**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. 6383 |  | Михайлов Ю.А. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2018

**Постановка задачи.**

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

**Описание функций и структур данных**

|  |  |
| --- | --- |
| **Название процедуры** | **Описание процедуры** |
| OPR\_TIP\_OS | Печатает тип ОС |
| OPR\_VERSIYU\_OS | Печатает версию ОС, серийный номер OEM и серийный номер пользователя |
| PECHAT | 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 |

bad.exe – «плохой» EXE, good.exe – «хороший» EXE.

**Последовательность действий, выполняемых утилитой**

1. Определение и вывод на экран типа ОС;
2. Определение и вывод на экран версии ОС;
3. Определение и вывод на экран серийного номера OEM;
4. Определение и вывод на экран серийного номера пользователя;

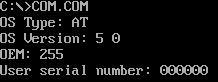
Результат работы программы представлен на рис.1.

Рисунок 1 – Результат выполнения программы good.exe.

**Вывод.**

По ходу выполнения данной лабораторной работы были исследованы различия структур исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

**Ответы на контрольные вопросы.**

**1.Отличия исходных текстов COM и EXE программ**

**1.1 Сколько сегментов должна содержать COM-программа?**

COM-программа всегда содержит 1 сегмент.

**1.2 EXE-программа?**

EXE-программа содержит как минимум 1 сегмент.

**1.3 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, т.к. в первых 256 байтах программы находится PSP. Также обязательна должна быть директива ASSUME, иначе если ее не будет, при компиляции будет выдавать ошибку: «Missing or unreachable CS», т.к. сегментный регистр не указывает на текущий кодовый сегмент или группу, содержащую текущий кодовый сегмент.

**1.4 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.

**2. Отличия форматов файлов COM и EXE модулей**

**2.1 Какова структура файла COM? С какого адреса располагается код?**

HEX-представление COM-файла(см. Рисунок 2):

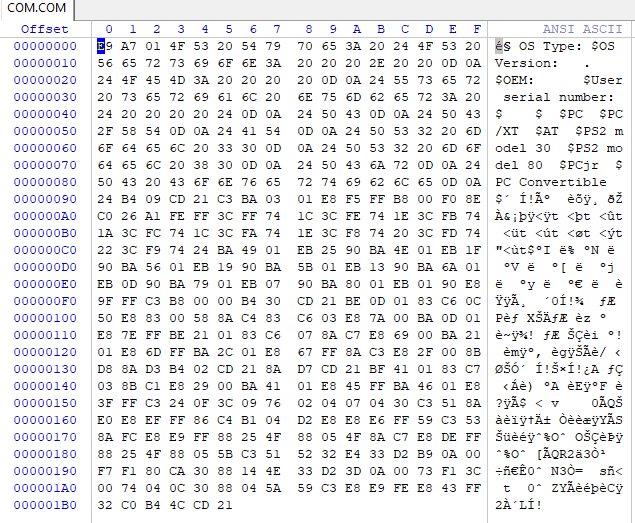
****

Рисунок 2 – HEX-представление COM-файла.

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

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

HEX-представление «плохого» EXE(см. Рисунки 3-4):

