

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

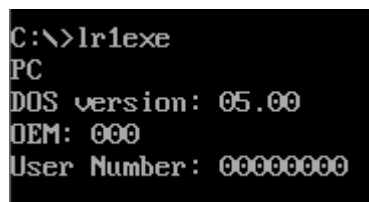
Из исходного кода .COM модуля получен «плохой» .EXE модуль.



```
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?

Нельзя использовать директивы упрощенной сегментации и создавать более одного сегмента.

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.

```
view LR1COM.COM - Far 3.0.5555 x64
C:\Users\Ahimas\OneDrive\Рабочий стол\MASM\LR1COM.COM |h|1252|494|Col 0|0%|17:51
00000000: E9 2F 01 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 é/0PC./$PC/XT./$
000000010: 50 53 32 20 6D 6F 64 65 6C 20 33 30 0D 0A 24 50 PS2 model 30./$P
000000020: 53 32 20 6D 6F 64 65 6C 20 35 30 20 6F 72 20 36 S2 model 50 or 6
000000030: 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 38 30 0./$PS2 model 80
000000040: 0D 0A 24 50 43 6A 72 0D 0A 24 50 43 20 43 6F 6E ./$PCjr./$PC Con
000000050: 76 65 72 74 69 62 6C 65 0D 0A 24 41 6E 6F 74 68 vertible./$Anoth
000000060: 65 72 3A 20 0D 0A 24 30 30 0D 0A 24 44 4F 53 20 er: ./$00./$DOS
000000070: 76 65 72 73 69 6F 6E 3A 20 24 30 30 2E 30 30 0D version: $00.00
000000080: 0A 24 4F 53 20 76 65 72 20 3C 32 2E 30 0D 0A 24 $OS ver <2.0./$
000000090: 4F 45 4D 3A 20 24 30 30 30 0D 0A 24 55 73 65 72 OEM: $000./$User
0000000A0: 20 4E 75 6D 62 65 72 3A 20 24 30 30 30 30 30 30 Number: $000000
0000000B0: 30 30 0D 0A 24 24 0F 3C 09 76 02 04 07 04 30 C3 00./$e<ov0+♦0Ä
0000000C0: 51 8A E0 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 QSaëiy†A±+0ëëyY
0000000D0: C3 53 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 E8 ASSüëëy' %0' *0SÇë
0000000E0: DE FF 88 25 4F 88 05 5B C3 51 52 32 E4 33 D2 B9 bÿ' %0' *†[ÄQR2ä30¹
