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. 7383 _____ Ханова Ю.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

Постановка задачи.

Исследование различий в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Ход работы.

В ходе выполнения данной работы был создан набор функций и структур данных, описанных в табл. 1-2.

Таблица 1 – Описание функций.

Название функции	**Назначение**
System	Печатает тип ОС
Ver_OS	Печатает версию ОС
Write_oem	Печатает серийный номер OEM
Num_of_serial	Печатает серийный номер пользователя
Write	Вызывает функцию печати строки
TETR_TO_HEX	Вспомогательная функция для работы функции BYTE_TO_HEX
BYTE_TO_HEX	Переводит число AL в коды символов 16-ой с/с, записывая получившееся в BL и BH
WRD_TO_HEX	Переводит число AX в строку в 16-ой с/с, записывая получившееся в di, начиная с младшей цифры
BYTE_TO_DEC	Переводит байт из AL в десятичную с/с и записывает получившееся число по адресу SI, начиная с младшей цифры

Таблица 2 - Описание структур данных.

Название	**Тип**	**Назначение**
OS	db	Тип ОС
OS_VERS	db	Версия ОС
OS_OEM	db	Серийный номер OEM
SER_NUM	db	Серийный номер пользователя
PC	db	PC
PCXT	db	PC/XT
_AT	db	AT
PS2_30	db	PS2 модель 30
PS2_80	db	PS2 модель 80
PCjr	db	PCjr
PC_Cnv	db	PC Convertible

Были созданы файлы типов .COM и .EXE («плохой» и «хороший»)

Программа определяет и выводит на экран следующие значения в заданном порядке: тип ОС, версия ОС, серийный номер пользователя, серийный номер OEM. Результаты работы программы представлены на рис. 1-3.

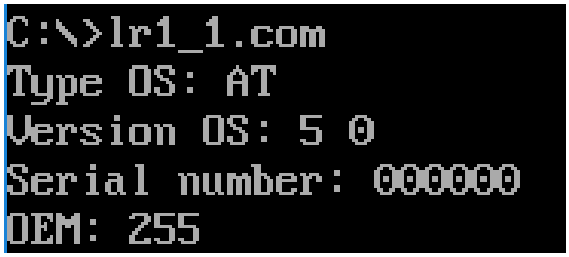

Рисунок 1 – Результат выполнения программы lr1_1.com

Рисунок 2 – Результат выполнения программы lr1_1.exe

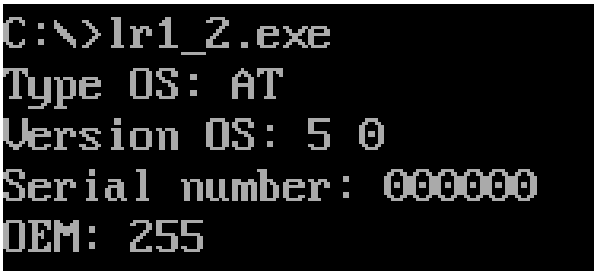

Рисунок 3 – Результат выполнения программы lr1_2.exe

Выводы.

В процессе выполнения данной лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Код программы lr1_1.asm и lr1_2.asm представлены в приложении А.

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

1 сегмент.

2. EXE-программа?

Минимум 1 сегмент.

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

В тексте COM-программы обязательно должна быть директива ORG 100h, которая сдвигает адресацию в программе на 256 байт для расположения PSP.

Так же, должна присутствовать директива ASSUME, ставящая в соответствие начало программы сегментам кода и данных (при отсутствии директивы ASSUME, программа не скомпилируется из-за невозможности обнаружения начала сегмента кода).

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

Нет, в COM-программе нельзя использовать команды вида mov register, segment и команды, содержащие дальнюю (far) адресацию, т.к. в этих командах используется таблица настройки в которой содержатся адреса сегментов. Такая таблица есть только в EXE-файлах, поэтому COM-программа не может использовать сегментную адресацию.

Отличия форматов файлов COM и EXE модулей

1. Какова структура файла COM? С какого адреса располагается код? HEX-представление .COM файла показано на рис. 4.

Рисунок 4 – HEX представление .COM файла

COM-файл содержит только код и данные. В файле код располагается с нулевого адреса.

2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0? HEX-представление «плохого» .EXE файла показано на рис. 5-6.

