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.8382 _____ Ершов М.И.

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.

Ход работы.

Был написан текст исходного **.COM** модуля, который определяет тип PC и версию системы. Для этого программа читает содержимое предпоследнего байта ROM BIOS и, в соответствии с табл. 1, определяет тип PC.

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

Таблица 1 - Соответствие кода и типа

Для определения версии системы требуется воспользоваться функцией 30H прерывания 21H, затем из регистра AL считать номер основной версии, из AH номер модификации, из BH серийный номер OEM, а из регистров BL:CX 24-битовый серийные номер пользователя.

Из исходного кода был получен «хороший» .COM модуль, а также «плохой» .EXE. После был написан текст исходного .EXE модуля, который выполняет те же функции, что и модуль, полученный ранее.

Ответы на контрольные вопросы

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

COM – программа должна содержать один сегмент (с кодом и данными).

2) Сколько сегментов должна содержать EXE-программа?

EXE-программа может содержать любое количество сегментов

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

- ORG 100h для резервирования памяти PSP
- ASSUME для инициализации регистров.

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

В COM-программах нельзя указывать адрес сегмента, так как в COM отсутствует таблица настроек**.**

Отличие форматов файлов COM и EXE модулей

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

Файл состоит из одного сегмента, в котором располагается код и данные, код располагается с адреса 100h.

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

С нулевого адреса «плохого» EXE файла располагается заголовок и таблица настроек, с адреса 300h начинается сегмент кода. На рис. 1 и 2 представлен шестнадцатеричный вид файла.

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

«Хороший» EXE в начале содержит задержит заголовок и стек, код начинается с 240h, в отличие от «плохого», у которого код начинается с 300h, а также «плохой» EXE содержит только один сегмент. На рис. 3 и 4 представлен шестнадцатеричный вид «хорошего» EXE.

