

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. 9383

Соседков К.С.

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 их загрузки в основную память.

Задание.

Шаг 1. Напишите текст исходного .COM модуля, который определяет тип РС и версию системы.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

Для выполнения первого шага работы на языке ассемблера был написан исходный код модуля COM определяющий тип PC и версию системы(Рисунок 1). Кроме этого был получен «плохой» EXE модуль, полученный из исходного кода для COM модуля(Рисунок 2).

```
C:\>LAB1.COM
IBM PC type: AT
Version: 5.0
OEM: 255
User serial number: 000000
```

Рисунок 1: результат работы
COM

```
C:\>LAB1.EXE

5 0
255
0sIBM PC type: PC000000 PC type: PC/XT
0sIBM PC type: PC
0sIBM PC type: PC
0sIBM PC type: PC
```

Рисунок 2: результат работы „плохого“ EXE

Для выполнения второго шага был написан исходный код EXE модуля, который выполняет те же функции что и COM модуль(Рисунок 3).

```
C:\>LAB1.exe
IBM PC type: AT
Version: 5.0
OEM: 255
User serial number: 000000
```

Рисунок 3: результат работы
„хорошего“ EXE

Вопросы к шагу 3. «Отличия исходных текстов COM и EXE программ».

1) Сколько сегментов должна содержать COM-программа?

COM-программа ограничена размером одного сегмента

2) Сколько сегментов должна содержать EXE-программа?

EXE-программа должна содержать один сегмент или больше.

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

Обязательно должна быть директива `org100h`, так как в DOS первые 256 байт(100h) занимает Program Segment Prefix(PSP). Директива `org100h` говорит о том, что первая команда находится по адресу 100h.

Так же необходима директива `ASSUME`. Данная директива позволяет установить значения сегментных регистров. Как минимум необходимо установить регистр CS. Без установки этого регистра будет ошибка компиляции „Missing or unreachable CS“.

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

Нельзя использовать команды вида `mov <регистр>, seg <имя сегмента>`, т.к. в COM-программе отсутствует таблица настроек.