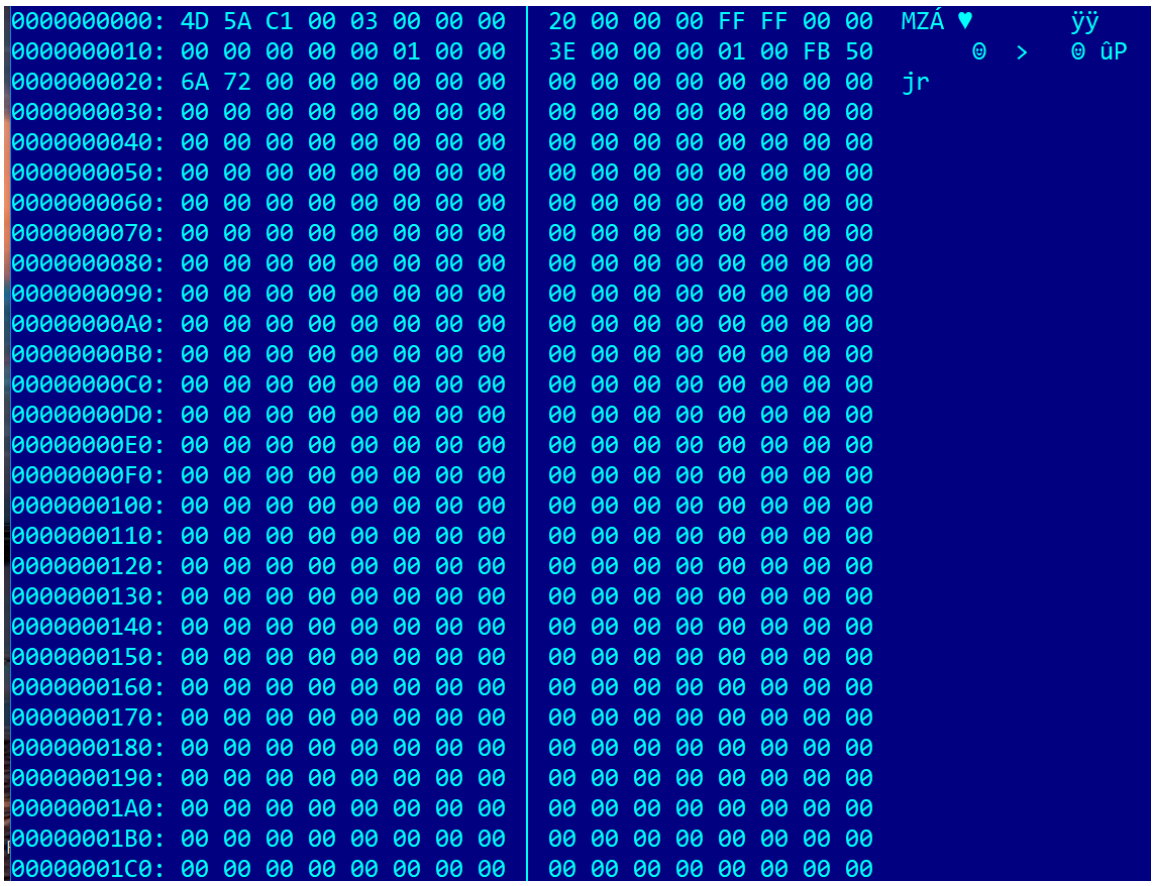

Рисунок 5 – HEX-представление «плохого» .EXE (1)

Рисунок 6 – HEX-представление «плохого» .EXE (2)

В «плохом» EXE код и данные не разделены по сегментам, а перемешаны (на скриншоте перед данными видно метку перехода E9 AE 01). Код располагается с адреса 300h, т.к. заголовок занимает 200h байт (байты 8 и 9 указывают, сколько параграфов занимает заголовок) и команда ORG 100h «сдвигает» код на дополнительные 100h. С нулевого адреса располагается заголовок. В первых двух байтах можно увидеть символы MZ, означающие, что формат файла – 16-битный и его следует запускать в соответствии со структурой EXE-файлов. За заголовком следует таблица настройки. Если их убрать, то файл будет загружаться в память как COM-файл.

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

HEX-представление «хорошего» .EXE файла показано на рис 7-8.

