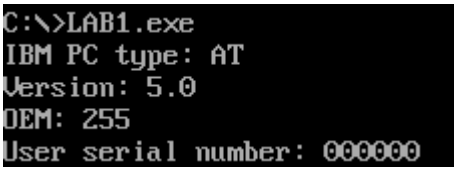

Рисунок 3: результат работы „хорошего“ EXE

Вопросы к шагу 3. «Отличия исходных текстов COM и EXE программ».

1) Сколько сегментов должна содержать COM-программа?

COM-программа ограничена размером одного сегмента

2) Сколько сегментов должна содержать EXE-программа?

EXE-программа должна содержать один сегмент или больше.

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

Обязательно должна быть директива org100h, так как в DOS первые 256 байт(100h) занимает Program Segment Prefix(PSP). Директива org100h говорит о том, что первая команда находится по адресу 100h.

Так же необходима директива ASSUME. Данная директива позволяет установить значения сегментных регистров. Как минимум необходимо установить регистр CS. Без установки этого регистра будет ошибка компиляции „Missing or unreachable CS“.

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

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>, т.к. в COM-программе отсутствует таблица настроек.

Для выполнения четвертого шага было выполнено сравнение файлов COM и EXE в шестнадцатеричном виде(см. Рисунки 4,5,6).

