

; Сборка проекта:

```
=====
; tsrinout.asm
;
; Сборка:
; > tasm.exe /l tsrinout.asm
; > tlink /t /x tsrinout.obj
; =====
```

```
code segment 'code'
    assume CS:code, DS:code
    org 100h
```

; Резидентная часть

; начало резидента

```
_start:
    jmp _initTSR                ; переход на начало программы ИНИЦИАЛИЗАЦИИ
```

; данные программы резидента

```
ignoredChars    DB 'abcde'    ; игнорируемые символы
ignoredLength   DB 5          ; длина строки ignoredChars
ignoreEnabled    DB 0          ; флаг функции игнорирования ввода
translateFrom    DB 'QWERTY'   ; заменяемые символы
translateTo      DB 'ЙЦУКЕН'  ; символы, на которые будет происходить замена
translateLength  DB 6          ; длина строки translateFrom
translateEnabled DB 0          ; флаг функции перевода
```

```
signaturePrintingEnabled DB 0    ; флаг вывода подписи
counter                 DW 0
printDelay              EQU 2     ; задержка перед выводом "подписи" в секундах
```

```
signatureLineLength  DW 52       ; длина одной строчки подписи
signatureLine1       DB 179, 'Иванов Иван Иванович'          ', 179
signatureLine2       DB 179, 'ИУ5-4X'                        ', 179
signatureLine3       DB 179, 'Вариант #0'                     ', 179
tableTop             DB '┌', 50 dup ('─'), '┐'
tableBottom          DB '└', 50 dup ('─'), '┘'
```

; Текст справки

```
helpMsg    DB '> kr.com [/?] [/u]', 10, 13
           DB ' [/?] - вывод данной справки', 10, 13
           DB ' [/u] - выгрузка резидента из памяти', 10, 13
           DB ' F1 - вывод ФИО и группы по таймеру в центре экрана', 10, 13
           DB ' F2 - включение/отключения курсивного вывода русского символа В', 10, 13
```

DB ' F3 - включение/отключение частичной русификации клавиатуры:
 QWERTY -> ЙЦУКЕН', 10, 13
 DB ' F4 - включение/отключение режима блокировки ввода букв abcde', 10, 13, 0

helpMsgLength EQU \$-helpMsg
 commandLineResult DB 0

cursiveEnabled DB 0 ; флаг перевода символа в курсив

; Курсивный образ

cursiveSymbol DB 00000000b ; символ, составленный из единиц (его курсивный вариант)

DB 00000000b
 DB 00000000b
 DB 00111110b
 DB 00111111b
 DB 00110011b
 DB 01100110b
 DB 01100110b
 DB 01111100b
 DB 11000110b
 DB 11000110b
 DB 11000110b
 DB 11111100b
 DB 00000000b
 DB 00000000b
 DB 00000000b

; Код символа для курсива

charToCursiveIndex DB 'B' ; символ для замены

; Сохраненный образ символа

savedSymbol DB 16 dup(0FFh) ; переменная для хранения старого символа

; Адреса старых обработчиков

old_int9hOffset DW ? ; адрес старого обработчика int 9h
 old_int9hSegment DW ? ; сегмент старого обработчика int 9h
 old_int1ChOffset DW ? ; адрес старого обработчика int 1Ch
 old_int1ChSegment DW ? ; сегмент старого обработчика int 1Ch
 old_int2FhOffset DW ? ; адрес старого обработчика int 2Fh
 old_int2FhSegment DW ? ; сегмент старого обработчика int 2Fh

; сообщения части резидента

installedMsg DB 'Резидент загружен.', 0
 alreadyInstalledMsg DB 'Резидент уже был загружен.', 0
 notInstalledMsg DB 'Резидент не был загружен.\$'

```

removedMsg          DB 'Резидент выгружен из памяти.'
removedMsg_length    EQU $-removedMsg

noRemoveMsg          DB 'Не удалось выгрузить резидент'
noRemoveMsg_length    EQU $-noRemoveMsg

noRemoveMsg          DB 'Не удалось выгрузить резидент'
noRemoveMsg_length    EQU $-noRemoveMsg

true                 EQU 0FFh      ; нужно для удобства использования not с флагами
                           ; 0FFh = 11111111b = инверсия 00000000b

```

; Новый 09h

```

; новый обработчик прерывания int 9h
; (работа с клавиатурой)
new_int9h proc far
    push SI AX BX CX DX ES DS      ; сохраняем значения всех, изменяемых
    ; регистров в стеке
    push CS                        ; синхронизируем CS и DS
    pop DS

    mov AX, 40h                    ; 40h - сегмент, где хранятся флаги состояния
    ; клавиатуры
    mov ES, AX
    in AL, 60h                     ; записываем в AL сканкод нажатой клавиши
    ; @@@ следующий блок отвечает за выгрузку по Ctrl-U: @@@
    cmp AL, 22                     ; была нажата клавиша U?
    jne _test_Fx
    mov AH, ES:[17h]                ; флаги клавиатуры
    and AH, 00001111b
    cmp AH, 00000100b               ; был ли нажат Ctrl?
    jne _test_Fx

    mov AH, 0FFh                   ; завершаем обработку нажатия
    mov AL, 01h                    ; и выгружаем резидент
    int 2Fh

```

; Работа с портом ввода и вывода для клавиатуры

(это отличие от модуля с tsrbuff!!! Все остальное как в нем!)