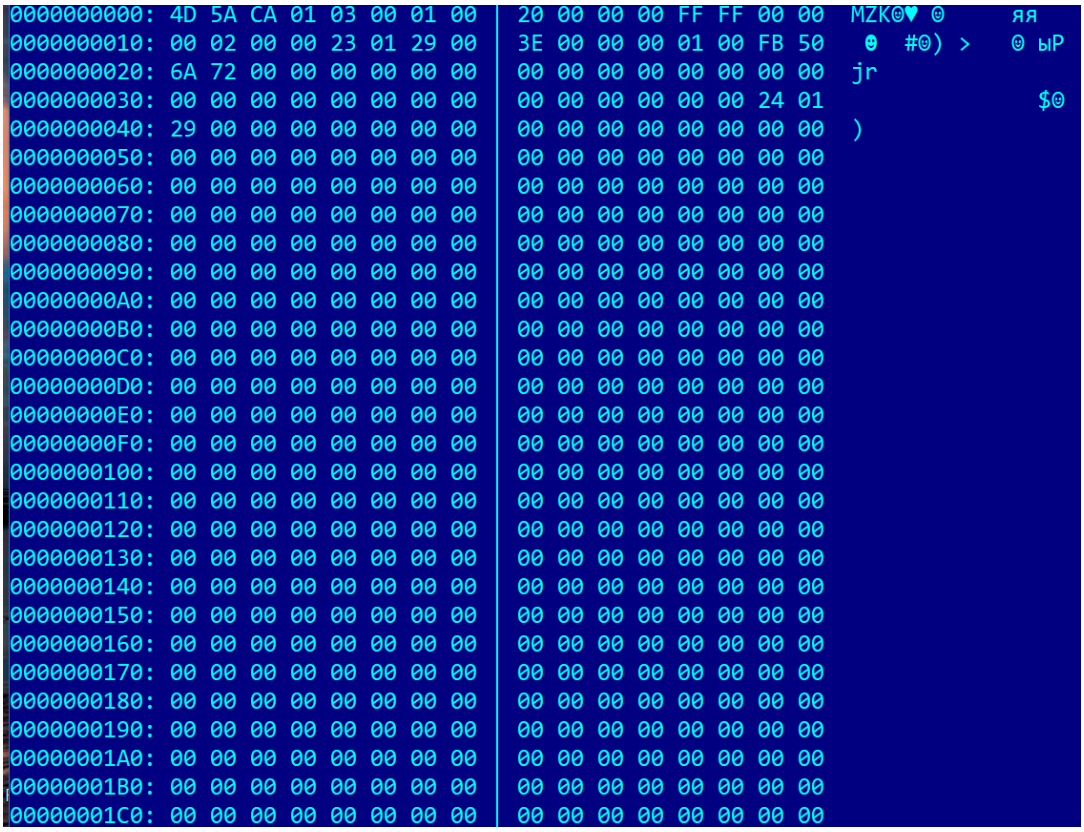

Рисунок 7 – HEX-представление «хорошего» .EXE (1)

Рисунок 8 – HEX-представление «хорошего» .EXE (2)

В отличие от «плохого» EXE, в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается с 400h, т.к. дополнительно выделено под стек 200 байт (100 слов). Для «хорошего» EXE в директиве org 100h нет необходимости, т.к. загрузчик автоматически расположит программу после PSP.

Результат загрузки COM модуля в основную память представлен на рис. 9.

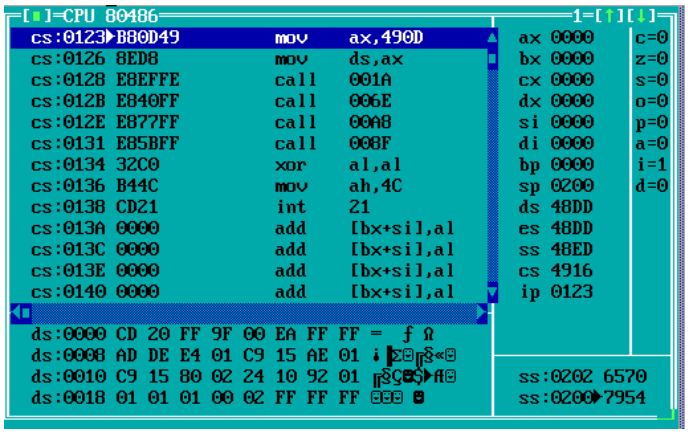

Рисунок 9 – Результат загрузки .COM в основную память

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

Формат загрузки модуля COM:

1. Выделение сегмента памяти для модуля
2. Установка всех сегментных регистров на начало выделенного сегмента памяти
3. Построение в первых 100h байтах памяти PSP
4. Загрузка содержимого COM-файла и присваивание регистру IP значения 100h.
5. Регистр SP устанавливается в конец сегмента

Код начинается с адреса, содержащимся в CS, в нашем случае это 48DD.

2. Что располагается с адреса 0?

С нулевого адреса располагается PSP.

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

Все сегментные регистры (CS, DS, ES, SS) в данном случае равны 48DD и указывают на начало PSP.

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

Стек занимает весь сегмент COM-программы, его начало находится в конце сегмента. SS указывает на начало сегмента, а SP=FFFEh – на его конец. Стек может дойти до кода/данных программы при достаточном количестве элементов.