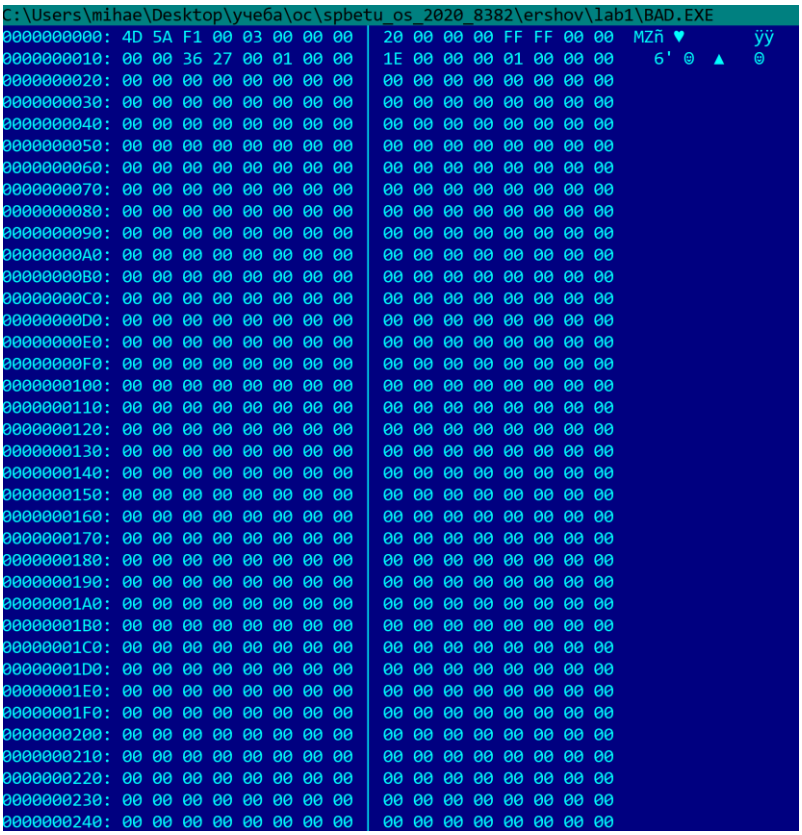

Рисунок 1 – начало «плохого» EXE файла

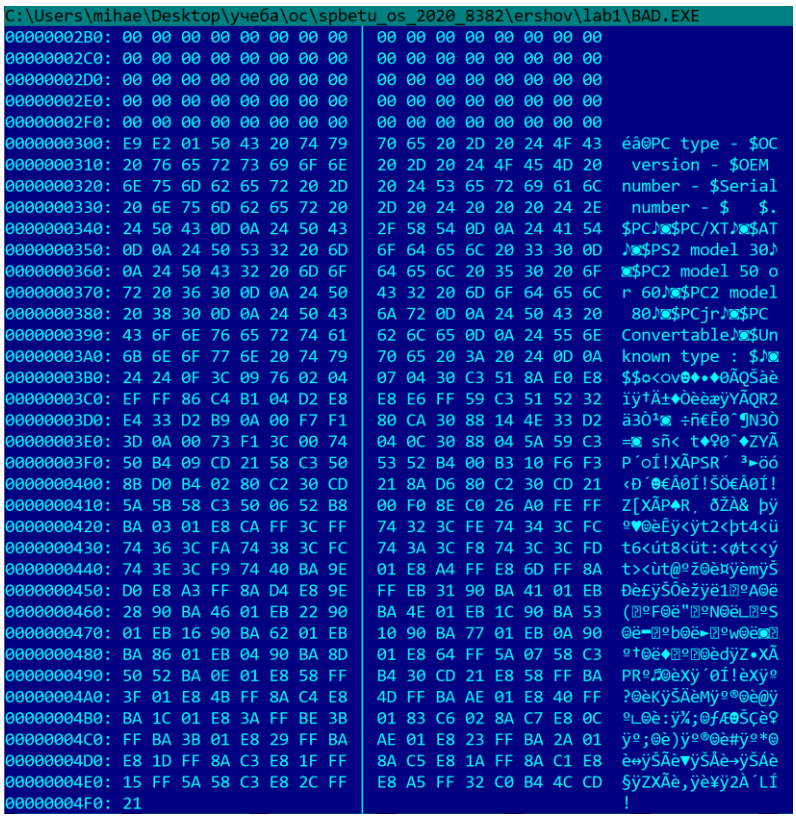

Рисунок 2 – конец «плохого» EXE файла

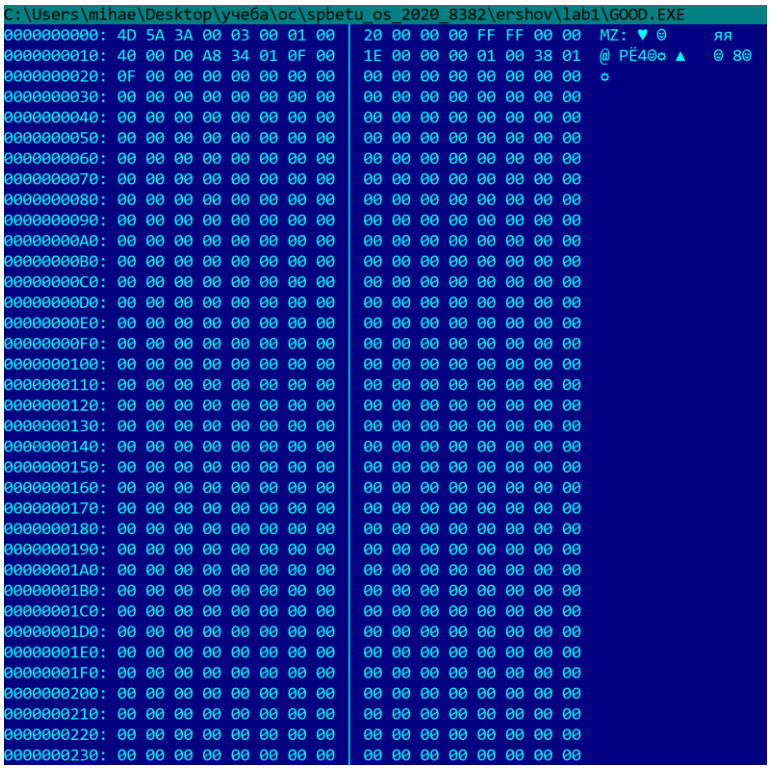

Рисунок 3 – начало «хорошего» EXE файла

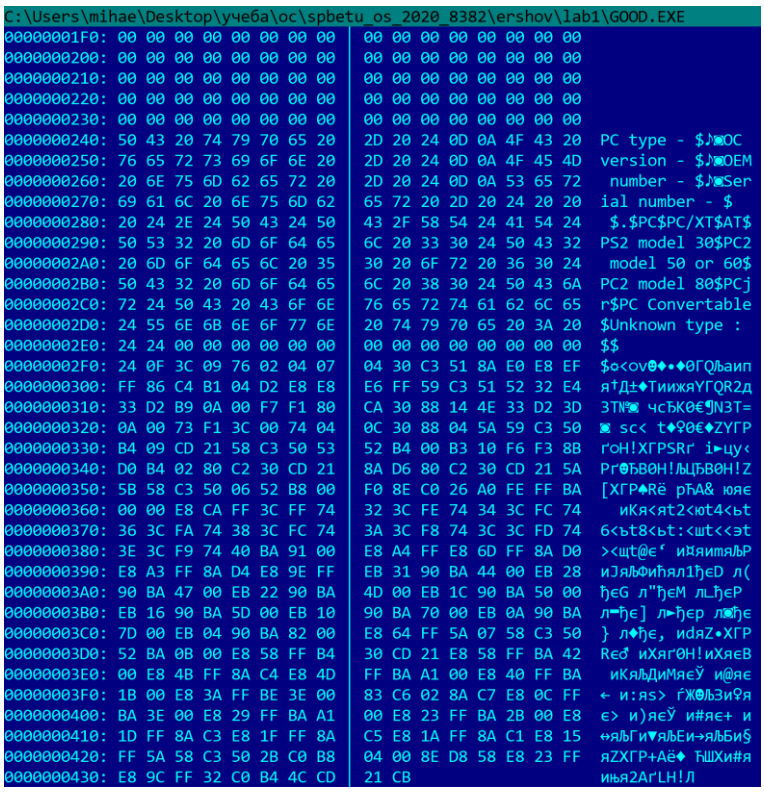

Рисунок 4 – конец «хорошего» EXE файла

Загрузка COM модуля в основную память

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

Код располагается с адреса 100h, до него располагается PSP. Данные, код и PSP расположены в одном сегменте.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP.

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

Все сегментные регистры имеют значение 48DD и указывают на начало PSP. На рис. 5 представлен COM модуль, загруженный в отладчик (TD).

