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. 9382 _____ Пя С.

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.

Сведения о функциях и структурах.

TETR_TO_HEX: процедура перевода из 10-ой сс в символы

BYTE_TO_HEX: процедура перевода байта из 16-ой сс в символы

WRD_TO_HEX: перевод слова из 16-ой сс в символы

BYTE_TO_DEC: перевод байта из 16-ой сс в 10-юю и символы

print_string: процедура вывода строки на экран

print_type_of_PC: процедура вывода типа IBM PC на экран

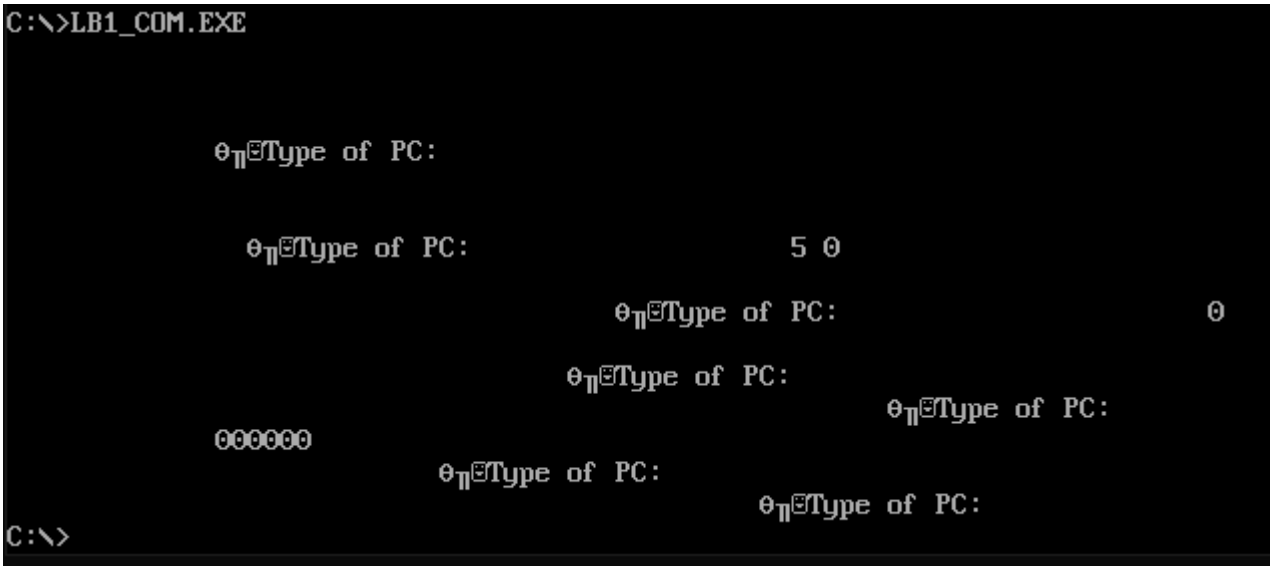
print_version_of_PC: процедура вывода версии MS DOS на экран

Последовательность действий, выполняемых утилитой.

1. Определяется тип PC путем считывания содержимого предпоследнего байта ROM BIOS с представленной таблицей в методических указаниях, и выводится строка, содержащая название соответствующего типа.
2. Определяется версия PC. Выводится строка в определенном формате, содержащая номер основной версии и номер модификации в десятичной системе счисления, затем выводится серийный номер OEM и серийный номер пользователя. Путь их определения написан в методических указаниях. Вывод «правильного» файла EXE:

```
C:\>LB1_EXE.EXE
Type of PC: AT
Version of MS DOS: 5.0
Serial number of OEM: 0
Serial number of user: 000000
```

Вывод «плохого» EXE файла:



Вывод COM файла:

Выводы.

В ходе выполнения лабораторной работы была написана программа для определения типа и версии PC, изучены различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Ответы на контрольные вопросы по лабораторной работе №1.

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать только один сегмент, стек генерируется автоматически, а привычные сегменты кода и данных содержатся в одном сегменте.

2) EXE-программа?

EXE-программа должна содержать более одного сегмента. Код, данные и стек будут храниться в отдельных сегментах.