0000000F0: 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 73 -ñçÉ0' ¶N3D- s
000000100: F1 3C 00 74 05 0C 30 88 04 4E 5A 59 C3 50 B4 09 ñ< t*90' ♦NZVÄP' o
000000110: CD 21 58 C3 50 52 8A D4 B4 02 CD 21 8A D0 CD 21 I!XÄPRS0' 0!l$Df!
000000120: 5A 58 C3 50 52 B2 0D B4 02 CD 21 B2 0A CD 21 5A ZXÄPR²' 0!l' *m!lZ
000000130: 58 C3 B8 00 F0 BE C0 26 A0 FE FF BA 03 01 EB 25 XÄ. öZÄ8. bÿ' %0ë%
000000140: 90 BA 08 01 EB 1F 90 BA 10 01 EB 19 90 BA 1F 01 B0B0ëW0-0ë100W0
000000150: EB 13 90 BA 34 01 EB 0D 90 BA 43 01 EB 07 90 BA ëll0*40ë200C0ë•00
000000160: 4A 01 EB 01 90 E8 A5 FF EB 13 90 BA 58 01 B4 09 0ë00ëëyëll00[0'o
000000170: CD 21 E8 48 FF 86 E0 E8 9A FF E8 A6 FF B4 30 CD ílekÿtäëÿëÿ'y'0í
000000180: 21 BA 6C 01 E8 86 FF 3C 00 74 1E BE 7A 01 46 50 !0ëÿÿ< tÄ%z0FP
000000190: B4 00 E8 54 FF 58 83 C6 04 86 E0 B4 00 E8 49 FF ' ëTÿXfÄ♦tä' ëTÿ
0000001A0: BA 7A 01 E8 67 FF EB 08 90 BA 82 01 B4 09 CD 21 ëz0ëgyë00,0'oi!
0000001B0: BA 90 01 E8 57 FF BE 96 01 83 C6 02 8A C7 B4 00 00ëëWÿ%-0fÄ0SC'
```

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.

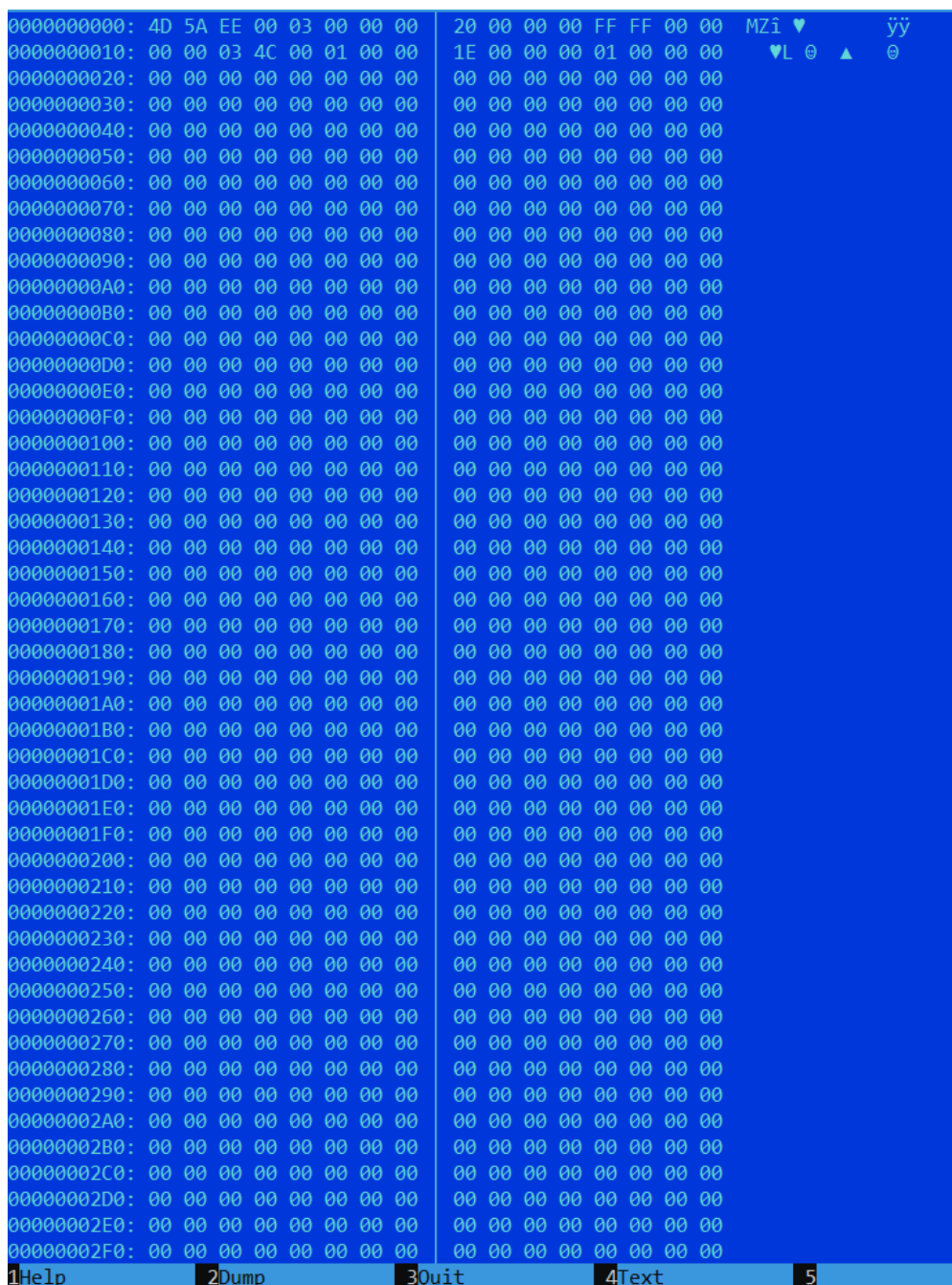


Рисунок 5. «Плохой» .EXE модуль в 16-ричном виде

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

«Хороший» .EXE состоит из заголовка, таблицы настройки адресов и трех сегментов: стека, данных, кода. В заголовке находится сигнатура файла, начальные значения SS, SP, CS, IP; размер файла, количество элементов в таблице адресов и другая информация. Таблица настройки содержит информацию, необходимую для обращения к дальним сегментам программы. Отличие в количестве сегментов. По адресу 300h, где в «плохом» .EXE начинался сегмент кода в «хорошего» .EXE лежит сегмент стека размером 100h

