МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

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

Студентка гр. 8382 _____ Кузина А.М.

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 из загрузки в основную память.

Ход выполнения работы.

Для определения типа IBM PC и версии системы были написаны .COM и .EXE модули, код которых приведен в приложениях A, B. В таблице 1 указаны соответствия кода и типа PC, который определяется последним байтом ROM BIOS.

Таблица 1 — соответствие кода и типа PC.

PC	FF
PC/XT	FE, FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 ил 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

Версия системы определяется значением регистров AL, AH, BH, BL:CX после выполнения функции 30h прерывания 21h, соответственно номер основной версии, номер модификации, серийный номер OEM и серийный номер пользователя. На рисунках 1 — 3 представлены результаты выполнения .COM модуля, «плохого» .EXE модуля, полученного из исходного кода .COM модуля и «хорошего» .EXE модуля.

Рисунок 1 — результат выполнения .COM модуля.

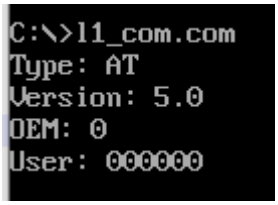
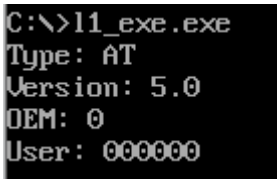


Рисунок 2 — результат выполнения «плохого» .EXE модуля.

Рисунок 3 — результат выполнения «хорошего» .EXE модуля



Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?
   COM-программа должна содержать всего один сегмент.
2) Сколько сегментов должна содержать EXE-программа?
   EXE-программа может содержать разное количество сегментов, в зависимости от модели памяти, но не менее одного.
3) Какие директивы обязательно должны быть к тексте COM-программы?
   Директивы ORG100h и ASSUME. ORG100h выделяет первые 100h байт от начального адреса под PSP, определяя с какого адреса машинный код должен быть помещен в память. ASSUME задает сегменты таким образом, чтобы сегменты данных и кода указывали на один сегмент.
4) Все ли форматы команд можно использовать с COM-программе?
   В COM-программе нельзя использовать команды, работающие с сегментами т. к. в COM-программе сегментные регистры определяются не при трансляции, а во время запуска программы.

Отличия форматов файлов COM и EXE модулей.

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

   COM-файл содержит всего один сегмент, и его размер не превышает 64Кб.

   Код программы располагается с адреса 0h, что показано на рисунке 4, но при запуске коду будет предшествовать 100h байт памяти для PSP.

   Рисунок 4 — COM-файл в шестнадцатеричном виде.

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

   «Плохой» EXE-файл состоит из одного сегмента, в котором располагаются данные и код, а также не определен сегмент стека, что приводит к неправильной работе файла. На рисунках 5, 6 показано, что код располагается с адреса 300h. С адреса 0h в «плохом» EXE-файле располагается заголовок, а также таблица настроек адресов. MZ в таблице означает, что данный файл является исполняемым.

Рисунок 5 — начало «плохого» EXE-файла в шестнадцатеричном виде.

