

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей

Студент гр. 8382

Янкин Д.О.

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 загрузки в основную память.

Ход работы.

Был написан код программы и получен «хороший» .COM модуль.



```
C:\>lr1com.com
PC
DOS version: 05.00
OEM: 000
User Number: 00000000
```

Рисунок 1. Результат работы «хорошего» .COM модуля

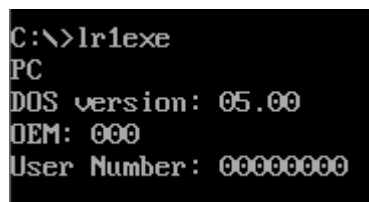
Из исходного кода .COM модуля получен «плохой» .EXE модуль.



```
C:\>lr1com.exe
0/EPC
5 0
0
0/EPC
0/EPC
0/EPC
0 0
0/EPC
```

Рисунок 2. Результат работы «плохого» .EXE модуля

Был написан исходный текст .EXE модуля и получен «хороший» .EXE.



```
C:\>lr1exe
PC
DOS version: 05.00
OEM: 000
User Number: 00000000
```

Рисунок 3. Результат работы «хорошего» .EXE модуля

Контрольные вопросы.

Отличия исходных текстов .COM и .EXE программ:

- 1) Сколько сегментов должна содержать .COM программа?

.COM содержит один сегмент.

2) Сколько сегментов должна содержать .EXE программа?

.EXE программа должна содержать как минимум один сегмент – кода. Обычно она имеет три и более: стек, данные, код.

3) Какие директивы должны обязательно присутствовать в тексте .COM программы?

ASSUME – указывает транслятору, на какие сегменты должны указывать сегментные регистры

ORG – резервирует память в начале программы под PSP и устанавливает соответствующее смещение для кода

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

.COM программа ограничена в количестве сегментов – он ровно один. Размер программы не превышает 64 КБ.

Отличия форматов файлов .COM и .EXE модулей:

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

Модуль состоит из одного сегмента. Код располагается с адреса 0.

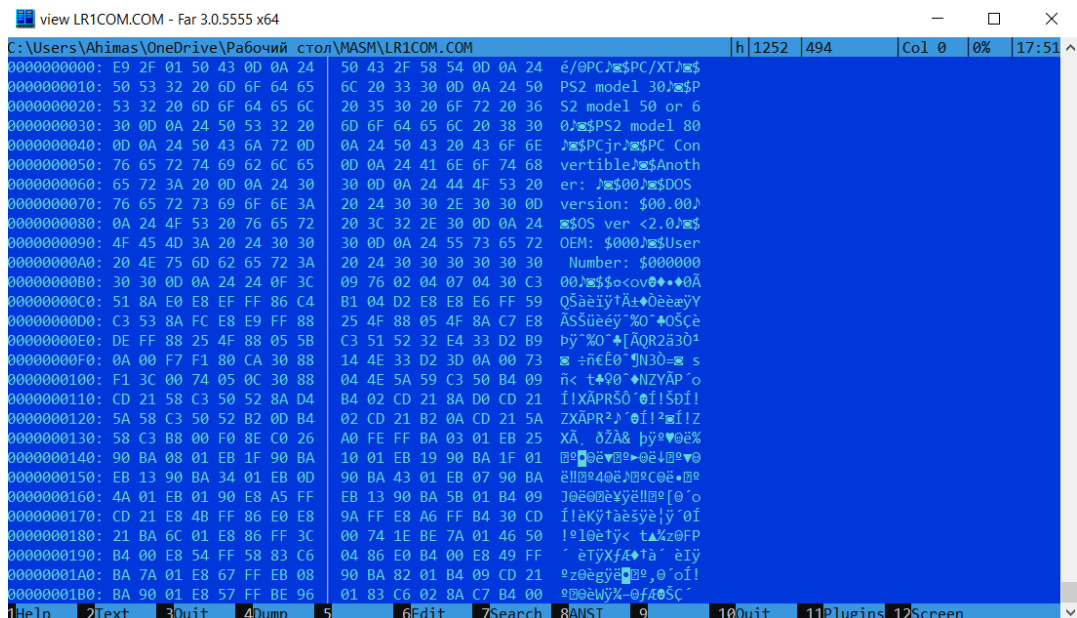


Рисунок 4. .COM модуль в 16-ричном виде

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

В начале модуля находится заголовок и таблица настройки адресов. Код начинается с адреса 300h.