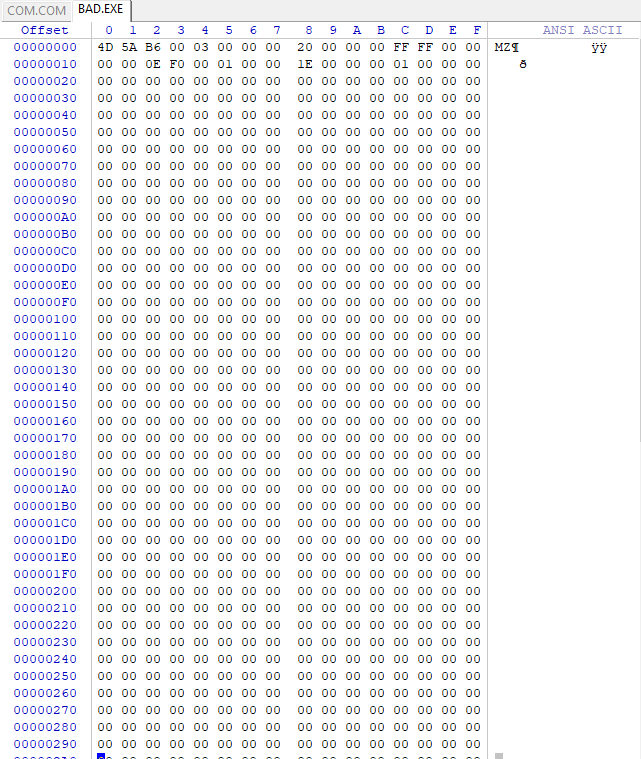
****

Рисунок 3 – первая часть HEX-представления «плохого» EXE.

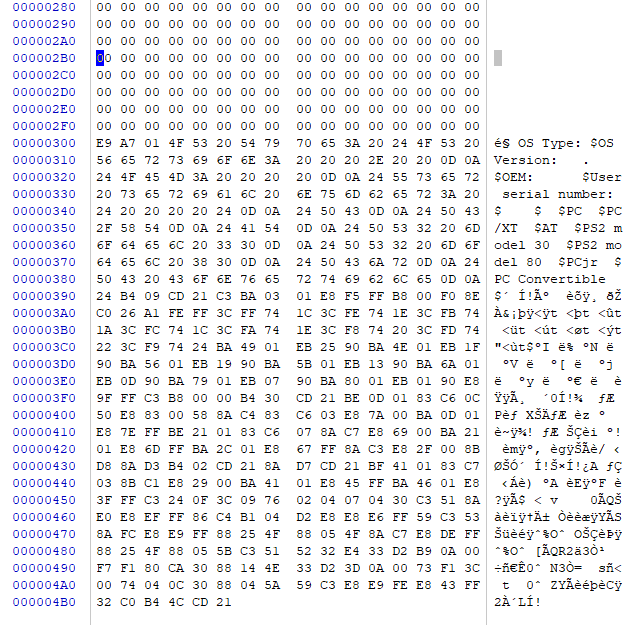
****

Рисунок 4 – вторая часть HEX-представления «плохого» EXE.

В «плохом» EXE код и данные не разделены по сегментам, а перемешаны (на скриншоте перед данными видно метку перехода E9 A7 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.

**2.3 Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?**

HEX-представление «хорошего» EXE(см. Рисунки 5-6):

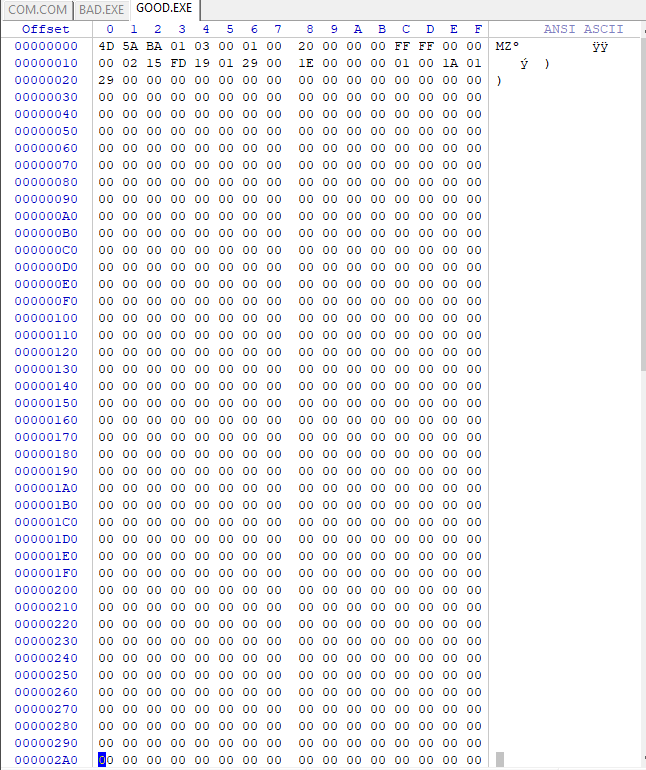
****

Рисунок 5 – первая часть HEX-представления «хорошего» EXE.