Для выполнения четвертого шага было выполнено сравнение файлов COM и EXE в шестнадцатеричном виде(см. Рисунки 4,5,6).

```
hp-pro@hppro-laptop: ~/Desktop/lab1
00000000  9 73 01 49 42 4D 20 50 43 20 74 79 70 65 3A 20 .s.IBM PC type:
00000010  50 43 0D 0A 24 49 42 4D 20 50 43 20 74 79 70 65 PC..$IBM PC type
00000020  3A 20 50 43 2F 58 54 0D 0A 24 49 42 4D 20 50 43 : PC/XT..$IBM PC
00000030  20 74 79 70 65 3A 20 41 54 0D 0A 24 49 42 4D 20 type: AT..$IBM
00000040  50 43 20 74 79 70 65 3A 20 50 53 32 20 6D 6F 64 PC type: PS2 mod
00000050  65 6C 20 33 30 0D 0A 24 49 42 4D 20 50 43 20 74 el 30..$IBM PC t
00000060  79 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 33 ype: PS2 model 3
00000070  30 20 6F 72 20 35 30 0D 0A 24 49 42 4D 20 50 43 0 or 50..$IBM PC
00000080  20 74 79 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C type: PS2 model
00000090  20 38 30 0D 0A 24 49 42 4D 20 50 43 20 74 79 70 80..$IBM PC typ
000000A0  65 3A 20 50 43 6A 72 0D 0A 24 49 42 4D 20 50 43 e: PCjr..$IBM PC
000000B0  20 74 79 70 65 3A 20 50 43 20 43 6F 6E 76 65 72 type: PC Conver
000000C0  74 69 62 6C 65 0D 0A 24 49 42 4D 20 50 43 20 74 tible..$IBM PC t
000000D0  79 70 65 3A 20 20 20 20 0D 0A 24 56 65 72 73 69 ype: ..$Versi
000000E0  6F 6E 3A 20 20 2E 20 0D 0A 24 4F 45 4D 3A 20 20 on: ..$OEM:
000000F0  20 20 20 20 20 20 20 20 20 20 20 0D 0A 24 55 73 ..$Us
00000100  65 72 20 73 65 72 69 61 6C 20 6E 75 6D 62 65 72 er serial number
00000110  3A 20 20 20 20 20 20 20 0D 0A 24 50 53 51 52 32 : ..$PSQR2
00000120  E4 33 D2 B9 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 .3.....0..N3.
00000130  3D 0A 00 73 F1 3C 00 74 04 0C 30 88 04 5A 59 5B =..s.<.t..0..ZY[
00000140  58 C3 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 X.$<.v....0.Q..
00000150  E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A .....Y.S.
00000160  FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 .....%0..0.....
00000170  25 4F 88 05 5B C3 33 C0 B8 00 F0 8E C0 26 A0 FE %0..[.3.....&..
00000180  FF 3C FF 74 22 3C FE 74 24 3C FB 74 20 3C FC 74 .<.t"<.t$<.t <.t
00000190  22 3C FA 74 24 3C FC 74 26 3C F8 74 28 3C FD 74 "<.t$<.t&<.t(<.t
000001A0  2A 3C F9 74 2C 75 30 BA 03 01 EB 36 90 BA 15 01 *<.t,u0....6....
000001B0  EB 30 90 BA 2A 01 EB 2A 90 BA 3C 01 EB 24 90 BA .0..*..*..<..$..
000001C0  58 01 EB 1E 90 BA 7A 01 EB 18 90 BA 96 01 EB 12 X.....z.....
000001D0  90 BA AA 01 EB 0C 90 E8 73 FF BF D5 01 89 05 BA .....s.....
000001E0  C8 01 B4 09 CD 21 33 DB 33 C0 B4 30 CD 21 BE E4 .....!3.3..0.!..
000001F0  01 E8 27 FF 8A C4 BE E6 01 E8 1F FF BA DB 01 B4 ..'.....
00000200  09 CD 21 8A C7 BE F1 01 E8 10 FF BA EA 01 B4 09 ..!.....
00000210  CD 21 8A C3 E8 36 FF BF 12 02 89 05 8B C1 BF 17 ..!...6.....
00000220  02 E8 3A FF BA FE 01 B4 09 CD 21 32 C0 B4 4C CD ...:.....!2..L.
00000230  21 !
00000240
00000250
--- LAB1.COM --- --0x0/0x231-----
```

Рисунок 4: COM

Вопросы к шагу 4. «Отличия форматов файлов COM и EXE модулей».

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

Файл типа COM содержит команды и данные. Код располагается с адреса 0h.

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

Данные и код находятся в одном сегменте. Код расположен по адресу 300h. По адресу 0h располагается управляющая информация для загрузчика которая состоит из заголовка и таблицы настроек. Начинается управляющая информация с бита MZ. Данный бит является сигнатурой файла EXE.

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

В „хорошем“ EXE стек, данные и код разделены по сегментам, так же „хороший“ EXE не содержит директивы ORG 100h и поэтому код начинается с адреса 200h.

Вопросы к шагу 5. «Загрузка COM модуля в основную память».

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

Сначала загружается PSP а после этого код. Код располагается с адреса 100h после PSP.

2) Что располагается с адреса 0?

С адреса 0h располагается PSP.

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

При загрузке сегментные регистры указывают на начало PSP.

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

В COM модуле стек создается автоматически и занимает всю область сегмента. Адреса расположены в диапазоне 0h-ffffh.

Вопросы к шагу 6. «Загрузка «хорошего» EXE модуля в основную память».

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

Так же как и COM. Сегментные регистры DS и ES устанавливаются на начало PSP, SS - на начало сегмента стека, CS — на начало сегмента команд.