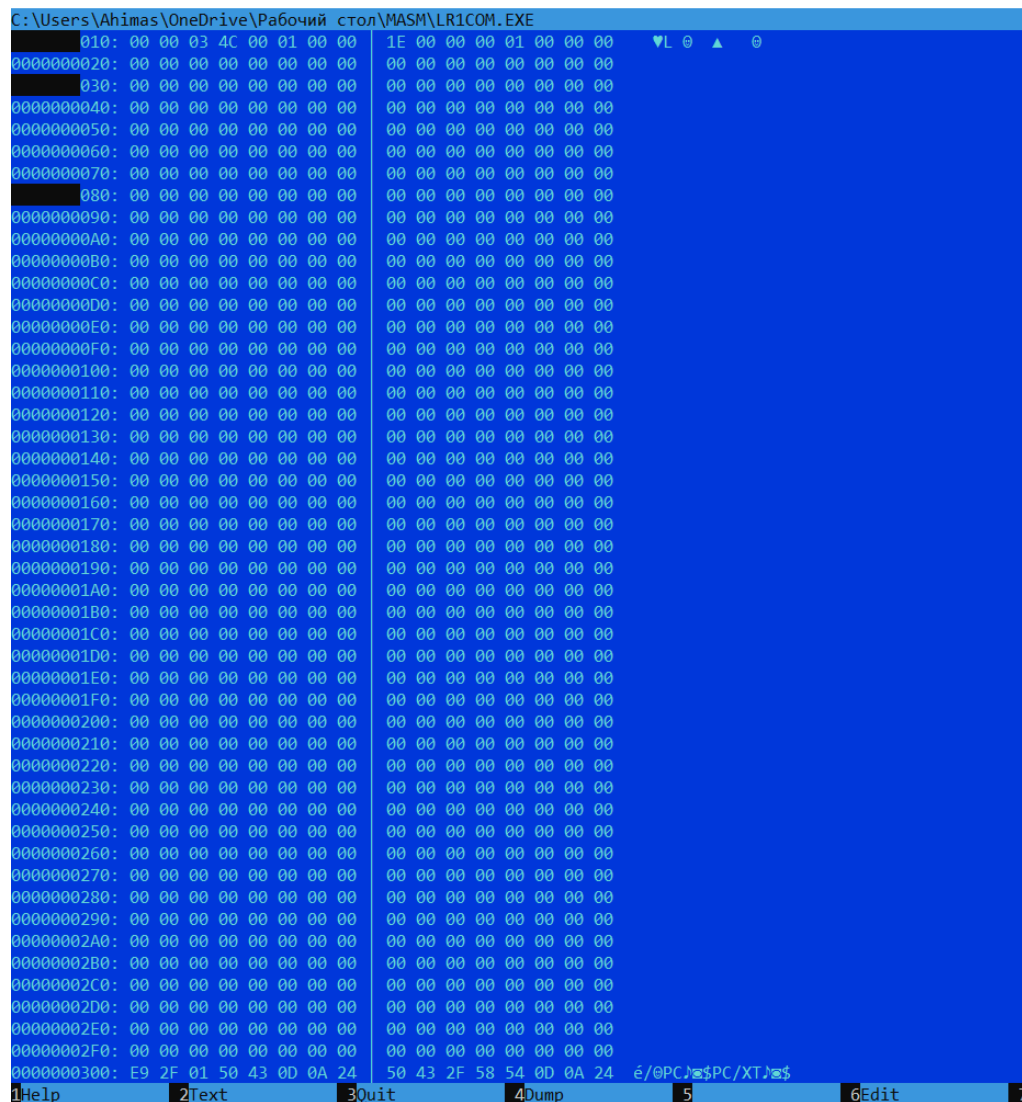


Рисунок 5. «Плохой» .EXE модуль в 16-ричном виде

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

«Хороший» .EXE состоит из заголовка, таблица настройки адресов и трех сегментов: стека, данных, кода. Отличие с количестве сегментов.

```

view LR1EXE.EXE - Far 3.0.5555 x64
C:\Users\Ahimas\OneDrive\Рабочий стол\MASM\LR1EXE.EXE
0000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000400: 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 50 53 32 PC\%$PC/XT\%$PS2
0000000410: 20 6D 6F 64 65 6C 20 33 30 0D 0A 24 50 53 32 20 model 30\%$PS2
0000000420: 6D 6F 64 65 6C 20 35 30 20 6F 72 20 36 30 0D 0A model 50 or 60\%
1Help 2Text 3Quit 4Dump 5

```

Рисунок 6. «Хороший» .EXE модуль в 16-ричном виде

Загрузка .COM модуля в основную память:

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

Перед телом программы добавляется PSP размером 100h байт. Код лежит с адреса 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP, в котором находится команда для выхода из программы, информация о параметрах командной строки, о доступной памяти и прочем.