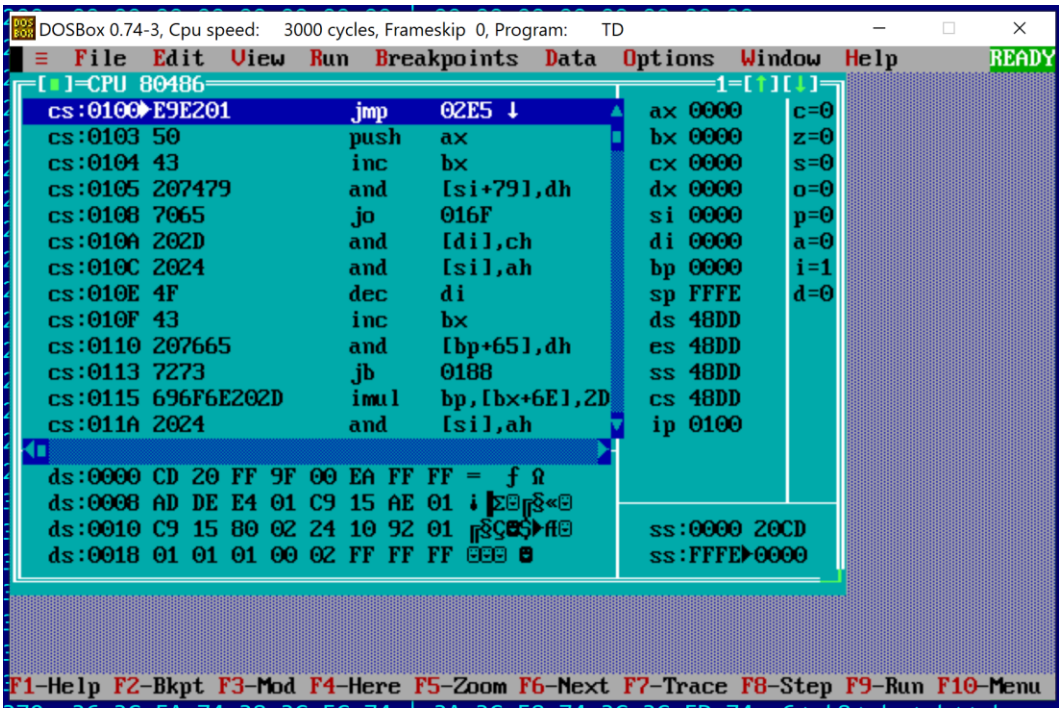

Рисунок 5 – COM модуль открытый в отладчике

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

Регистр сегмента стека (SP, рис. 5) при запуске программы равен FFFEh, стек занимает всю пространство не занятое PSP и кодом программы.

Загрузка "хорошего" EXE модуля в основную память

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

Регистры DS и ES имеют значение 48DD, регистр CS 48FC, а SS 48ED сегмента стека. На рис. 6 представлен EXE модуль, загруженный в отладчик.

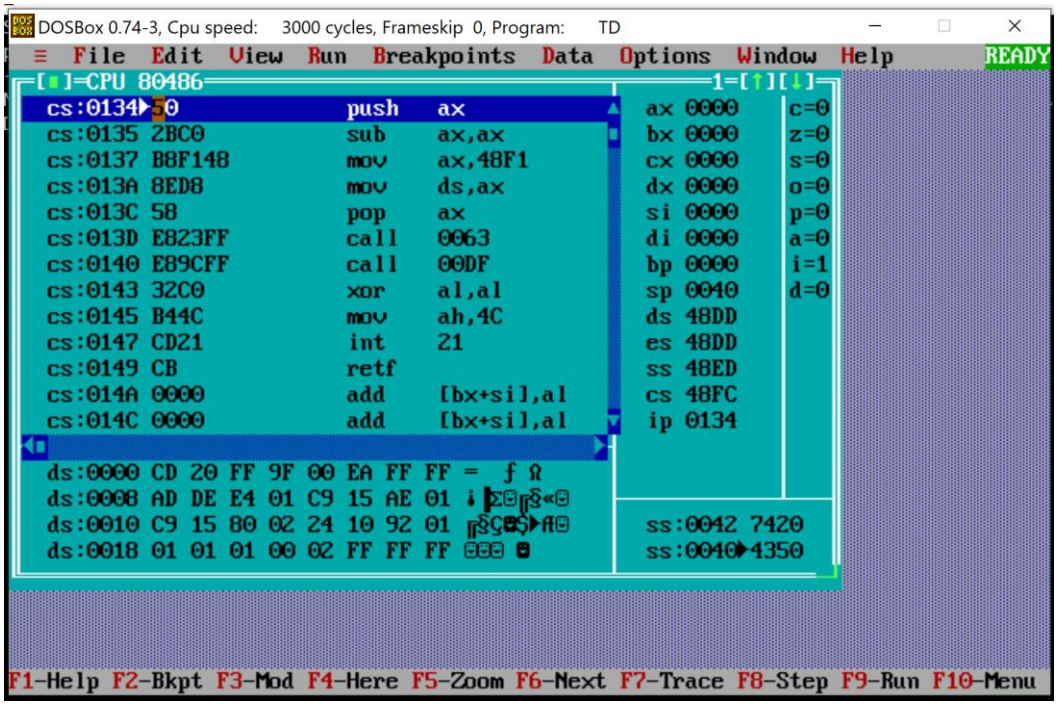

Рисунок 6 – Рисунок 5 – EXE модуль открытый в отладчике

2) На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало PSP.

3) Как определяется стек?

Сегмент стека может выделяться с помощью директивы stack, либо автоматически в соответствии с моделью памяти.

4) Как определяется точка входа?

Точка входа определяется началом сегмента кода, либо с помощью директивы END.