; Из порта получаем скан код – номер клавиши

```

    in AL, 61h                     ; контроллер состояния клавиатуры
    or AL, 10000000b               ; пометим, что клавишу нажали
    out 61h, AL
    and AL, 01111111b              ; пометим, что клавишу отпустили
    out 61h, AL
    mov AL, 20h
    out 20h, AL                    ; отправим в контроллер прерываний признак конца
    ; обработки прерывания

```

```

        jmp _quit                ; выходим из процедуры

; @@@ конец блока @@@

;F1
; здесь из порта получим номер скан-кода (номер клавиши)
_test_Fx:                ; проверка F1-F4
    sub AL, 58            ; в AL теперь номер функциональной клавиши (сдвиг на
58)
    _F1:
        cmp AL, 1          ; F1
        jne _F2
        not signaturePrintingEnabled
        jmp _translateOrIgnore

```

;F2

```

    _F2:
        cmp AL, 2          ; F2
        jne _F3
        not cursiveEnabled
        call toggleCursive    ; перевод символа в курсив и обратно
                                ; в зависимости от флага cursiveEnabled
        jmp _translateOrIgnore
    _F3:

```

;F3

```

cmp AL, 3                ; F3
jne _F4
not translateEnabled
jmp _translateOrIgnore

```

;F4

```

    _F4:
        cmp AL, 4          ; F4
        jne _translateOrIgnore
        not ignoreEnabled
        jmp _translateOrIgnore

```

; Вывод символа без изменения

```

_translateOrIgnore:      ; просто выводим набранный символ на экран
    pushf
    call dword ptr CS:[old_int9hOffset] ; вызываем стандартный обработчик прерывания
    mov AX, 40h          ; 40h - сегмент, где хранятся флаги состояния
клавИАТУРЫ
    mov ES, AX
    mov BX, ES:[1Ch]      ; адрес хвоста

```

```

sub BX, 2h                ; сместимся назад к последнему введённому символу
cmp BX, 1Eh              ; не вышли ли мы за пределы буфера?
jae _go
mov BX, 3Ch              ; хвост вышел за пределы буфера: значит, последний
                        ; введённый символ находится в конце буфера

_go:

mov DX, ES:[BX]          ; в DX 0 введённый символ
cmp ignoreEnabled, true  ; включен ли режим блокировки ввода?
jne _checkTranslate

mov SI, 0                ; да, включен
mov CL, ignoredLength    ; количество игнорируемых символов

```

; контроль блокировок ввода символов (игнорирования/замены)

```

_checkIgnored:
  cmp DL, ignoredChars[SI]    ; проверяем, присутствует ли текущий символ в
  ; списке игнорируемых
  je _block
  inc SI
  loop _checkIgnored          ; закидываем ignoredLength раз
  jmp _checkTranslate

; блокируем
_block:
  mov ES:[1Ch], BX            ; блокировка ввода символа
  ; если по варианту нужно не блокировать ввод символа,
  ; а заменять одни символы другими, замените строку выше строкой
  ; mov ES:[BX], AX
  ; на месте AX может быть '*' для замены всех символов множества ignoredChars на
  ; звёздочки
  ; или, для перевода одних символов в другие - завести массив
  ; replaceWith DB '...', где перечислить символы, на которые пойдёт замена
  ; и раскомментировать строки ниже:
  ; xor AX, AX
  ; mov AL, replaceWith[SI]
  ; mov ES:[BX], AX            ; замена символа
  jmp _quit

```

; перевод символов (translateFrom-> translateTo)

```

_checkTranslate:
  cmp translateEnabled, true  ; включен ли режим перевода?
  jne _quit

  mov SI, 0                  ; да, включен
  mov CL, translateLength    ; кол-во символов для перевода

_checkTranslateLoop:
  cmp DL, translateFrom[SI]   ; присутствует ли текущий символ в списке для
  ; перевода?

```

```

        je _translate
        inc SI
        loop _checkTranslateLoop      ; продолжаем, пока не закончим проверять каждый
символ
        jmp _quit

_translate:
        xor AX, AX                    ; переводим
        mov AL, translateTo[SI]
        mov ES:[BX], AX              ; замена символа

_quit:
        pop DS ES DX CX BX AX SI     ; восстанавливаем все регистры
        iret
new_int9h endp

```

; Новый 2Fh

```

; обработчик прерывания int 2Fh
; служит для:
; 1) проверки факта присутствия TSR в памяти (при AH=0FFh, AL=0)
;    будет возвращён AH='i' в случае, если TSR уже загружен
; 2) выгрузки TSR из памяти (при AH=0FFh, AL=1)
new_int2Fh proc
        cmp AH, 0FFh                 ; наша процедура?
        jne _2Fh_default             ; нет - на стандартный обработчик
        cmp AL, 0                     ; подпроцедура проверки, загружен ли резидент в память?
        je _alreadyInstalled2Fh
        cmp AL, 1                     ; подпроцедура выгрузки из памяти?
        je _uninstall
        jmp _2Fh_default