2) На что указывают регистры DS и ES?

Сегментные регистры DS и ES указывают на начало PSP

3) Как определяется стек?

С помощью регистров SS и SP.

SS указывает на начало, а SS:SP на конец.

4) Как определяется точка входа?

Точка входа определяется директивой END + <метка или процедура>.

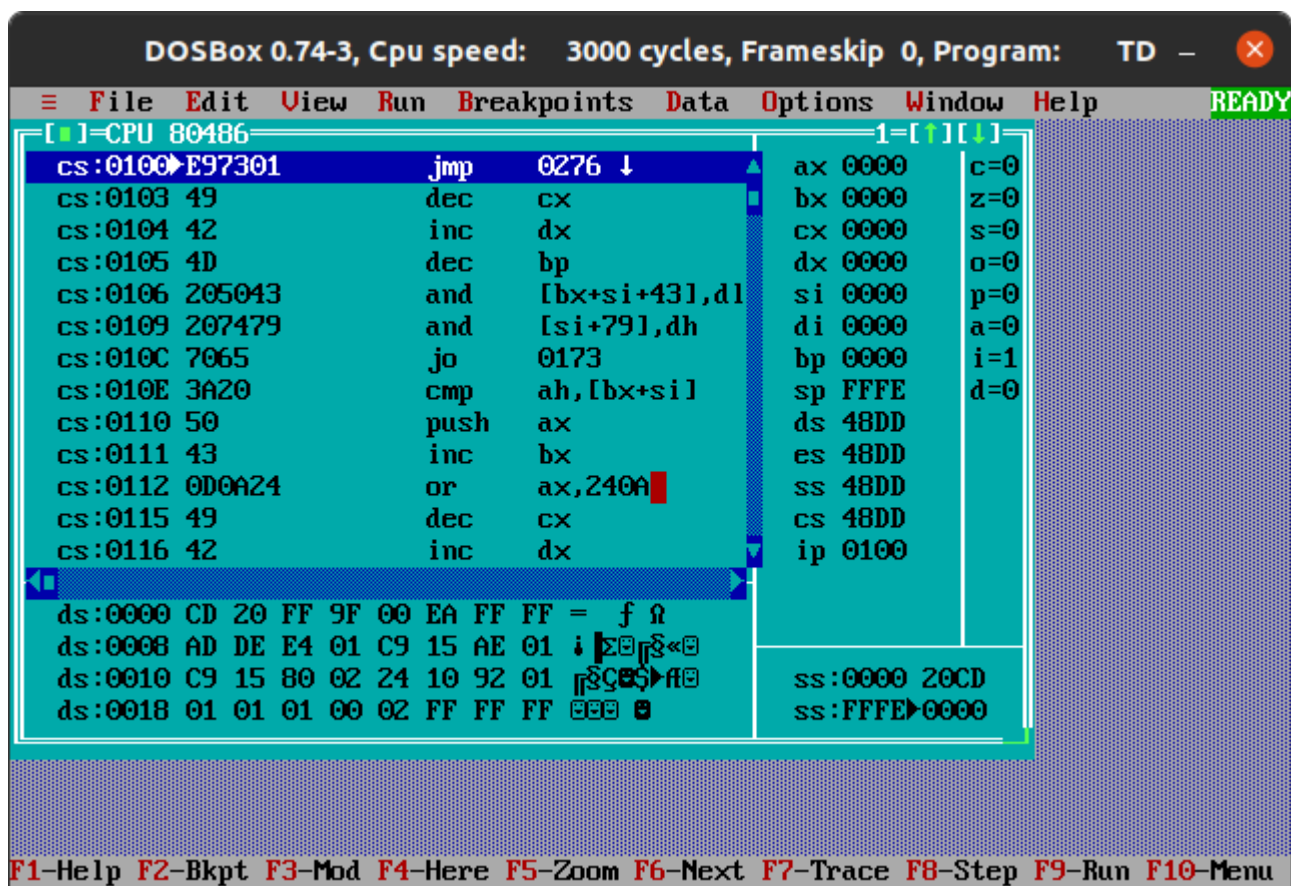


Рисунок 7: TD.exe

Выводы.