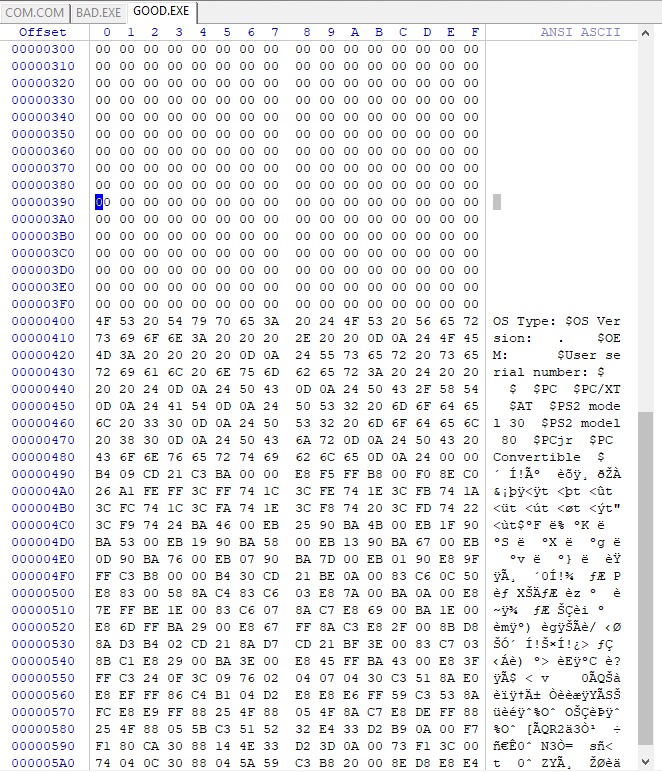
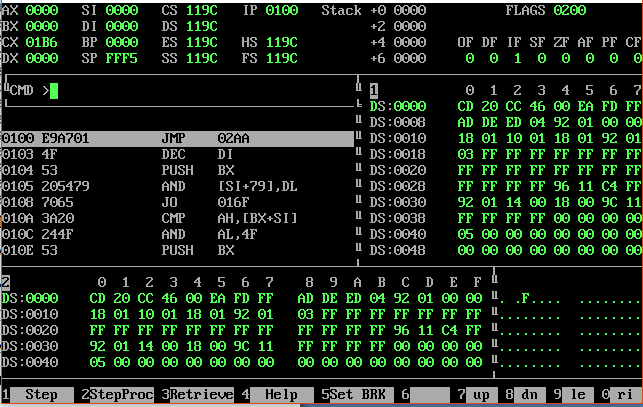
****

Рисунок 6 – вторая часть HEX-представления «хорошего» EXE.

В отличие от «плохого» EXE, в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается с 400h байта(т.к я выделил 200 байт под стек строчкой DW 0100h DUP(?), если бы я использовал ORG 100H, то код программы начался бы с 300h).

Для «хорошего» EXE в директиве org 100h нет необходимости, т.к. загрузчик автоматически расположит программу после PSP.

**3.Загрузка COM модуля в основную память**

****

**3.1 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

**3.2 Что располагается с адреса 0?**

С нулевого адреса располагается PSP.