```
view LR1EXE.EXE - Far 3.0.5555 x64
C:\Users\Ahimas\OneDrive\Рабочий стол\MASM\LR1EXE.EXE
00000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000003F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000400: 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 50 53 32 PC\PS2
00000000410: 20 6D 6F 64 65 6C 20 33 30 0D 0A 24 50 53 32 20 model 30 PS2
00000000420: 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.

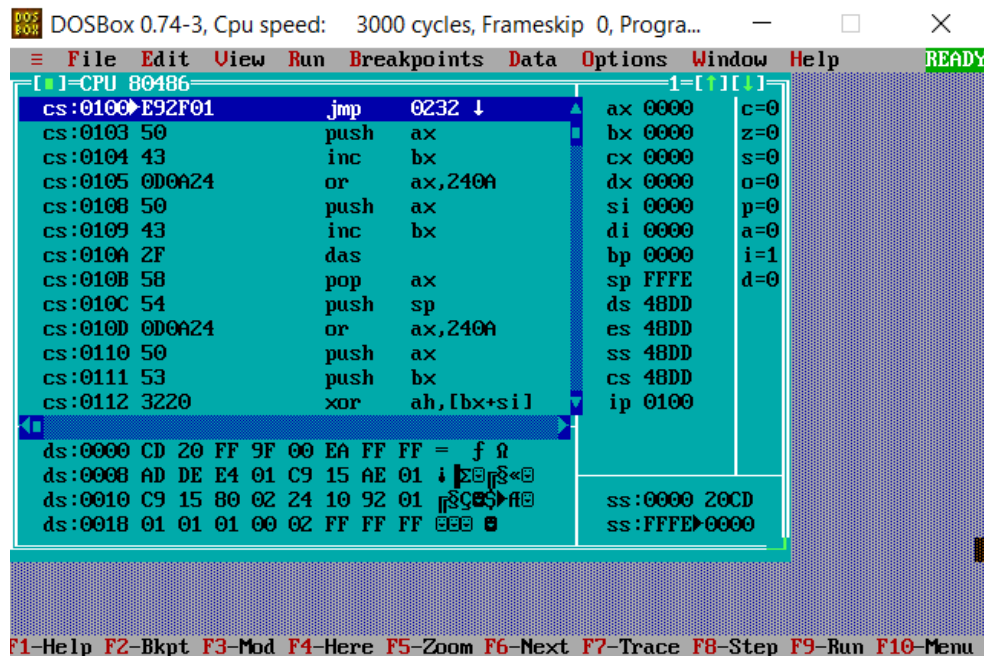


Рисунок 7. Запуск .COM в отладчике

- 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.

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

Сегментные регистры имеют значение 4BDD и указывают на PSP.

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

Стек располагается после кода программы. В начале программы SS указывает на PSP, а SP равен FFFE, то есть дно стека располагается в последних байтах сегмента программы. В начале программы в стеке лежит значение 0000h. Таким образом, можно предположить, что стек может занимать адреса от 4BDD:FFFF до 4BDD: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).

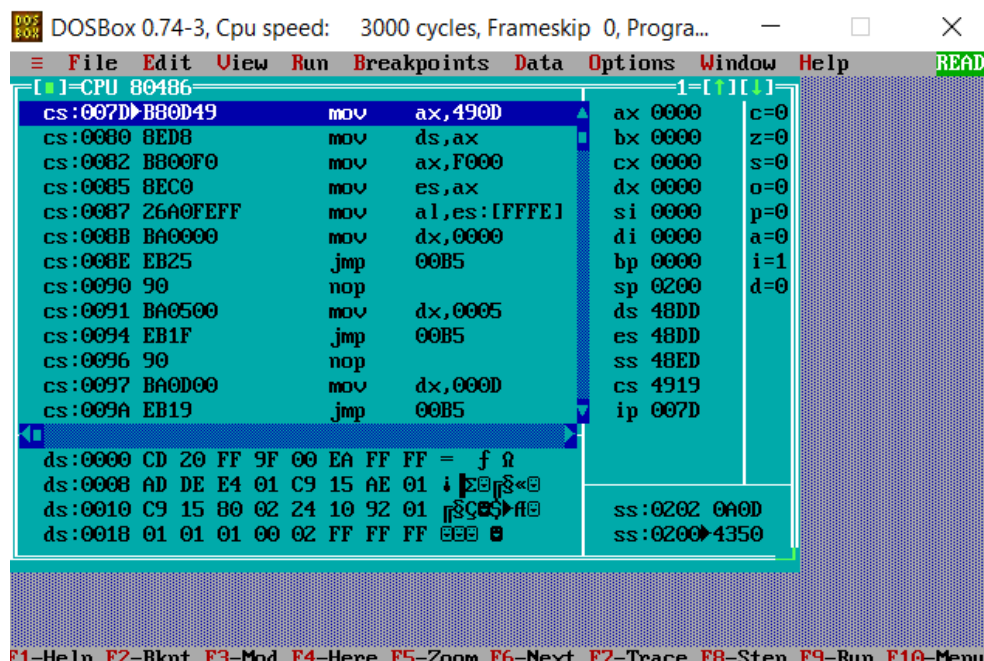


Рисунок 8. Запуск «хорошего» .EXE в отладчике

- 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 «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
```