При выполнении лабораторной работы были изучены структуры загрузочных модулей COM и EXE.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab1.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H

START: JMP BEGIN

pc db 'IBM PC type: PC',0dh,0ah,'\$'
pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'\$'
at db 'IBM PC type: AT',0dh,0ah,'\$'
ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'\$'
ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'\$'
ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'\$'
pcjr db 'IBM PC type: PCjr',0dh,0ah,'\$'
pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'\$'
unknown db 'IBM PC type: ',0dh,0ah,'\$'
version db 'Version: . ',0dh,0ah,'\$'
oem db 'OEM: ',0dh,0ah,'\$'
user_serial_number db 'User serial number: ',0dh,0ah,'\$'

;-----

BYTE_TO_DEC PROC near

push AX

push BX

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
    pop BX
    pop AX
    ret
BYTE_TO_DEC 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
    ; 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 ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
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
;-----

```

BEGIN:

```

    xor ax,ax
    mov ax,0f000h
    mov es,ax
    mov al,es:[0fffeh]

```

```
    cmp al,0ffh
    je label_pc
    cmp al,0feh
    je label_pc_xt
    cmp al,0fbh
    je label_pc_xt
    cmp al,0fch
    je label_at
    cmp al,0fah
    je label_ps2_30
    cmp al,0fch
    je label_ps2_50_60
    cmp al,0f8h
    je label_ps2_80
    cmp al,0fdh
    je label_pcjr
    cmp al,0f9h
    je label_pc_convertible
    jne label_unknown
```

```
label_pc:
    mov dx,offset pc
    jmp print_ibm_pc_version
label_pc_xt:
    mov dx,offset pc_xt
    jmp print_ibm_pc_version
label_at:
    mov dx,offset at
    jmp print_ibm_pc_version
label_ps2_30:
    mov dx,offset ps2_30
    jmp print_ibm_pc_version
label_ps2_50_60:
```

```

    mov dx,offset ps2_50_60
    jmp print_ibm_pc_version
label_ps2_80:
    mov dx,offset ps2_80
    jmp print_ibm_pc_version
label_pcjr:
    mov dx,offset pcjr
    jmp print_ibm_pc_version
label_pc_convertible:
    mov dx,offset pc_convertible
    jmp print_ibm_pc_version
label_unknown:
    call BYTE_TO_HEX
    mov di, offset unknown+13
    mov [di], ax
    mov dx,offset unknown

```

```

print_ibm_pc_version:
    mov ah,09h
    int 21h

```

```

    xor bx, bx
    xor ax, ax
    mov ah, 30h
    int 21h

```

```

;Version
    mov si, offset version+9
    call BYTE_TO_DEC

```

```

    mov al, ah
    mov si, offset version+11
    call BYTE_TO_DEC

```

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

```
;OEM
mov al, bh
mov si, offset oem+7
call BYTE_TO_DEC
```

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

```
;User serial number
mov al, bl
call BYTE_TO_HEX
```

```
mov di, offset user_serial_number+20
mov [di], ax
```

```
mov ax, cx
mov di, offset user_serial_number+25
call WRD_TO_HEX
```

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