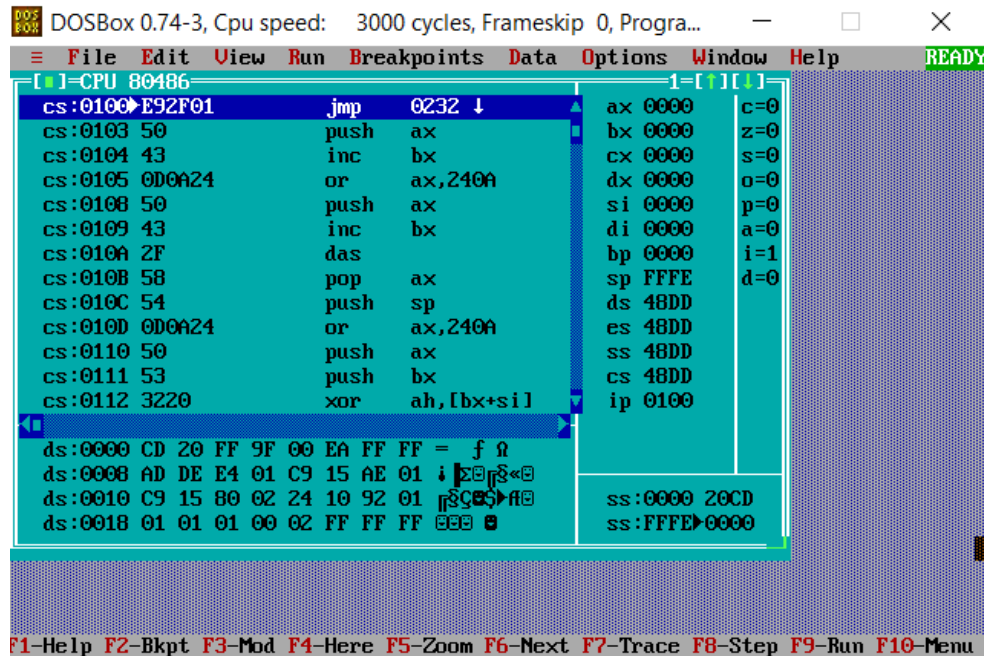


Рисунок 7. Запуск .COM в отладчике

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

Сегментные регистры имеют значение 48DD и указывают на PSP.

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

Стек располагается после кода программы. Вначале программы SS указывает на PSP, а SP равен FFFE, то есть дно стека располагается в последних байтах сегмента программы. Таким образом, можно предположить, что стек может занимать адреса от 48DD:FFFE до 48DD:0000, однако это совпадает со всем сегментом программы, и в случае сильного расширения стека был бы испорчен код программы. Механизм контроля верхней границы стека не выяснен в ходе работы.

Загрузка «хорошего» .EXE модуля в основную память:

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

При загрузке .EXE модуля к нему добавляется PSP. DS и ES указывают на PSP (48DD). SS указывает на стек (48ED). CS указывает на сегмент кода (4919).

- 2) На что указывают регистры DS и ES?

DS и ES указывают на PSP (48DD).

- 3) Как определяется стек?