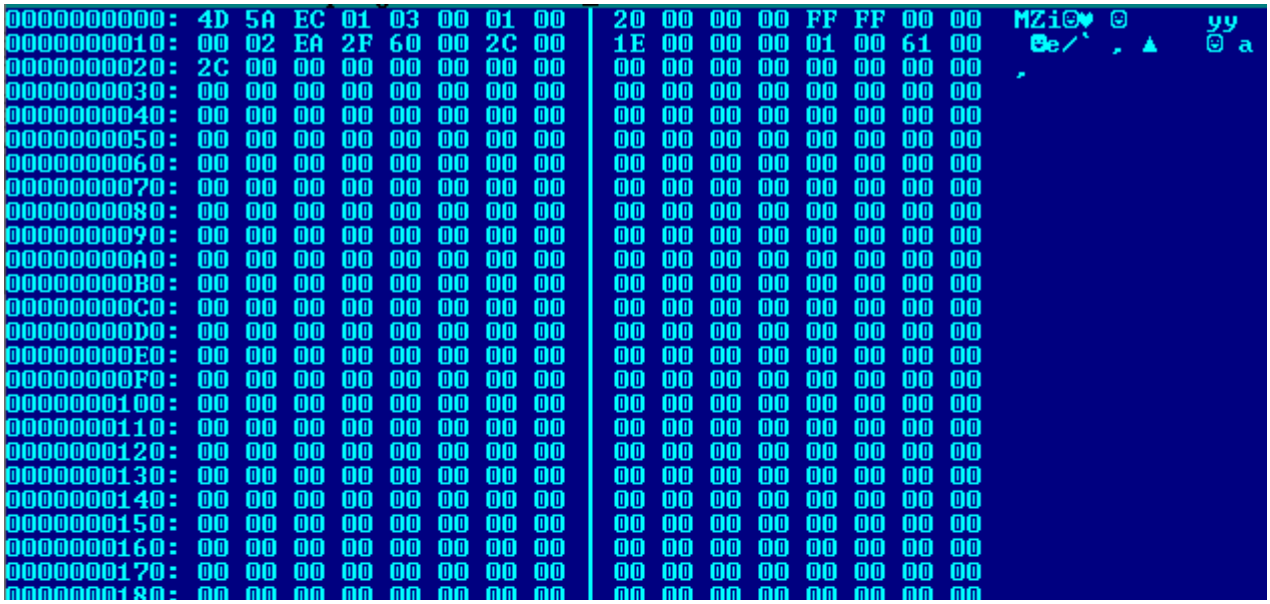


Рисунок 6 — конец «плохого»EXE-файла в шестнадцатеричном виде.

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

   У «хорошего» EXE-файла присутствуют сегменты кода, данных и стека, в то время как у «плохого» только один общий сегмент. В «плохом» файле код всегда начинается с адреса 300h, в «хорошем» файле вначале также идет заголовок и таблица настроек, но т. к. в нем определен сегмент стека, то адресация начинается с 200h + размер стека, 400h в данном случае. На рисунке 7 представлено расположение «хорошего» EXE-файла в памяти.

Рисунок 7 — «хороший» EXE-файл в шестнадцатеричном виде.

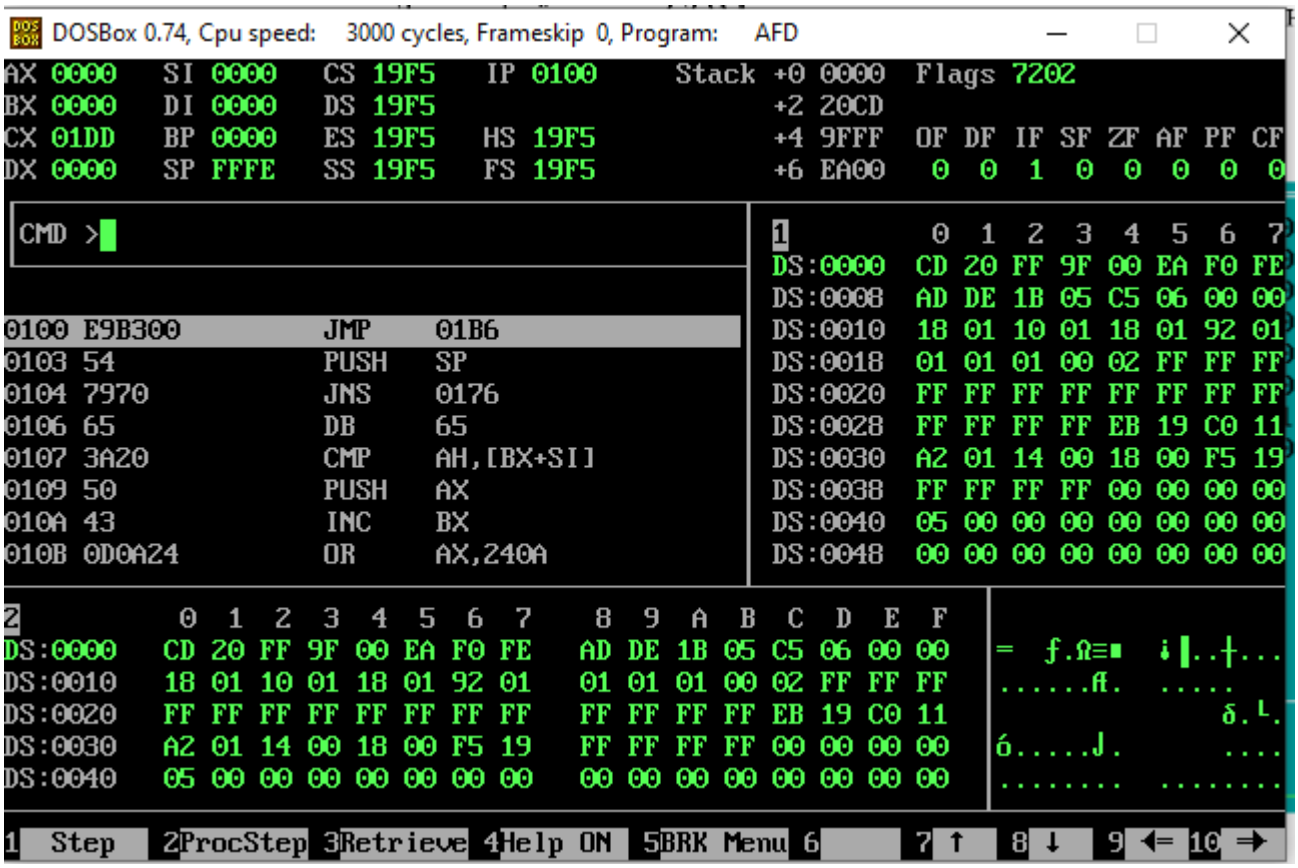
Загрузка COM модуля в основную память.