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

В тексте COM-программы обязательно должны быть директивы ORG 100h и ASSUME, так как при загрузке модуля в начале определяется 100h префикс программного сегмента, так что адресация должна быть смещена на столько же и вторая для того, чтобы указать один сегмент как сегмент данных и кода.

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

Не все форматы команд можно использовать в COM-программе, а именно команды типа mov (регистр), seg (наименование сегмента). COM-программы имеют один сегмент, поэтому сегментные регистры имеют одни и те же значения. А EXE-программы могут быть больше 64КБ, поэтому загрузчик ОС распределяет содержимое по нескольким сегментам.

Отличия форматов файлов COM и EXE модулей

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

Файл COM состоит из одного сегмента и может быть размером не больше 64КБ. Код располагается с 0h, но при загрузке модуля смещается на 100h.

```
E:\OC 2021\LB1_COM.COM
0000000000: E9 B7 01 54 79 70 65 20 | 6F 66 20 50 43 3A 20 24 | é·▯Type of PC: $
0000000010: 50 43 0D 0A 24 50 43 2F | 58 54 0D 0A 24 41 54 0D | PC ▯$PC/XT ▯$AT
0000000020: 0A 24 50 53 32 20 6D 6F | 64 65 6C 20 33 30 0D 0A | ▯$PS2 model 30 ▯
0000000030: 24 50 53 32 20 6D 6F 64 | 65 6C 20 38 30 0D 0A 24 | $PS2 model 80 ▯$
0000000040: 50 43 6A 72 0D 0A 24 50 | 43 20 43 6F 6E 76 65 72 | PCjr ▯$PC Conver
0000000050: 74 69 62 6C 65 0D 0A 24 | 56 65 72 73 69 6F 6E 20 | tible ▯$Version
0000000060: 6F 66 20 4D 53 20 44 4F | 53 3A 20 20 2E 20 20 0D | of MS DOS:  .
0000000070: 0A 24 53 65 72 69 61 6C | 20 6E 75 6D 62 65 72 20 | ▯$Serial number
0000000080: 6F 66 20 4F 45 4D 3A 20 | 20 20 24 53 65 72 69 61 | of OEM:   $Seria
0000000090: 6C 20 6E 75 6D 62 65 72 | 20 6F 66 20 75 73 65 72 | l number of user
00000000A0: 3A 20 20 20 20 20 20 20 | 20 20 24 0D 0A 24 24 0F | :       $ ▯$$▯
00000000B0: 3C 09 76 02 04 07 04 30 | C3 51 8A E0 E8 EF FF 86 | <▯v▯▯▯▯0ÃQŠàèïÿ†
00000000C0: C4 B1 04 D2 E8 E8 E6 FF | 59 C3 53 8A FC E8 E9 FF | Ä±▯Òèèæÿ YÃSŠüèéÿ
00000000D0: 88 25 4F 88 05 4F 8A C7 | E8 DE FF 88 25 4F 88 05 | ˆ%Oˆ▯OŠÇèÞÿˆ%Oˆ▯
00000000E0: 5B C3 51 52 32 E4 33 D2 | B9 0A 00 F7 F1 80 CA 30 | [ÃQR2ä3Ò¹▯ ÷ñ€Ê0
00000000F0: 88 14 4E 33 D2 3D 0A 00 | 73 F1 3C 00 74 04 0C 30 | ˆ▯N3Ò=▯ sñ< t▯▯0
0000000100: 88 04 5A 59 C3 B4 09 CD | 21 C3 BA 03 01 E8 F5 FF | ˆ▯ZYÃ´▯Í!Ãº▯▯èõÿ
0000000110: B8 00 F0 8E C0 26 A0 FE | FF 3C FF 74 1C 3C FE 74 | ¸ ðŽÀ& þÿ<ÿt▯<þt
0000000120: 1E 3C FB 74 1A 3C FC 74 | 1C 3C FA 74 1E 3C F8 74 | ▯<ût▯<üt▯<út▯<øt
0000000130: 20 3C FD 74 22 3C F9 74 | 24 BA 10 01 EB 22 90 BA |  <ýt"<ùt$º▯▯ë"▯º
0000000140: 15 01 EB 1C 90 BA 1D 01 | EB 16 90 BA 22 01 EB 10 | ▯▯ë▯▯º▯▯ë▯▯º"▯ë▯
0000000150: 90 BA 31 01 EB 0A 90 BA | 40 01 EB 04 90 BA 47 01 | ▯º1▯ë▯▯º@▯ë▯▯ºG▯
0000000160: E8 A2 FF C3 B4 30 CD 21 | 50 BE 58 01 83 C6 13 E8 | è¢ÿÃ´0Í!P¾X▯ƒÆ▯è
0000000170: 70 FF 58 8A C4 83 C6 03 | E8 67 FF BA 58 01 E8 84 | pÿXŠÄƒÆ▯ègÿºX▯è„
0000000180: FF BE 72 01 83 C6 16 8A | C7 E8 56 FF BA 72 01 E8 | ÿ¾r▯ƒÆ▯ŠÇèVÿºr▯è
0000000190: 73 FF BA AB 01 E8 6D FF | BF 8B 01 83 C7 1C 8B C1 | sÿº«▯èmÿ¿‹▯ƒÇ▯‹Á
00000001A0: E8 27 FF 8A C3 E8 11 FF | 83 EF 02 89 05 BA 8B 01 | è'ÿŠÃè▯ÿƒï▯‰▯º‹▯
00000001B0: E8 52 FF BA AB 01 E8 4C | FF C3 E8 4D FF E8 A4 FF | èRÿº«▯èLÿÃèMÿè¤ÿ
00000001C0: 32 C0 B4 4C CD 21       |                       | 2À´LÍ!
```