Адреса расположены в диапазоне 0000h-FFFEh. Стек растет от больших адресов к меньшим.

Результат загрузки «хорошего» EXE модуля в основную память представлен на рис. 10.

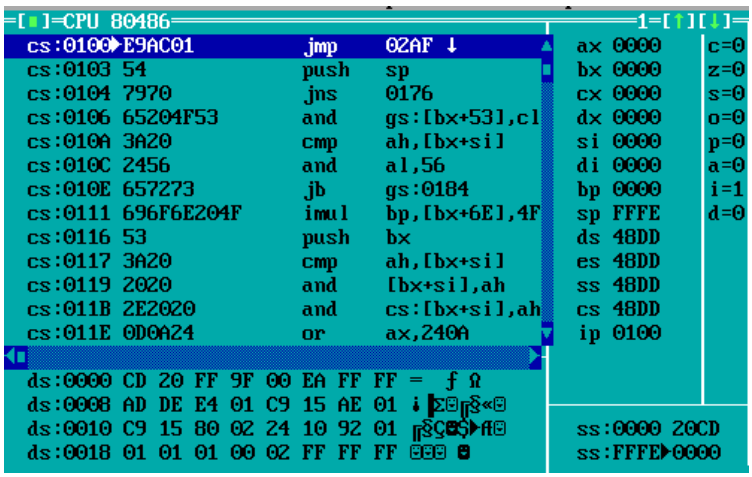

Рисунок 10– Результат загрузки «хорошего» .EXE в основную память

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

SS=48DD – начало сегмента стека, CS=48DD – начало сегмента команд.

2. На что указывают регистры DS и ES?

На начало PSP.

3. Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в SP – его вершины.

4. Как определяется точка входа?

Точка входа в программу определяется с помощью директивы END. После этой директивы указывается метка, куда переходит программа при запуске.

ПРИЛОЖЕНИЕ А

lr1_1.asm

```
TESTPC SEGMENT
             ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
             ORG    100H
START: JMP BEGIN
;ДАННЫЕ
OS db 'Type OS: $'
OS_VERS db 'Version OS:   .  ',0DH,0AH,'$'
OS_OEM db 'OEM:    ',0DH,0AH,'$'
SER_NUM db 'Serial number: ','$'
STRING db '    $'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PCXT db 'PC/XT',0DH,0AH,'$'
_AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCjr db 'PCjr',0DH,0AH,'$'
PC_Cnv db 'PC Convertible',0DH,0AH,'$'


Write PROC near
       mov AH,09h
       int 21h
       ret
Write 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

System PROC near
       mov ax,0F000h
       mov es,ax
       mov ax,es:0FFFEh
       ret
System ENDP

;определение типа ОС
```

```
Write_system PROC near
	mov dx, OFFSET OS
	call Write
	call System

	cmp al,0FFh
	je PC_mtk
	cmp al,0FEh
	je PCXT_mtk
	cmp al,0FBh
	je PCXT_mtk
	cmp al,0FCh
	je AT_mtk
	cmp al,0FAh
	je PS2_30_mtk
	cmp al,0F8h
	je PS2_80_mtk
	cmp al,0FDh
	je PCjr_mtk
	cmp al,0F9h
	je PC_Cnv_mtk

	PC_mtk:
		mov dx, OFFSET PC
		jmp to_write
	PCXT_mtk:
		mov dx, OFFSET PCXT
		jmp to_write
	AT_mtk:
		mov dx, OFFSET _AT
		jmp to_write
	PS2_30_mtk:
		mov dx, OFFSET PS2_30
		jmp to_write
	PS2_80_mtk:
		mov dx, OFFSET PS2_80
		jmp to_write
	PCjr_mtk:
		mov dx, OFFSET PCjr
		jmp to_write
	PC_Cnv_mtk:
		mov dx, OFFSET PC_Cnv


	to_write:
	call Write
	ret
```

```
Write_system ENDP

;определение версии системы
Ver_OS PROC near
	;получение данных
	mov ax,0
	mov ah,30h
	int 21h
	;запись в строку OS_VERS номер основной версии ОС
	mov si,offset OS_VERS
	add si,12
	push ax
	call BYTE_TO_DEC

	pop ax
	mov al,ah
	add si,3
	call BYTE_TO_DEC
	;запись версии ОС в консоль
	mov dx,offset OS_VERS
	call Write
	ret
Ver_OS ENDP

;определение oem
Write_oem PROC near
	mov ax,0
	mov ah,30h
	int 21h

	mov si,offset OS_OEM
	add si,7
	mov al,bh
	call BYTE_TO_DEC

	mov dx,offset OS_OEM
	call Write
	ret
Write_oem ENDP

Num_of_serial PROC near
	;запись серийного номера пользователя
	mov dx,offset SER_NUM
	call Write
	mov  al,bl
	call BYTE_TO_HEX
	mov bx,ax
```

```
        mov dl,bl
        mov ah,02h
        int 21h
        mov dl,bh
        int 21h
        mov di,offset STRING
        add di,3
        mov ax,cx
        call WRD_TO_HEX
        mov dx,offset STRING
        call Write

        mov dx,offset ENDSTR
        call Write

        ret
Num_of_serial 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
        pop CX
        ret
BYTE_TO_HEX ENDP

; перевод в 16с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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

    ; перевод в 10с/с, SI - адрес поля младшей цифры
    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

    BEGIN:
        call Write_system
        call Ver_OS
        call Num_of_serial
        call Write_oem
        xor AL,AL
        mov AH,4Ch
        int 21H
    TESTPC ENDS
END START
```