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

   Сначала выделяется свободный сегмент памяти под программу, первые 100h байт выделенной памяти отводятся под PSP, сразу после него в память

загружается файл COM. На рисунке 8 показан COM-файл запущенный в отладчике AFD.

Рисунок 8 — Запуск COM-программы в отладчике.



2) Что располагается с адреса 0?

   Начиная с адреса 0 располагается 100h байт PSP.

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

   Сегментные регистры — CS, DS, ES, SS – имеют значение 19F5, что показано на рисунке 8, и указывают на начало выделенной памяти — PSP.

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

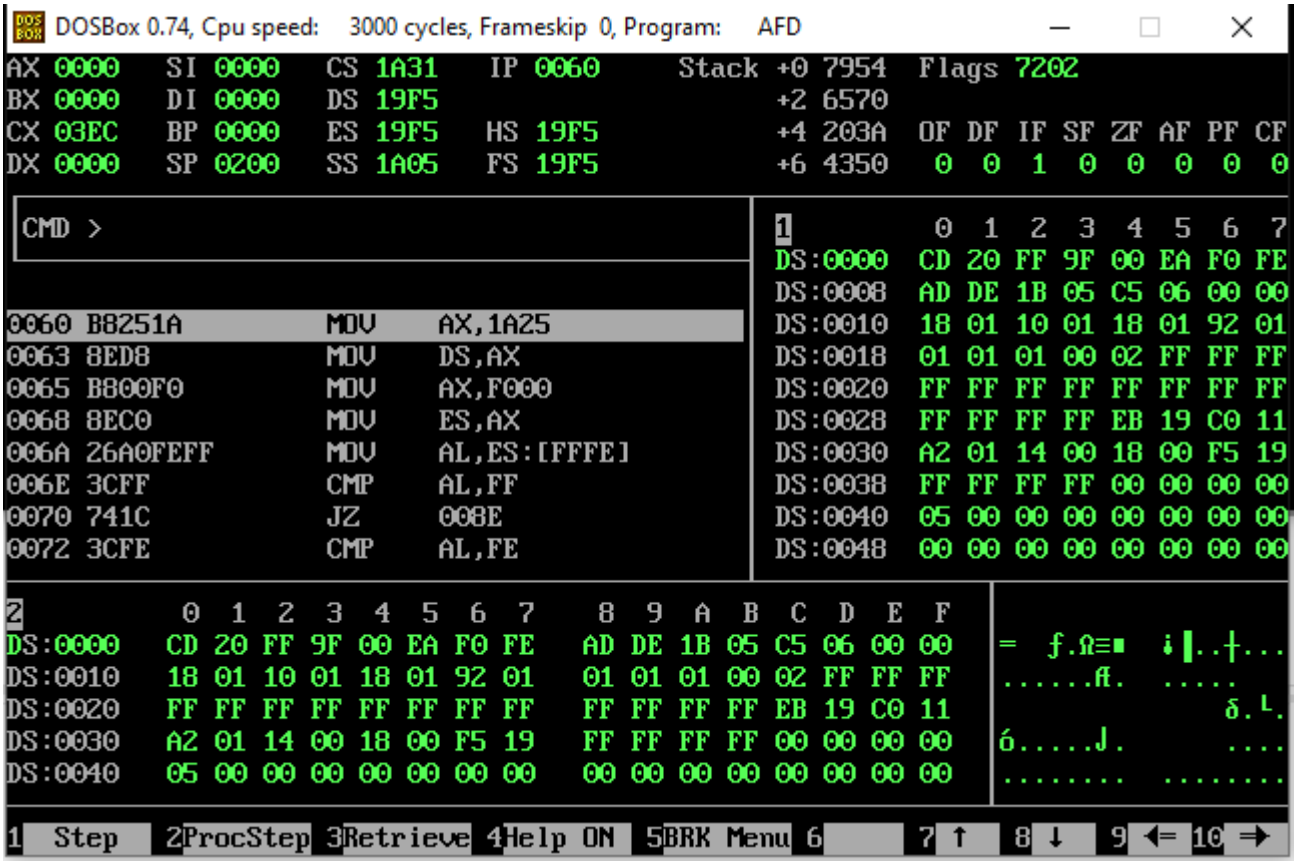
   В COM-программе стек генерируется автоматически. Он располагается в выделенной памяти сразу за содержимым COM-файла. Адреса стека могут находиться в памяти от начала выделенного сегмента памяти, до адреса–FFFE, хранящегося в регистре SP , он указывает на последнюю доступную ячейку памяти в сегменте.

Загрузка «хорошего» EXE модуля в основную память.

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

   Сначала выделяется свободный сегмент памяти под программу, вначале загружается PSP, затем сама программа. Регистры DS, ES указывают на начало PSP, SS – на начало сегмента стека, CS – на начало сегмента команд. Также IP будет указывать на первую к выполнению команду. На рисунке 9 показан «хороший» EXE-файл запущенный в отладчике AFD.

   Рисунок 9 — Запуск «хорошего» EXE-файла в отладчике.