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

У файла «плохого» EXE код и данные будут располагаться в одном сегменте, код будет располагаться с адреса 300h, с адреса 0h будет таблица настроек.

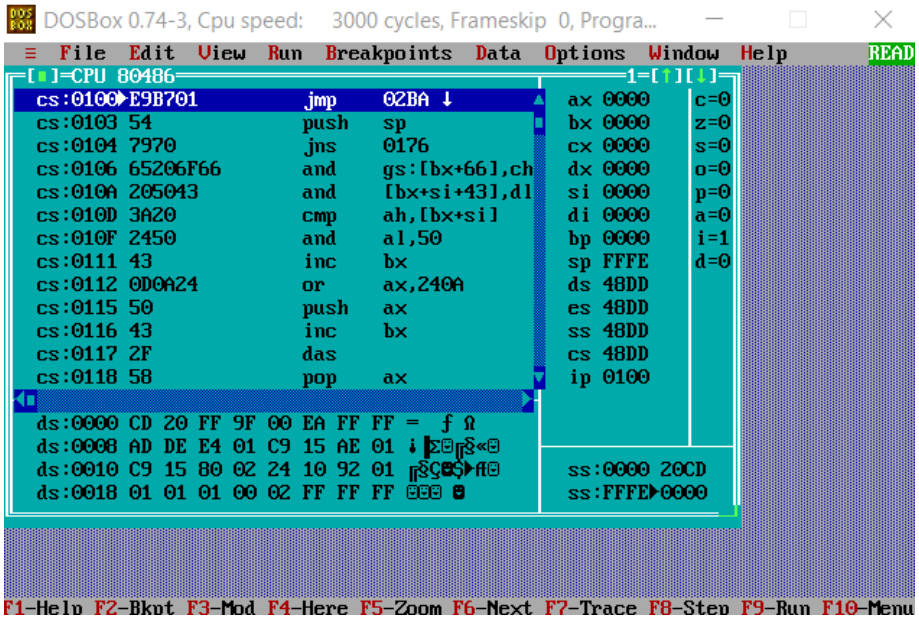
3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

У файла «хорошего» EXE данные, код и стек размещены в разных сегментах, и файл может быть большего размера, чем 64КБ. У «плохого» файла EXE код начинается с 300h, так как он получается из файла COM, в котором код смещен на 100h, а 200h – это размер PSP модуля, а у «хорошего» EXE выделяется память под стек между PSP и кодом.

Загрузка COM модуля в основную память

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

Сначала определяется сегментный адрес участка ОП, способного вместить загрузку программы, затем создается блок памяти для PSP и программы, COM-файл считывается помещается в память с 100h. После сегментные регистры устанавливаются на начало PSP. SP устанавливается на конец PSP, 0000h помещается в стек, в IP записывается 100h. Код располагается с адреса 100h.



2) Что располагается с адреса 0?