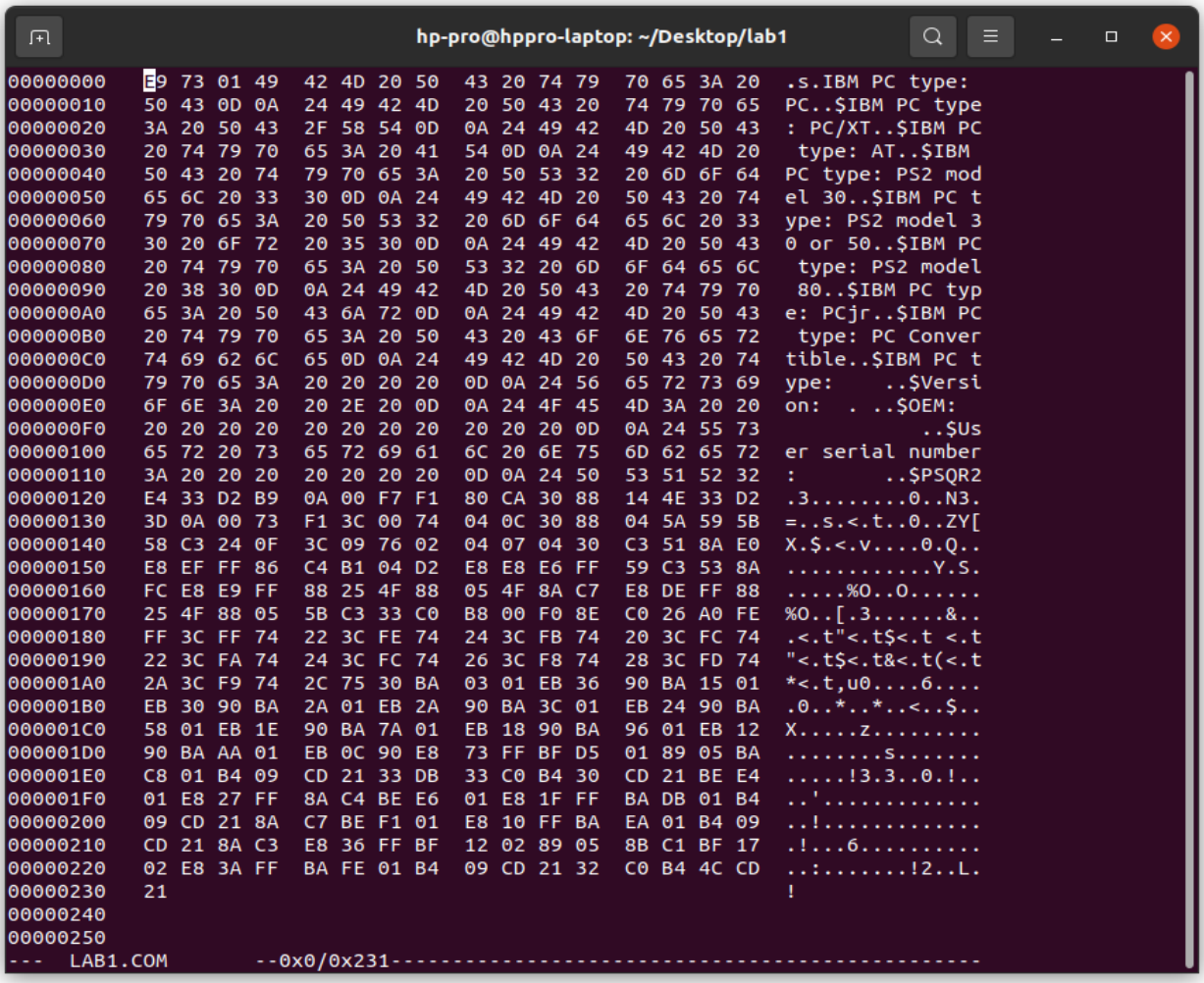

Рисунок 4: COM

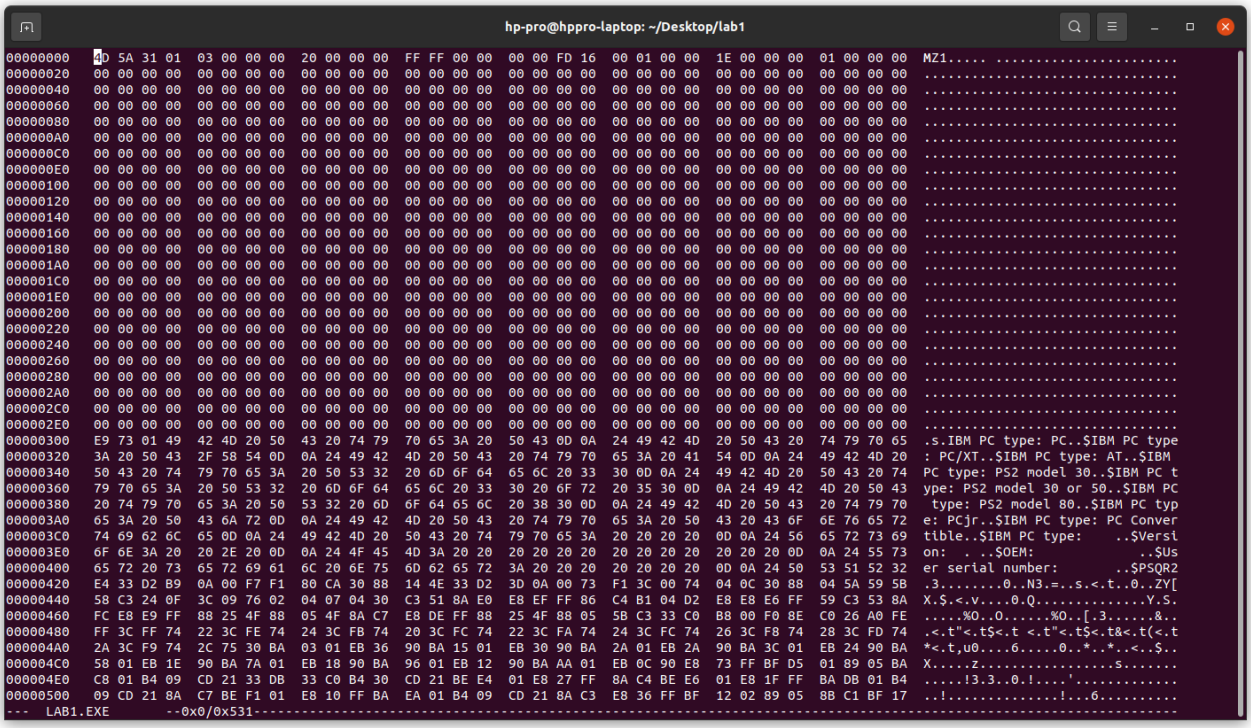

Рисунок 5: "плохой" EXE

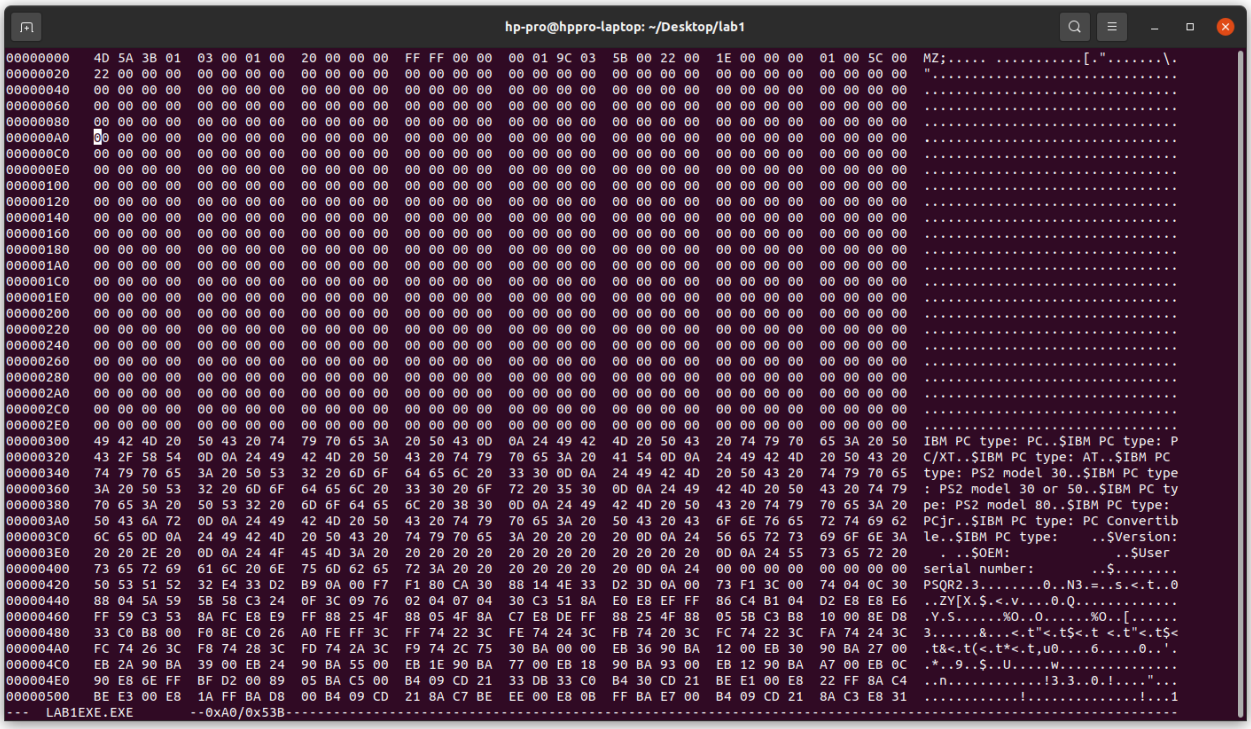

Рисунок 6: "хороший" EXE

Вопросы к шагу 4. «Отличия форматов файлов COM и EXE модулей».

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

Файл типа COM содержит команды и данные. Код располагается с адреса 0h.

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

Данные и код находятся в одном сегменте. Код расположен по адресу 300h. По адресу 0h располагается управляющая информация для загрузчика которая состоит из заголовка и таблицы настроек. Начинается управляющая информация с бита MZ. Данный бит является сигнатурой файла EXE.