Размер стека определяется в исходном коде программы, при запуске

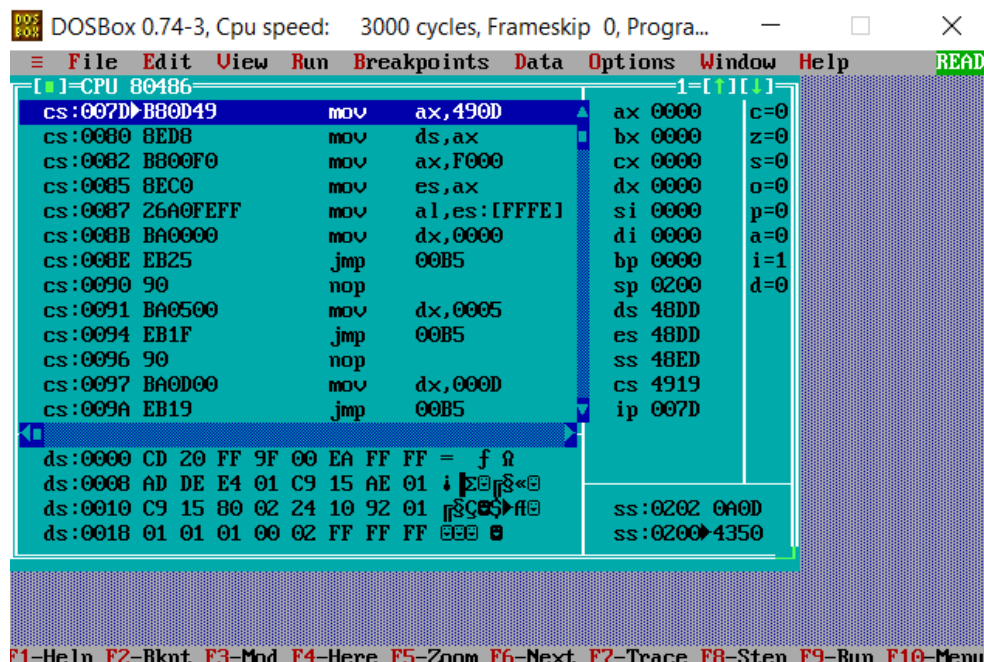


Рисунок 8. Запуск «хорошего» .EXE в отладчике

«хорошего» .EXE SS указывает на начало сегментного регистра, а SP имеет значение, равное размеру стека.

- 4) Как определяется точка входа?

Точка входа определяется директивой END и меткой, с которой нужно начать выполнение программы.

Выводы.

В ходе работы были получены навыки реализации .COM и .EXE программ и изучены их различия. Была написана программа для определения некоторых сведений об операционной системе.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LABCOM.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

TYPE_PC	db 'PC', 13, 10, '\$'
TYPE_PCorXT	db 'PC/XT', 13, 10, '\$'
TYPE_PS2_30	db 'PS2 model 30', 13, 10, '\$'
TYPE_PS2_50or60	db 'PS2 model 50 or 60', 13, 10, '\$'
TYPE_PS2_80	db 'PS2 model 80', 13, 10, '\$'
TYPE_PCjr	db 'PCjr', 13, 10, '\$'
TYPE_PC_Convertible	db 'PC Convertible', 13, 10, '\$'
TYPE_ANOTHER_MESSAGE	db 'Another: ', 13, 10, '\$'
TYPE_ANOTHER	db '00', 13, 10, '\$'
OS_VER_MESSAGE	db 'DOS version: ', '\$'
OS_VER	db '00.00', 13, 10, '\$'
OS_VER_BELOW_2	db 'OS ver <2.0', 13, 10, '\$'
OEM_MESSAGE	db 'OEM: ', '\$'
OEM	db '000', 13, 10, '\$'
USER_NUMBER_MESSAGE	db 'User Number: ', '\$'
USER_NUMBER	db '00000000', 13, 10, '\$'

; ПРОЦЕДУРЫ

;-----

TETR_TO_HEX PROC near

; младшая шестн. цифра AL в шестн. цифру ASCII

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 шестн. числа ASCII 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
        ;xchg AL,AH           ;; а теперь наоборот!
        ret
BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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_1
        or AL,30h
        mov [SI],AL
        dec si
end_1:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP

;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
PRINT_WORD ENDP

;-----
PRINT_ENDL PROC near

```

; Выводит регистр 13, 10

push ax
push dx

mov dl, 13
mov ah, 02h
int 21h

mov dl, 10
int 21h

pop dx
pop ax
ret

PRINT_ENDL ENDP

;-----

; КОД

BEGIN:

mov ax, 0F000h
mov es, ax
mov al, es:[0FFFEh]

PRINT_PC:

mov dx, offset TYPE_PC
jmp PRINT_TYPE

PRINT_PCorXT:

mov dx, offset TYPE_PCorXT
jmp PRINT_TYPE

```

PRINT_PS2_30:
    mov dx, offset TYPE_PS2_30
    jmp PRINT_TYPE

PRINT_PS2_50or60:
    mov dx, offset TYPE_PS2_50or60
    jmp PRINT_TYPE

PRINT_PS2_80:
    mov dx, offset TYPE_PS2_80
    jmp PRINT_TYPE

PRINT_PCjr:
    mov dx, offset TYPE_PCjr
    jmp PRINT_TYPE

PRINT_PC_Convertible:
    mov dx, offset TYPE_PC_Convertible
    jmp PRINT_TYPE

PRINT_TYPE:
    call PRINT_STRING
    jmp DOS_VERSION

PRINT_ANOTHER:
    mov dx, offset TYPE_ANOTHER_MESSAGE
    mov ah, 09h
    int 21h
    call BYTE_TO_HEX
    xchg ah, al
    call PRINT_WORD
    call PRINT_ENDL

```

DOS_VERSION:

```
    mov ah, 30h
    int 21h

    mov dx, offset OS_VER_MESSAGE
    call PRINT_STRING

    cmp al, 0
    je    DOS_VER_LESS_2
    mov si, offset OS_VER
    inc si

    push ax
    mov ah, 0
    call BYTE_TO_DEC
    pop ax

    add si, 4
    xchg ah, al
    mov ah, 0
    call BYTE_TO_DEC

    mov dx, offset OS_VER
    call PRINT_STRING

    jmp PRINT_OEM
```

DOS_VER_LESS_2:

```
    mov dx, offset OS_VER_BELOW_2
    mov ah, 09h
    int 21h
```

PRINT_OEM:

```
mov dx, offset OEM_MESSAGE
call PRINT_STRING
```

```
mov si, offset OEM
add si, 2
mov al, bh
mov ah, 0
call BYTE_TO_DEC
```

```
mov dx, offset OEM
call PRINT_STRING
```

PRINT_USER_NUM:

```
mov dx, offset USER_NUMBER_MESSAGE
call PRINT_STRING
```

```
mov si, offset USER_NUMBER
add si, 7
mov ax, cx
call BYTE_TO_DEC
```

```
dec si
mov al, bl
mov ah, 0
call BYTE_TO_DEC
```

```
mov dx, offset USER_NUMBER
call PRINT_STRING
```

```
xor AL,AL
mov AH,4Ch
```


int 21H

TESTPC ENDS

END START ; конец модуля, START – точка входа

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ LABEXE.ASM

```
AStack    SEGMENT    STACK
           DW 100h DUP(0)
AStack    ENDS
```

DATA SEGMENT

; ДАННЫЕ

```
TYPE_PC          db 'PC', 13, 10, '$'
TYPE_PCorXT      db 'PC/XT', 13, 10, '$'
TYPE_PS2_30      db 'PS2 model 30', 13, 10, '$'
TYPE_PS2_50or60  db 'PS2 model 50 or 60', 13, 10, '$'
TYPE_PS2_80      db 'PS2 model 80', 13, 10, '$'
TYPE_PCjr        db 'PCjr', 13, 10, '$'
TYPE_PC_Convertible db 'PC Convertible', 13, 10, '$'
TYPE_ANOTHER_MESSAGE db 'Another: ', 13, 10, '$'
TYPE_ANOTHER     db '00', 13, 10, '$'