PSP сегмент располагается с адреса 0.

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

Сегментные регистры имеют значения 48DD и указывают на PSP.

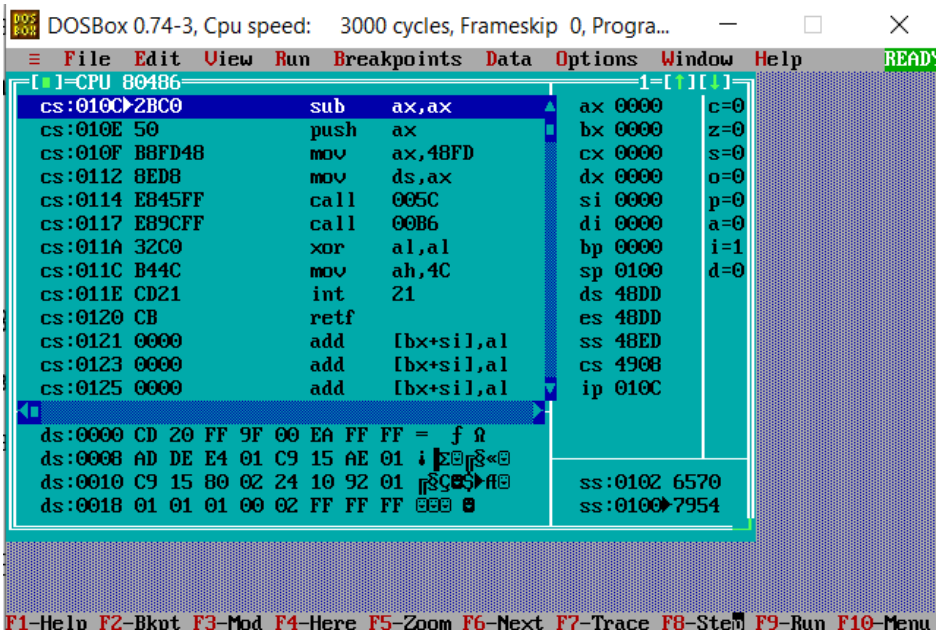
4) Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек генерируется автоматически. SP указывает на конец стека, а SS – на начало. Адреса расположены в диапазоне 0h–FFFEh (потому что это последний адрес, который кратен двум).

Загрузка «хорошего» EXE модуля в основную память

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

Загружается «хороший» EXE со считыванием информации заголовка EXE, выполняется перемещение адресов сегментов, ES и DS устанавливаются на начало PSP, SS – на начало сегмента стека, а CS – на начало сегмента команд.



2) На что указывают регистры DS и ES?

ES и DS указывают на начало PSP.

3) Как определяется стек?

Стек определяется с помощью директивы .stack с указанием размера стека. SS указывает на начало сегмента стека, а SP – на конец.

4) Как определяется точка входа?

Точка входа определяется директивой END.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Lb1_COM.ASM

```
TESTPC SEGMENT
 ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
 ORG 100H
START: JMP BEGIN
; ДАННЫЕ ТИП PC
type_of db 'Type of PC: $'
pc db 'PC',0DH,0AH,'$'
pc_xt db 'PC/XT',0DH,0AH,'$'
at db 'AT',0DH,0AH,'$'
ps2_model_30 db 'PS2 model 30',0DH,0AH,'$'
ps2_model_80 db 'PS2 model 80',0DH,0AH,'$'
pcjr db 'PCjr',0DH,0AH,'$'
pc_convertible db 'PC Convertible',0DH,0AH,'$'
; ДАННЫЕ ВЕРСИЯ PC
version_pc db 'Version of MS DOS:  .  ',0DH,0AH,'$'
serial_number_oem db 'Serial number of OEM:   $'
serial_number_of_user db 'Serial number of user:         $'
at_end db 0DH,0AH,'$'

;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 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
;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 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
;--------------------------------
; КОД
print_string proc near
    mov ah, 09h
    int 21h
    ret
print_string endp

print_type_of_PC proc near
    mov dx, offset type_of
    call print_string
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]
    cmp al, 0FFh
    je print1
    cmp al, 0FEh
    je print2
    cmp al, 0FBh
    je print2
    cmp al, 0FCh
    je print3
    cmp al, 0FAh
    je print4
    cmp al, 0F8h
    je print5
    cmp al, 0FDh
    je print6
    cmp al, 0F9h
    je print7

print1:
    mov dx, offset pc
    jmp to_end
```

```
print2:
    mov dx, offset pc_xt
    jmp to_end

print3:
    mov dx, offset at
    jmp to_end

print4:
    mov dx, offset ps2_model_30
    jmp to_end

print5:
    mov dx, offset ps2_model_80
    jmp to_end

print6:
    mov dx, offset pcjr
    jmp to_end

print7:
    mov dx, offset pc_convertible

to_end:
    call print_string
    ret
print_type_of_PC endp

print_version_of_PC proc near
    mov ah, 30h
    int 21h

    push ax
    mov si, offset version_pc
    add si, 19
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset version_pc
    call print_string

    mov si, offset serial_number_oem
    add si, 22
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset serial_number_oem
    call print_string
    mov dx, offset at_end
    call print_string

    mov di, offset serial_number_of_user
    add di, 28
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset serial_number_of_user
    call print_string
    mov dx, offset at_end
```

```
    call print_string
    ret
print_version_of_PC endp

BEGIN:
; . . . . . .
    call print_type_of_PC
    call print_version_of_PC
; . . . . . .
; Выход в DOS
 xor AL,AL
 mov AH,4Ch
 int 21H
TESTPC ENDS
 END START ;конец модуля, START - точка входа
```