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

В „хорошем“ EXE стек, данные и код разделены по сегментам, так же „хороший“ EXE не содержит директивы ORG 100h и поэтому код начинается с адреса 200h.

Вопросы к шагу 5. «Загрузка COM модуля в основную память».

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

Сначала загружается PSP а после этого код. Код располагается с адреса 100h после PSP.

2) Что располагается с адреса 0?

С адреса 0h располагается PSP.

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

При загрузке сегментные регистры указывают на начало PSP.

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

В COM модуле стек создается автоматически и занимает всю область сегмента. Адреса расположены в диапазоне 0h-ffffh.

Вопросы к шагу 6. «Загрузка «хорошего» EXE модуля в основную память».

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

Так же как и COM. Сегментные регистры DS и ES устанавливаются на начало PSP, SS - на начало сегмента стека, CS — на начало сегмента команд.

2) На что указывают регистры DS и ES?

Сегментные регистры DS и ES указывают на начало PSP

3) Как определяется стек?

С помощью регистров SS и SP.

SS указывает на начало, а SS:SP на конец.

4) Как определяется точка входа?

Точка входа определяется директивой END + <метка или процедура>.

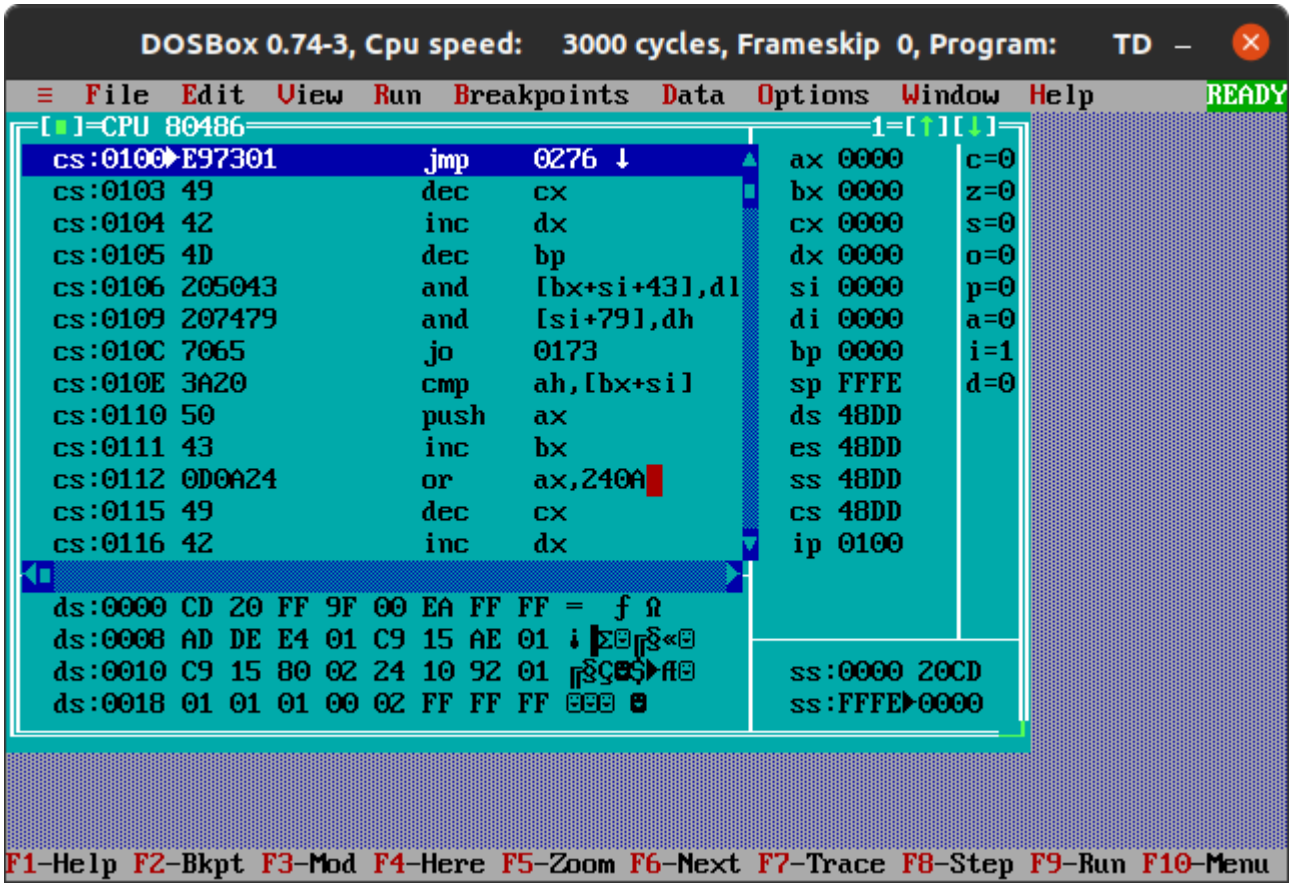

Рисунок 7: TD.exe

Выводы.

При выполнении лабораторной работы были изучены структуры загрузочных модулей COM и EXE.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab1.asm

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

START: JMP BEGIN



pc db 'IBM PC type: PC',0dh,0ah,'$'
pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'$'
at db 'IBM PC type: AT',0dh,0ah,'$'
ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'$'
ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'$'
ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'$'
pcjr db 'IBM PC type: PCjr',0dh,0ah,'$'
pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'$'
unknown db 'IBM PC type:    ',0dh,0ah,'$'
version db 'Version:  . ',0dh,0ah,'$'
oem db 'OEM:            ',0dh,0ah,'$'
user_serial_number db 'User serial number:       ',0dh,0ah,'$'



;------------------------------
BYTE_TO_DEC PROC near
    push AX
    push BX
    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
    pop BX
    pop AX
    ret
BYTE_TO_DEC ENDP
;------------------------------
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 шестн. 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
;-----------------------------



BEGIN:
    xor ax,ax
    mov ax,0f000h
    mov es,ax
    mov al,es:[0fffeh]
```

```
    cmp al,0ffh
    je label_pc
    cmp al,0feh
    je label_pc_xt
    cmp al,0fbh
    je label_pc_xt
    cmp al,0fch
    je label_at
    cmp al,0fah
    je label_ps2_30
    cmp al,0fch
    je label_ps2_50_60
    cmp al,0f8h
    je label_ps2_80
    cmp al,0fdh
    je label_pcjr
    cmp al,0f9h
    je label_pc_convertible
    jne label_unknown

label_pc:
    mov dx,offset pc
    jmp print_ibm_pc_version
label_pc_xt:
    mov dx,offset pc_xt
    jmp print_ibm_pc_version
label_at:
    mov dx,offset at
    jmp print_ibm_pc_version
label_ps2_30:
    mov dx,offset ps2_30
    jmp print_ibm_pc_version
label_ps2_50_60:
```

```
    mov dx,offset ps2_50_60
    jmp print_ibm_pc_version
label_ps2_80:
    mov dx,offset ps2_80
    jmp print_ibm_pc_version
label_pcjr:
    mov dx,offset pcjr
    jmp print_ibm_pc_version
label_pc_convertible:
    mov dx,offset pc_convertible
    jmp print_ibm_pc_version
label_unknown:
    call BYTE_TO_HEX
    mov di, offset unknown+13
    mov [di], ax
    mov dx,offset unknown