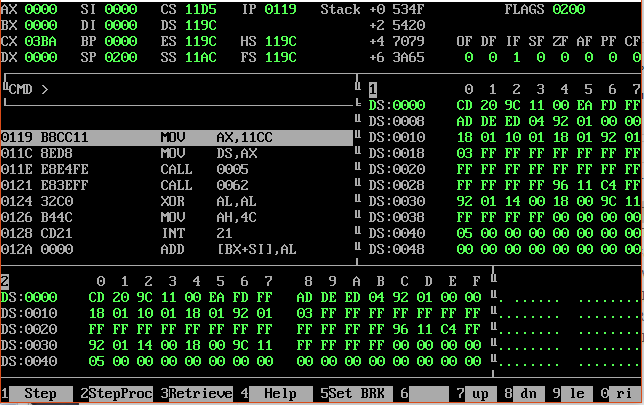
**3.3 Какие значения имеют сегментные регистры? На какие области памяти они указывают?**

Все сегментные регистры (CS, DS, ES, SS) в данном случае равны 119C и указывают на начало PSP.

**3.4 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=FFF5h – на его конец. Элементы стека располагаются в области памяти, отведенной под стек, начиная со дна стека (т.е. с его макс. адреса) по последовательно уменьшающимся адресам. Стек может дойти до кода/данных программы при достаточном количестве элементов.

**4.Загрузка «хорошего» EXE модуля в основную память**



**4.1 Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?**

SS=11AC – начало сегмента стека, CS=11D5 – начало сегмента команд.

**4.2 На что указывают регистры DS и ES?**

DS и ES имеют значение 119C и указывают на начало PSP.

**4.3 Как определяется стек?**

Стек определяется с помощью директивы .stack, после которой задаётся размер стека.

**4.4 Как определяется точка входа?**

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

**КОД ПРОГРАММЫ GOOD.ASM**

; Шаблон текста программы для модуля типа .COM

STACK SEGMENT STACK

DW 0100h DUP(?)

STACK ENDS

DATA SEGMENT

; ДАННЫЕ

OS db 'OS Type: $'

OS\_VERS db 'OS Version: . ',0DH,0AH,'$'

OS\_OEM db 'OEM: ',0DH,0AH,'$' ; additional 3 bytes for digits

USER\_NUMBER db 'User serial number: ','$'

PUT 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

; ПРОЦЕДУРЫ

;---------------------------------------

; Вызывает прерывание, печатающее строку.

WRT\_MSG PROC near

mov AH,09h

int 21h

ret

WRT\_MSG ENDP

;---------------------------------------

; Печатает тип ОС

GET\_TYPE\_OS PROC near

mov dx, OFFSET OS

call WRT\_MSG

mov ax,0F000h

mov es,ax

mov ax,es:0FFFEh

; Определяем тип ОС

cmp al,0FFh

je PC\_MARK

cmp al,0FEh

je PCXT\_MARK

cmp al,0FBh

je PCXT\_MARK

cmp al,0FCh

je AT\_MARK

cmp al,0FAh

je PS2\_30\_MARK

cmp al,0F8h

je PS2\_80\_MARK

cmp al,0FDh

je PCjr\_MARK

cmp al,0F9h

je PC\_Cnv\_MARK

PC\_MARK:

mov dx, OFFSET PC

jmp end1

PCXT\_MARK:

mov dx, OFFSET PCXT

jmp end1

AT\_MARK:

mov dx, OFFSET \_AT

jmp end1

PS2\_30\_MARK:

mov dx, OFFSET PS2\_30

jmp end1

PS2\_80\_MARK:

mov dx, OFFSET PS2\_80

jmp end1

PCjr\_MARK:

mov dx, OFFSET PCjr

jmp end1

PC\_Cnv\_MARK:

mov dx, OFFSET PC\_Cnv

jmp end1

end1:

call WRT\_MSG

ret

GET\_TYPE\_OS ENDP

;---------------------------------------

; Печатает версию системы

GET\_VERS\_OS PROC near

; Получаем данные

mov ax,0

mov ah,30h

int 21h

; Пишем в строку OS\_VERS номер основной версии ОС

mov si,offset OS\_VERS

add si,12

push ax

call BYTE\_TO\_DEC

; Пишем модификацию ОС

pop ax

mov al,ah

add si,3

call BYTE\_TO\_DEC

; Пишем версию ОС в консоль

mov dx,offset OS\_VERS

call WRT\_MSG

; Пишем OEM

mov si,offset OS\_OEM

add si,7

mov al,bh

call BYTE\_TO\_DEC

mov dx,offset OS\_OEM

call WRT\_MSG

; Пишем серийный номер пользователя

mov dx,offset USER\_NUMBER

call WRT\_MSG

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 PUT

add di,3

mov ax,cx

call WRD\_TO\_HEX

mov dx,offset PUT

call WRT\_MSG

mov dx,offset ENDSTR

call WRT\_MSG

ret

GET\_VERS\_OS 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

;---------------------------------------

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

;---------------------------------------

; 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 GET\_TYPE\_OS

call GET\_VERS\_OS

xor AL,AL

mov AH,4Ch

int 21H

CODE ENDS

END BEGIN

**КОД ПРОГРАММЫ BAD.ASM**

; Шаблон текста программы для модуля типа .COM

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

OS db 'OS Type: $'

OS\_VERS db 'OS Version: . ',0DH,0AH,'$'

OS\_OEM db 'OEM: ',0DH,0AH,'$' ; additional 3 bytes for digits

USER\_NUMBER db 'User serial number: ','$'

PUT 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,'$'

; ПРОЦЕДУРЫ

;---------------------------------------

; Вызывает прерывание, печатающее строку.

WRT\_MSG PROC near

mov AH,09h

int 21h

ret

WRT\_MSG ENDP

;---------------------------------------

; Печатает тип ОС

GET\_TYPE\_OS PROC near

mov dx, OFFSET OS

call WRT\_MSG

mov ax,0F000h

mov es,ax

mov ax,es:0FFFEh

; Определяем тип ОС

cmp al,0FFh

je PC\_MARK

cmp al,0FEh

je PCXT\_MARK

cmp al,0FBh

je PCXT\_MARK

cmp al,0FCh

je AT\_MARK

cmp al,0FAh

je PS2\_30\_MARK

cmp al,0F8h

je PS2\_80\_MARK

cmp al,0FDh

je PCjr\_MARK

cmp al,0F9h

je PC\_Cnv\_MARK

PC\_MARK:

mov dx, OFFSET PC

jmp end1

PCXT\_MARK:

mov dx, OFFSET PCXT

jmp end1

AT\_MARK:

mov dx, OFFSET \_AT

jmp end1

PS2\_30\_MARK:

mov dx, OFFSET PS2\_30

jmp end1

PS2\_80\_MARK:

mov dx, OFFSET PS2\_80

jmp end1

PCjr\_MARK:

mov dx, OFFSET PCjr

jmp end1

PC\_Cnv\_MARK:

mov dx, OFFSET PC\_Cnv

jmp end1

end1:

call WRT\_MSG

ret

GET\_TYPE\_OS ENDP

;---------------------------------------

; Печатает версию системы

GET\_VERS\_OS PROC near

; Получаем данные

mov ax,0

mov ah,30h

int 21h

; Пишем в строку OS\_VERS номер основной версии ОС

mov si,offset OS\_VERS

add si,12

push ax

call BYTE\_TO\_DEC

; Пишем модификацию ОС

pop ax

mov al,ah

add si,3

call BYTE\_TO\_DEC

; Пишем версию ОС в консоль

mov dx,offset OS\_VERS

call WRT\_MSG

; Пишем OEM

mov si,offset OS\_OEM

add si,7

mov al,bh

call BYTE\_TO\_DEC

mov dx,offset OS\_OEM

call WRT\_MSG

; Пишем серийный номер пользователя

mov dx,offset USER\_NUMBER

call WRT\_MSG

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 PUT

add di,3

mov ax,cx

call WRD\_TO\_HEX

mov dx,offset PUT

call WRT\_MSG

mov dx,offset ENDSTR

call WRT\_MSG

ret

GET\_VERS\_OS 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

;---------------------------------------

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

;---------------------------------------

; 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 GET\_TYPE\_OS

call GET\_VERS\_OS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START