```
exit:
xor al,al
mov ah,4ch
int 21h
TESTPC ENDS
```

END START

Название файла: lab1.exe.asm

AStack SEGMENT STACK

DW 128 DUP(?)

AStack ENDS

DATA SEGMENT

pc db 'IBM PC type: PC',0dh,0ah,'\$'

pc_xt db 'IBM PC type: PC/XT',0dh,0ah,'\$'

at db 'IBM PC type: AT',0dh,0ah,'\$'

ps2_30 db 'IBM PC type: PS2 model 30',0dh,0ah,'\$'

ps2_50_60 db 'IBM PC type: PS2 model 30 or 50',0dh,0ah,'\$'

ps2_80 db 'IBM PC type: PS2 model 80',0dh,0ah,'\$'

pcjr db 'IBM PC type: PCjr',0dh,0ah,'\$'

pc_convertible db 'IBM PC type: PC Convertible',0dh,0ah,'\$'

unknown db 'IBM PC type: ',0dh,0ah,'\$'

version db 'Version: . ',0dh,0ah,'\$'

oem db 'OEM: ',0dh,0ah,'\$'

user_serial_number db 'User serial number: ',0dh,0ah,'\$'

DATA ENDS

CODE SEGMENT

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

;-----

BYTE_TO_DEC PROC near

push AX

push BX

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
pop BX
pop AX
ret

```

BYTE_TO_DEC 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

```

; 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 ;в AL старшая цифра
pop CX ;в AH младшая
ret
BYTE_TO_HEX ENDP

```

```

;-----
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

```

```

;-----

```

```

Main PROC FAR
    mov ax, DATA

```

```
mov ds, ax
```

```
xor ax,ax
```

```
mov ax,0f000h
```

```
mov es,ax
```

```
mov al,es:[0fffeh]
```

```
cmp al,0ffh
```

```
je label_pc
```

```
cmp al,0feh
```

```
je label_pc_xt
```

```
cmp al,0fbh
```

```
je label_pc_xt
```

```
cmp al,0fch
```

```
je label_at
```

```
cmp al,0fah
```

```
je label_ps2_30
```

```
cmp al,0fch
```

```
je label_ps2_50_60
```

```
cmp al,0f8h
```

```
je label_ps2_80
```

```
cmp al,0fdh
```

```
je label_pcjr
```

```
cmp al,0f9h
```

```
je label_pc_convertible
```

```
jne label_unknown
```

```
label_pc:
```

```
mov dx,offset pc
```

```
jmp print_ibm_pc_version
```

```
label_pc_xt:
```

```
mov dx,offset pc_xt
```

```
jmp print_ibm_pc_version
```

```
label_at:
```

```

    mov dx,offset at
    jmp print_ibm_pc_version
label_ps2_30:
    mov dx,offset ps2_30
    jmp print_ibm_pc_version
label_ps2_50_60:
    mov dx,offset ps2_50_60
    jmp print_ibm_pc_version
label_ps2_80:
    mov dx,offset ps2_80
    jmp print_ibm_pc_version
label_pcjr:
    mov dx,offset pcjr
    jmp print_ibm_pc_version
label_pc_convertible:
    mov dx,offset pc_convertible
    jmp print_ibm_pc_version
label_unknown:
    call BYTE_TO_HEX
    mov di, offset unknown+13
    mov [di], ax
    mov dx,offset unknown

```

```

print_ibm_pc_version:

```

```

    mov ah,09h

```

```

    int 21h

```

```

    xor bx, bx

```

```

    xor ax, ax

```

```

    mov ah, 30h

```

```

    int 21h

```

```

;Version

```

```
mov si, offset version+9  
call BYTE_TO_DEC
```

```
mov al, ah  
mov si, offset version+11  
call BYTE_TO_DEC
```

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

```
;OEM  
mov al, bh  
mov si, offset oem+7  
call BYTE_TO_DEC
```

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

```
;User serial number  
mov al, bl  
call BYTE_TO_HEX
```

```
mov di, offset user_serial_number+20  
mov [di], ax
```

```
mov ax, cx  
mov di, offset user_serial_number+25  
call WRD_TO_HEX
```

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

```
exit:
    xor al,al
    mov ah,4ch
    int 21h
```

```
Main ENDP
CODE ENDS
END Main
```