Вывод

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД COM.ASM

```
LR1 SEGMENT
    ASSUME CS:LR1, DS:LR1, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
;DATA

PC_TYPE db "PC type - $"
OC_VERSION db "OC version - $"
OEM_NUM db "OEM number - $"
S_NUM db "Serial number - $"
OEM db "   $"
DOT db ".$"
T_PC db 'PC' , 0DH, 0AH, '$'
T_XT db 'PC/XT' , 0DH, 0AH, '$'
T_AT db 'AT' , 0DH, 0AH, '$'
T_PS2_30 db 'PS2 model 30' , 0DH, 0AH, '$'
T_PS2_50 db 'PC2 model 50 or 60' , 0DH, 0AH, '$'
T_PS2_80 db 'PC2 model 80' , 0DH, 0AH, '$'
T_PCJR db 'PCjr' , 0DH, 0AH, '$'
T_PC_C db 'PC Convertable' , 0DH, 0AH, '$'
T_UNKNOWN db "Unknown type : $"
ENTER_SYMB db 0DH, 0AH, '$'

;PROCEDURES

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

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_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_SYMBOL PROC near
      push AX
      push BX
```

```
        push DX

        mov AH, 0
        mov BL, 16
        div BL
        mov DX, AX
        mov AH, 02h
        add DL, '0'
        int 21h
        mov DL, DH
        add DL, '0'
        int 21h

        pop DX
        pop BX
        pop AX
        ret
PRINT_SYMBOL ENDP

PRINT_PC_TYPE PROC near
    push AX
    push ES
    push DX
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]
    mov dx, offset PC_TYPE
    call PRINT_STRING
    cmp AL, 0FFh
    je pc_t
    cmp AL, 0FEh
    je xt_t
    cmp AL, 0FCh
    je at_t
    cmp AL, 0FAh
    je ps2_30_t
    cmp AL, 0FCh
    je ps2_50_t
    cmp AL, 0F8h
    je ps2_80_t
    cmp AL, 0FDh
    je pcjr_t
```

```
    cmp AL, 0F9h
    je pc_c_t
    mov dx, offset T_UNKNOWN
    call PRINT_STRING
    call BYTE_TO_HEX

    mov dl, al
    call PRINT_SYMBOL
    mov dl, ah
    call PRINT_SYMBOL

    jmp p_out
pc_t:
    mov dx, offset T_PC
    jmp print_end
xt_t:
    mov dx, offset T_XT
    jmp print_end
at_t:
    mov dx, offset T_AT
    jmp print_end
ps2_30_t:
    mov dx, offset T_PS2_30
    jmp print_end
ps2_50_t:
    mov dx, offset T_PS2_50
    jmp print_end
ps2_80_t:
    mov dx, offset T_PS2_80
    jmp print_end
pcjr_t:
    mov dx, offset T_PCJR
    jmp print_end
pc_c_t:
    mov dx, offset T_PC_C



print_end:
    call PRINT_STRING
p_out:
    pop DX
```

```
    pop ES
    pop AX

    ret
PRINT_PC_TYPE ENDP

PRINT_OC_VERSION Proc near
    push ax
    push dx
    mov dx, offset OC_VERSION
    call PRINT_STRING
    mov ah, 30h
    int 21h
    call PRINT_SYMBOL
    mov dx, offset DOT
    call PRINT_STRING
    mov al, ah
    ;add dl, '0'
    call PRINT_SYMBOL
    mov dx, offset ENTER_SYMB
    call PRINT_STRING
    mov dx, offset OEM_NUM
    call PRINT_STRING
    mov si, offset OEM
      add si, 2
      mov al, bh
      call BYTE_TO_DEC
      mov dx, offset OEM
    call PRINT_STRING
    mov dx, offset ENTER_SYMB
    call PRINT_STRING
    mov dx, offset S_NUM
    call PRINT_STRING
    mov al, bl
    call PRINT_SYMBOL
    mov al, ch
    call PRINT_SYMBOL
    mov al, cl
    call PRINT_SYMBOL


    pop dx
```

```
    pop ax

    ret
PRINT_OC_VERSION ENDP

BEGIN:
    call PRINT_PC_TYPE
    call PRINT_OC_VERSION
      xor AL, AL
      mov AH, 4Ch
      int 21h
LR1 ENDS
END START
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД EXE.ASM