2) На что указывают регистры DS и ES?

   Данные регистры указывают на начало PSP.

3) Как определяется стек?

   Имя_сегмента SEGMENT STACK

   DW Количество_выделяемой_памяти DUP(?)

   Имя_сегмента ENDS

   ASSUME SS:Имя_сегмента

После этого регистр SS будет указывать на начало сегмента стека.

4) Как определяется точка входа?

Точка входа определяется с помощью директивы END — после нее идет название метки или функции с которой будет начато выполнение программы.

Выводы

В ходе выполнения лабораторной работы были написаны два варианта программы, выводящей на экран сведения о типе PC, версии системы, номере OEM и номере пользователя, написанной на ассемблере. Из исходных кодов были получены три различных загрузочных модуля: COM, «хороший» и «плохой» EXE. Были исследованы различия в структурах исходных кодов этих модулей, особенностей работы с каждым из них, структур файлов модулей, а также особенностях их загрузки в память.

ПРИЛОЖЕНИЕ А

Исходный код программы l1_com.asm

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

typePC db  'Type: PC',0DH,0ah,'$'
typeXT db  'Type: PC/XT',0DH,0ah,'$'
typeAT db  'Type: AT',0DH,0ah,'$'
typePS2_30 db  'Type: PS2 model 30',0DH,0ah,'$'
typePS2_80 db  'Type: PS2 model 80',0DH,0ah,'$'
typePCJR db  'Type: PCjr',0DH,0ah,'$'
typePCConv db  'Type: PC Convertible',0DH,0ah,'$'
Endl db '       ', 0DH,0ah,'$'
Version db  'Version:  .  ',0DH,0ah,'$'
Version2 db  'Version <2.0',0DH,0ah,'$'
OEM db  'OEM: ',0DH,0ah,'$'
UserNumber db 'User: ', 0ah, '      ', 0DH,0ah,'$'