OS_VER_MESSAGE   db 'DOS version: ', '$'
OS_VER           db '00.00', 13, 10, '$'
OS_VER_BELOW_2   db 'OS ver <2.0', 13, 10, '$'
OEM_MESSAGE      db 'OEM: ', '$'
OEM              db '000', 13, 10, '$'
USER_NUMBER_MESSAGE db 'User Number: ', '$'
USER_NUMBER      db '00000000', 13, 10, '$'
DATA ENDS
```

CODE SEGMENT

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

; ПРОЦЕДУРЫ

```

;-----
TETR_TO_HEX PROC near
; младшая шестн. цифра AL в шестн. цифру ASCII
    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 шестн. числа ASCII 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
    ;xchg AL,AH         ;; а теперь наоборот!
    ret
BYTE_TO_HEX ENDP

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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_1
        or AL,30h
        mov [SI],AL
        dec si
end_1:   pop DX
        pop CX

```

```
ret
BYTE_TO_DEC ENDP
```

```
;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP
```

```
;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
```

PRINT_WORD ENDP

;-----

PRINT_ENDL PROC near

; Выводит регистр 13, 10

push ax

push dx

mov dl, 13

mov ah, 02h

int 21h

mov dl, 10

int 21h

pop dx

pop ax

ret

PRINT_ENDL ENDP

;-----

; КОД

MAIN PROC FAR

mov ax, DATA

mov ds, ax

mov ax, 0F000h

mov es, ax

mov al, es:[0FFFEh]

PRINT_PC:

```
    mov dx, offset TYPE_PC
    jmp PRINT_TYPE
```

PRINT_PCorXT:

```
    mov dx, offset TYPE_PCorXT
    jmp PRINT_TYPE
```

PRINT_PS2_30:

```
    mov dx, offset TYPE_PS2_30
    jmp PRINT_TYPE
```

PRINT_PS2_50or60:

```
    mov dx, offset TYPE_PS2_50or60
    jmp PRINT_TYPE
```

PRINT_PS2_80:

```
    mov dx, offset TYPE_PS2_80
    jmp PRINT_TYPE
```

PRINT_PCjr:

```
    mov dx, offset TYPE_PCjr
    jmp PRINT_TYPE
```

PRINT_PC_Convertible:

```
    mov dx, offset TYPE_PC_Convertible
    jmp PRINT_TYPE
```

PRINT_TYPE:

```
    call PRINT_STRING
    jmp DOS_VERSION
```

PRINT_ANOTHER:

```
    mov dx, offset TYPE_ANOTHER_MESSAGE
    mov ah, 09h
```

```
int 21h
call BYTE_TO_HEX
xchg ah, al
call PRINT_WORD
call PRINT_ENDL
```

DOS_VERSION:

```
mov ah, 30h
int 21h

mov dx, offset OS_VER_MESSAGE
call PRINT_STRING

cmp al, 0
je DOS_VER_LESS_2
mov si, offset OS_VER
inc si

push ax
mov ah, 0
call BYTE_TO_DEC
pop ax

add si, 4
xchg ah, al
mov ah, 0
call BYTE_TO_DEC

mov dx, offset OS_VER
call PRINT_STRING

jmp PRINT_OEM
```


DOS_VER_LESS_2:

```
    mov dx, offset OS_VER_BELOW_2
    mov ah, 09h
    int 21h
```

PRINT_OEM:

```
    mov dx, offset OEM_MESSAGE
    call PRINT_STRING
```

```
    mov si, offset OEM
    add si, 2
    mov al, bh
    mov ah, 0
    call BYTE_TO_DEC
```

```
    mov dx, offset OEM
    call PRINT_STRING
```

PRINT_USER_NUM:

```
    mov dx, offset USER_NUMBER_MESSAGE
    call PRINT_STRING
```

```
    mov si, offset USER_NUMBER
    add si, 7
    mov ax, cx
    call BYTE_TO_DEC
```

```
    ;dec si
    mov al, bl
    mov ah, 0
    call BYTE_TO_DEC
```

```
mov dx, offset USER_NUMBER  
call PRINT_STRING
```

```
xor AL,AL  
mov AH,4Ch  
int 21H
```

```
MAIN ENDP
```

```
CODE ENDS
```

```
END MAIN ; конец модуля, START – точка входа
```