```
ASTACK SEGMENT STACK
        DW 20h DUP(?)
ASTACK ENDS

DATA SEGMENT
    PC_TYPE db "PC type - $"
    OC_VERSION db 13, 10, "OC version - $"
    OEM_NUM db 13, 10, "OEM number - $"
    S_NUM db 13, 10, "Serial number - $"
    OEM db "   $"
    DOT db ".$"
    T_PC db "PC$"
    T_XT db "PC/XT$"
    T_AT db "AT$"
    T_PS2_30 db "PS2 model 30$"
    T_PS2_50 db "PC2 model 50 or 60$"
    T_PS2_80 db "PC2 model 80$"
    T_PCJR db "PCjr$"
    T_PC_C db "PC Convertable$"
    T_UNKNOWN db "Unknown type : $"
    ENTER_SYMB db "$"
DATA ENDS

CODE    SEGMENT
                ASSUME  CS:CODE, 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

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_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_SYMBOL PROC near
        push AX
        push BX
        push DX

        mov AH, 0
        mov BL, 16
        div BL
        mov DX, AX
        mov AH, 02h
        add DL, '0'
        int 21h
        mov DL, DH
        add DL, '0'
        int 21h

        pop DX
        pop BX
        pop AX
        ret
PRINT_SYMBOL ENDP

PRINT_PC_TYPE PROC near
    push AX
    push ES
    push DX
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]
    mov dx, offset PC_TYPE
    call PRINT_STRING
    cmp AL, 0FFh
    je pc_t
    cmp AL, 0FEh
    je xt_t
    cmp AL, 0FCh
    je at_t
    cmp AL, 0FAh
    je ps2_30_t
    cmp AL, 0FCh
    je ps2_50_t
    cmp AL, 0F8h
```

```
    je ps2_80_t
    cmp AL, 0FDh
    je pcjr_t
    cmp AL, 0F9h
    je pc_c_t
    mov dx, offset T_UNKNOWN
    call PRINT_STRING
    call BYTE_TO_HEX

    mov dl, al
    call PRINT_SYMBOL
    mov dl, ah
    call PRINT_SYMBOL

    jmp p_out
pc_t:
    mov dx, offset T_PC
    jmp print_end
xt_t:
    mov dx, offset T_XT
    jmp print_end
at_t:
    mov dx, offset T_AT
    jmp print_end
ps2_30_t:
    mov dx, offset T_PS2_30
    jmp print_end
ps2_50_t:
    mov dx, offset T_PS2_50
    jmp print_end
ps2_80_t:
    mov dx, offset T_PS2_80
    jmp print_end
pcjr_t:
    mov dx, offset T_PCJR
    jmp print_end
pc_c_t:
    mov dx, offset T_PC_C


print_end:
    call PRINT_STRING
p_out:
    pop DX
    pop ES
    pop AX

    ret
PRINT_PC_TYPE ENDP

PRINT_OC_VERSION Proc near
    push ax
    push dx
    mov dx, offset OC_VERSION
    call PRINT_STRING
    mov ah, 30h
    int 21h
    call PRINT_SYMBOL
    mov dx, offset DOT
    call PRINT_STRING
    mov al, ah
    call PRINT_SYMBOL
    mov dx, offset ENTER_SYMB
```

```
    call PRINT_STRING
    mov dx, offset OEM_NUM
    call PRINT_STRING
    mov si, offset OEM
        add si, 2
        mov al, bh
        call BYTE_TO_DEC
        mov dx, offset OEM
    call PRINT_STRING
    mov dx, offset ENTER_SYMB
    call PRINT_STRING
    mov dx, offset S_NUM
    call PRINT_STRING
    mov al, bl
    call PRINT_SYMBOL
    mov al, ch
    call PRINT_SYMBOL
    mov al, cl
    call PRINT_SYMBOL

    pop dx
    pop ax

    ret
PRINT_OC_VERSION ENDP

MAIN PROC FAR
    push ax
    sub ax,ax
    mov ax,DATA
    mov ds,ax
    pop ax
    call PRINT_PC_TYPE
    call PRINT_OC_VERSION
        xor al, al
        mov ah, 4Ch
        int 21h
    ret
MAIN ENDP
CODE ENDS
END MAIN
```