lr1_2.asm

```
STACK SEGMENT STACK
       DW 0100H DUP(?)
STACK ENDS

DATA SEGMENT
; ДАННЫЕ
OS db 'Type OS: $'
OS_VERS db 'Version OS:    .  ',0DH,0AH,'$'
OS_OEM db 'OEM:     ',0DH,0AH,'$'
SER_NUM db 'Serial number: ','$'
STRING db '    $'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PCXT db 'PC/XT',0DH,0AH,'$'
_AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCjr db 'PCjr',0DH,0AH,'$'
PC_Cnv db 'PC Convertible',0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK
Write PROC near
       mov AH,09h
       int 21h
       ret
Write 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

System PROC near
       mov ax,0F000h
       mov es,ax
       mov ax,es:0FFFEh
       ret
System ENDP

Write_system PROC near
       mov dx, OFFSET OS
       call Write
       call System

       cmp al,0FFh
       je PC_mtk
       cmp al,0FEh
       je PCXT_mtk
       cmp al,0FBh
       je PCXT_mtk
       cmp al,0FCh
       je AT_mtk
       cmp al,0FAh
```

```
        je PS2_30_mtk
        cmp al,0F8h
        je PS2_80_mtk
        cmp al,0FDh
        je PCjr_mtk
        cmp al,0F9h
        je PC_Cnv_mtk

        PC_mtk:
                mov dx, OFFSET PC
                jmp to_write
        PCXT_mtk:
                mov dx, OFFSET PCXT
                jmp to_write
        AT_mtk:
                mov dx, OFFSET _AT
                jmp to_write
        PS2_30_mtk:
                mov dx, OFFSET PS2_30
                jmp to_write
        PS2_80_mtk:
                mov dx, OFFSET PS2_80
                jmp to_write
        PCjr_mtk:
                mov dx, OFFSET PCjr
                jmp to_write
        PC_Cnv_mtk:
                mov dx, OFFSET PC_Cnv


        to_write:
        call Write
        ret
Write_system ENDP

Ver_OS PROC near
        mov ax,0
        mov ah,30h
        int 21h

        mov si,offset OS_VERS
        add si,12
        push ax
        call BYTE_TO_DEC

        pop ax
        mov al,ah
        add si,3
        call BYTE_TO_DEC

        mov dx,offset OS_VERS
        call Write
        ret
Ver_OS ENDP

Write_oem PROC near
        mov ax,0
        mov ah,30h
        int 21h

        mov si,offset OS_OEM
```

```
       add si,7
       mov al,bh
       call BYTE_TO_DEC

       mov dx,offset OS_OEM
       call Write
       ret
Write_oem ENDP

Num_of_serial PROC near

       mov dx,offset SER_NUM
       call Write
       mov  al,bl
       call BYTE_TO_HEX
       mov bx,ax
       mov dl,bl
       mov ah,02h
       int 21h
       mov dl,bh
       int 21h
       mov di,offset STRING
       add di,3
       mov ax,cx
       call WRD_TO_HEX
       mov dx,offset STRING
       call Write

       mov dx,offset ENDSTR
       call Write

       ret
Num_of_serial 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
       pop CX
       ret
BYTE_TO_HEX ENDP

; перевод в 16с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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

; перевод в 10с/с, SI - адрес поля младшей цифры
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

BEGIN:
       mov ax,DATA
       mov ds,ax
       call Write_system
       call Ver_OS
       call Num_of_serial
       call Write_oem
       xor AL,AL
       mov AH,4Ch
       int 21H
CODE ENDS
 END BEGIN
```