print_ibm_pc_version:
    mov ah,09h
    int 21h

    xor bx, bx
    xor ax, ax
    mov ah, 30h
    int 21h

    ;Version
    mov si, offset version+9
    call BYTE_TO_DEC

    mov al, ah
    mov si, offset version+11
    call BYTE_TO_DEC
```

```
    mov dx, offset version
    mov ah,09h
    int 21h

    ;OEM
    mov al, bh
    mov si, offset oem+7
    call BYTE_TO_DEC

    mov dx, offset oem
    mov ah,09h
    int 21h

    ;User serial number
    mov al, bl
    call BYTE_TO_HEX

    mov di, offset user_serial_number+20
    mov [di], ax

    mov ax, cx
    mov di, offset user_serial_number+25
    call WRD_TO_HEX

    mov dx, offset user_serial_number
    mov ah,09h
    int 21h

exit:
    xor al,al
    mov ah,4ch
    int 21h
TESTPC ENDS
```

```
END START
```

Название файла: lab1exe.asm

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

DATA SEGMENT
   pc db 'IBM PC type: PC',0dh,0ah,'$'
   pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'$'
   at db 'IBM PC type: AT',0dh,0ah,'$'
   ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'$'
   ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'$'
   ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'$'
   pcjr db 'IBM PC type: PCjr',0dh,0ah,'$'
   pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'$'
   unknown db 'IBM PC type:    ',0dh,0ah,'$'
   version db 'Version:  . ',0dh,0ah,'$'
   oem db 'OEM:             ',0dh,0ah,'$'
   user_serial_number db 'User serial number:       ',0dh,0ah,'$'
DATA ENDS



CODE SEGMENT
   ASSUME CS:CODE, DS:DATA, SS:AStack
;------------------------------
BYTE_TO_DEC PROC near
   push AX
   push BX
   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
    pop BX
    pop AX
    ret
BYTE_TO_DEC ENDP
;------------------------------
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 шестн. 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
;--------------------------------




Main PROC FAR
    mov ax, DATA
```

```
    mov ds, ax

    xor ax,ax
    mov ax,0f000h
    mov es,ax
    mov al,es:[0fffeh]

    cmp al,0ffh
    je label_pc
    cmp al,0feh
    je label_pc_xt
    cmp al,0fbh
    je label_pc_xt
    cmp al,0fch
    je label_at
    cmp al,0fah
    je label_ps2_30
    cmp al,0fch
    je label_ps2_50_60
    cmp al,0f8h
    je label_ps2_80
    cmp al,0fdh
    je label_pcjr
    cmp al,0f9h
    je label_pc_convertible
    jne label_unknown

label_pc:
    mov dx,offset pc
    jmp print_ibm_pc_version
label_pc_xt:
    mov dx,offset pc_xt
    jmp print_ibm_pc_version
label_at:
```

```
    mov dx,offset at
    jmp print_ibm_pc_version
label_ps2_30:
    mov dx,offset ps2_30
    jmp print_ibm_pc_version
label_ps2_50_60:
    mov dx,offset ps2_50_60
    jmp print_ibm_pc_version
label_ps2_80:
    mov dx,offset ps2_80
    jmp print_ibm_pc_version
label_pcjr:
    mov dx,offset pcjr
    jmp print_ibm_pc_version
label_pc_convertible:
    mov dx,offset pc_convertible
    jmp print_ibm_pc_version
label_unknown:
    call BYTE_TO_HEX
    mov di, offset unknown+13
    mov [di], ax
    mov dx,offset unknown


print_ibm_pc_version:
    mov ah,09h
    int 21h

    xor bx, bx
    xor ax, ax
    mov ah, 30h
    int 21h

    ;Version
```

```
mov si, offset version+9
call BYTE_TO_DEC

mov al, ah
mov si, offset version+11
call BYTE_TO_DEC

mov dx, offset version
mov ah,09h
int 21h

;OEM
mov al, bh
mov si, offset oem+7
call BYTE_TO_DEC

mov dx, offset oem
mov ah,09h
int 21h

;User serial number
mov al, bl
call BYTE_TO_HEX

mov di, offset user_serial_number+20
mov [di], ax

mov ax, cx
mov di, offset user_serial_number+25
call WRD_TO_HEX

mov dx, offset user_serial_number
mov ah,09h
int 21h
```

```
exit:
    xor al,al
    mov ah,4ch
    int 21h

Main ENDP
CODE ENDS
END Main
```