BEGIN:
       mov ax,0F000H
       mov es,ax
       mov al,es:[0FFFEH]

       cmp al, 0FFH
              je tPC
       cmp al, 0FEH
              je tPC_XT
       cmp al, 0Fbh
              je tPC_XT
       cmp al, 0FCH
              je tAT
       cmp al, 0Fah
              je tPS2_30
       cmp al, 0F8H
              je tPS2_80
       cmp al, 0FDH
              je tPCjr
       cmp al, 0F9H
              je tPCconv


tPC:
       mov dx, offset typePC
       jmp writeType

tPC_XT:
       mov dx, offset typeXT
       jmp writeType

tAT:
       mov dx, offset typeAT
       jmp writeType

tPS2_30:
       mov dx, offset typePS2_30
       jmp writeType
```

```
tPS2_80:
	mov dx, offset typePS2_80
	jmp writeType

tPCjr:
	mov dx, offset typePCJR
	jmp writeType

tPCconv:
	mov dx, offset typePCConv
	jmp writeType

writeType:
	mov ah, 09h
	int 21h
	jmp OSvertion

;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
OSvertion:
	mov ah, 30h
	int 21h
	push cx
	push bx
printVer:
	mov ah,30h
	int 21h
	push ax
	cmp al, 0
		je ver2
	mov si,offset Version
	add si,9
	call ByteToDec
	pop ax
	mov al,ah
	add si,3
	call ByteToDec
	mov dx,offset Version
	mov ah,09h
	int 21h
	jmp OEMnumber
ver2:
	mov dx,offset Version2
	mov ah,09h
	int 21h
	pop ax
	jmp OEMnumber
OEMnumber:
	mov si,offset OEM
	add si,5
	mov al,bh
	call ByteToDec
	mov dx,offset OEM
	mov ah,09h
	int 21h
	jmp User
User:
	mov di,offset UserNumber
	add di,11
	mov ax,cx
	call WrdToHex
	mov al,bl
	call ByteToHex
```

```
	sub di,2
	mov [di],ax
	mov dx,offset UserNumber
	mov ah,09h
	int 21h
ext:
	xor al, al
	mov ah, 4Ch
	int 21h
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
PrintEndl PROC near
	push ax
	mov dx, offset Endl
	mov ah, 09h
	int 21h
	pop ax
PrintEndl ENDP

TetrToHex PROC near
	and al,0Fh
	cmp al,09
		jbe next
	add al,07
next:
	add al,30h
	ret
TetrToHex ENDP

ByteToHex PROC near
	push cx
	mov ah,al
	call TetrToHex
	xchg al,ah
	mov cl,4
	shr al,cl
	call TetrToHex
	pop cx
	ret
ByteToHex ENDP

WrdToHex PROC near
	push bx
	mov bh,ah
	call ByteToHex
	mov [di],ah
	dec di
	mov [di],al
	dec di
	mov al,bh
	call ByteToHex
	mov [di],ah
	dec di
	mov [di],al
	pop bx
	ret
WrdToHex ENDP

ByteToDec PROC near
	push cx
	push dx
	xor ah,ah
	xor dx,dx
```

```
       mov cx,10
bd:
       div cx
       or dl,30h
       mov [si],dl
       dec si
       xor dx,dx
       cmp ax,10
              jae bd
       cmp al,00h
              je Endbd
       or al,30h
       mov [si],al
Endbd:
       pop dx
       pop cx
       ret
ByteToDec ENDP