Название файла: Lb1_EXE.ASM

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

DATA  SEGMENT
; ДАННЫЕ ТИП PC
type_of db 'Type of PC: $'
pc db 'PC',0DH,0AH,'$'
pc_xt db 'PC/XT',0DH,0AH,'$'
at db 'AT',0DH,0AH,'$'
ps2_model_30 db 'PS2 model 30',0DH,0AH,'$'
ps2_model_80 db 'PS2 model 80',0DH,0AH,'$'
pcjr db 'PCjr',0DH,0AH,'$'
pc_convertible db 'PC Convertible',0DH,0AH,'$'
; ДАННЫЕ ВЕРСИЯ PC
version_pc db 'Version of MS DOS:  .  ',0DH,0AH,'$'
serial_number_oem db 'Serial number of OEM:   $'
serial_number_of_user db 'Serial number of user:         $'
at_end db 0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE,DS:DATA,SS:MyStack
;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 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
;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 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
;--------------------------------
; КОД
print_string proc near
    mov ah, 09h
    int 21h
    ret
print_string endp

print_type_of_PC proc near
    mov dx, offset type_of
    call print_string
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]
    cmp al, 0FFh
    je print1
    cmp al, 0FEh
    je print2
    cmp al, 0FBh
    je print2
    cmp al, 0FCh
```

```
    je print3
    cmp al, 0FAh
    je print4
    cmp al, 0F8h
    je print5
    cmp al, 0FDh
    je print6
    cmp al, 0F9h
    je print7

print1:
    mov dx, offset pc
    jmp to_end

print2:
    mov dx, offset pc_xt
    jmp to_end

print3:
    mov dx, offset at
    jmp to_end

print4:
    mov dx, offset ps2_model_30
    jmp to_end

print5:
    mov dx, offset ps2_model_80
    jmp to_end

print6:
    mov dx, offset pcjr
    jmp to_end

print7:
    mov dx, offset pc_convertible

to_end:
    call print_string
    ret
print_type_of_PC endp

print_version_of_PC proc near
    mov ah, 30h
    int 21h

    push ax
    mov si, offset version_pc
    add si, 19
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset version_pc
    call print_string

    mov si, offset serial_number_oem
    add si, 22
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset serial_number_oem
    call print_string
    mov dx, offset at_end
```

```
    call print_string

    mov di, offset serial_number_of_user
    add di, 28
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset serial_number_of_user
    call print_string
    mov dx, offset at_end
    call print_string
    ret
print_version_of_PC endp

Main proc far
   sub   AX,AX
   push  AX
   mov   AX,DATA
   mov   DS,AX
; . . . . . .
    call print_type_of_PC
    call print_version_of_PC
; . . . . . .
; Выход в DOS
 xor AL,AL
 mov AH,4Ch
 int 21H
 ret
Main ENDP
CODE ENDS
END Main
```