TesTPC ENDS
              END START
```

ПРИЛОЖЕНИЕ В

Исходный код программы l1_exe.asm

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

DATA SEGMENT
typePC db  'Type: PC',0DH,0ah,'$'
typeXT db  'Type: PC/XT',0DH,0ah,'$'
typeAT db  'Type: AT',0DH,0ah,'$'
typePS2_30 db  'Type: PS2 model 30',0DH,0ah,'$'
typePS2_80 db  'Type: PS2 model 80',0DH,0ah,'$'
typePCJR db  'Type: PCjr',0DH,0ah,'$'
typePCConv db  'Type: PC Convertible',0DH,0ah,'$'
Endl db '       ', 0DH,0ah,'$'
Version db  'Version:  .  ',0DH,0ah,'$'
Version2 db  'Version <2.0',0DH,0ah,'$'
OEM db  'OEM: ',0DH,0ah,'$'
UserNumber db 'User: ', 0ah, '      ', 0DH,0ah,'$'

DATA ENDS

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

PrintEndl PROC near
       push ax
       mov dx, offset Endl
       mov ah, 09h
       int 21h
       pop ax
PrintEndl ENDP

TetrToHex PROC near
       and al,0Fh
       cmp al,09
              jbe next
       add al,07
next:
       add al,30h
       ret
TetrToHex ENDP

ByteToHex PROC near
       push cx
       mov ah,al
       call TetrToHex
       xchg al,ah
       mov cl,4
       shr al,cl
       call TetrToHex
       pop cx
       ret
ByteToHex ENDP

WrdToHex PROC near
       push bx
       mov bh,ah
       call ByteToHex
       mov [di],ah
```

```
        dec di
        mov [di],al
        dec di
        mov al,bh
        call ByteToHex
        mov [di],ah
        dec di
        mov [di],al
        pop bx
        ret
WrdToHex ENDP

ByteToDec PROC near
        push cx
        push dx
        xor ah,ah
        xor dx,dx
        mov cx,10
bd:
        div cx
        or dl,30h
        mov [si],dl
        dec si
        xor dx,dx
        cmp ax,10
                jae bd
        cmp al,00h
                je Endbd
        or al,30h
        mov [si],al
Endbd:
        pop dx
        pop cx
        ret
ByteToDec ENDP
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
Main PROC FAR
        mov ax, DATA
        mov ds, ax
        mov ax,0F000H
        mov es,ax
        mov al,es:[0FFFEH]
        cmp al, 0FFH
                je tPC
        cmp al, 0FEH
                je tPC_XT
        cmp al, 0Fbh
                je tPC_XT
        cmp al, 0FCH
                je tAT
        cmp al, 0Fah
                je tPS2_30
        cmp al, 0F8H
                je tPS2_80
        cmp al, 0FDH
                je tPCjr
        cmp al, 0F9H
                je tPCconv
tPC:
        mov dx, offset typePC
        jmp writeType
tPC_XT:
```

```
       mov dx, offset typeXT
       jmp writeType
tAT:
       mov dx, offset typeAT
       jmp writeType
tPS2_30:
       mov dx, offset typePS2_30
       jmp writeType
tPS2_80:
       mov dx, offset typePS2_80
       jmp writeType
tPCjr:
       mov dx, offset typePCJR
       jmp writeType
tPCconv:
       mov dx, offset typePCConv
       jmp writeType
writeType:
       mov ah, 09h
       int 21h
       jmp OSvertion

;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
OSvertion:
       mov ah, 30h
       int 21h
       push cx
       push bx
printVer:
       mov ah,30h
       int 21h
       push ax
       cmp al, 0
              je ver2
       mov si,offset Version
       add si,9
       call ByteToDec
       pop ax
       mov al,ah
       add si,3
       call ByteToDec
       mov dx,offset Version
       mov ah,09h
       int 21h
       jmp OEMnumber
ver2:
       mov dx,offset Version2
       mov ah,09h
       int 21h
       pop ax
       jmp OEMnumber
OEMnumber:
       mov si,offset OEM
       add si,5
       mov al,bh
       call ByteToDec
       mov dx,offset OEM
       mov ah,09h
       int 21h
       jmp User
User:
       mov di,offset UserNumber
```

```
        add di,11
        mov ax,cx
        call WrdToHex
        mov al,bl
        call ByteToHex
        sub di,2
        mov [di],ax
        mov dx,offset UserNumber
        mov ah,09h
        int 21h
ext:
        xor al, al
        mov ah, 4Ch
        int 21h

;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

Main ENDP
CODE ENDS
       END Main
```