```

; Вызов стандартного обработчика

```

_2Fh_default:
        jmp dword ptr CS:[old_int2FhOffset] ; вызов стандартного обработчика

```

; TSR Уже в памяти

```

_alreadyInstalled2Fh:
        mov AH, 'i'                  ; пусть AH = 'i', если резидент уже загружен в память
        iret                         ; конечно, вместо 'i' может быть любое значение

```

; выгрузка резидента

```

_uninstall:
        push DS ES DX BX             ; подпроцедура выгрузки из памяти
        xor BX, BX

        push CS                      ; CS = ES, для доступа к переменным
        pop ES

```

; Восстановление старых обработчиков

; 9H

```
mov AX, 2509h
mov DX, ES:old_int9hOffset      ; возвращаем вектор прерывания 09h на место
mov DS, ES:old_int9hSegment
int 21h
```

; 1CH

```
mov AX, 251Ch
mov DX, ES:old_int1ChOffset     ; возвращаем вектор прерывания 1Ch на место
mov DS, ES:old_int1ChSegment
int 21h
```

; 2FH

```
mov AX, 252Fh
mov DX, ES:old_int2FhOffset     ; возвращаем вектор прерывания 2Fh на место
mov DS, ES:old_int2FhSegment
int 21h
```

```
mov ES, CS:2Ch                 ; загрузим в ES адрес окружения
mov AH, 49h                    ; выгрузим из памяти окружение
int 21h
jc _notRemove
```

; Освобождение памяти под резидент

```
push CS
pop ES                         ; в ES - адрес резидентной программы
mov AH, 49h                    ; выгрузим из памяти резидент
int 21h
```

; не удалось выполнить выгрузку

```
jc _notRemove
jmp _unloaded
```

```
_notRemove:                  ; не удалось выполнить выгрузку => вывод ошибки
mov AH, 03h                  ; получаем позицию курсора
int 10h
lea BP, noRemoveMsg
mov CX, noRemoveMsg_length
mov BL, 0111b
mov AX, 1301h
int 10h
jmp _2Fh_exit
```

; выгрузка прошла успешно

```
_unloaded:                                ; выгрузка прошла успешно => вывод сообщения
    mov AH, 03h                            ; получаем позицию курсора
    int 10h
    lea BP, removedMsg
    mov CX, removedMsg_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h

_2Fh_exit:
    pop BX DX ES DS
    iret
new_int2Fh endp
```

;Новый 1Ch

```
; обработчик прерывания int 1Ch
; вызывается каждые 55 мс
new_int1Ch proc far
    push AX
    push CS
    pop DS

    pushf
    call dword ptr CS:[old_int1ChOffset] ; вызываем стандартный обработчик
    прерывания

    cmp signaturePrintingEnabled, true ; если нажата управляющая клавиша (в данном
    случае F1)
    jne _notToPrint
```

; Проверяем число тиков (printDelay – задано в сек.)

```
    cmp counter, printDelay*1000/55 + 1 ; если кол-во "тактов" равно printDelay
    секундам
    je _letsPrint
```

; Выводим сообщение по времени

```
    jmp _dontPrint

_letsPrint:
    not signaturePrintingEnabled
    mov counter, 0
    call printSignature ; выводим подпись на экран
```


; НЕ Выводим сообщение по времени (увеличиваем счетчик counter)

```
_dontPrint:
    inc counter                ; увеличим значение счетчика на 1

_notToPrint:
    pop AX
    iret
new_int1Ch endp
```

; Основная часть программы
; 1) установка видеорежима
; 2) проверка, запущен ли резидент
; 3) установка вектора прерываний

; Часть ИНИЦИАЛИЗАЦИИ

```
_initTSR:
```

; 1) установка видеорежима

```
mov AH, 03h
int 10h
push DX
mov AH, 00h                ; установка видеорежима
mov AL, 83h
int 10h
pop DX
mov AH, 02h
int 10h
```

; Аргументы командной строки

```
call commandParamsParser    ; читаем аргументы командной строки
cmp commandLineResult, 2    ; если результат = 2, значит была выведена
справка
jne _shouldContinue        ; соответственно, никаких других действий делать не
нужно
jmp _exit
_shouldContinue:
; ### следующий блок отвечает за выгрузку при аргументе командной строки /u и при
простом перезапуске ###
cmp commandLineResult, 1    ; проверяем результат работы процедуры
jne _go_on
mov AH, 0FFh
mov AL, 0
int 2Fh                    ; проверка того, загружена ли уже программа
cmp AH, 'i'                ; если запущена, то AH = 'i' (см. процедуру new_int2Fh)
je _remove
```

```

mov AH, 09h
lea DX, notInstalledMsg      ; не была загружена
int 21h
int 20h
_go_on:
; ### конец блока ###
; @@@ отвечает за выгрузку при перезапуске @@@
mov AH, 0FFh                ; ещё раз проверяем, запущен ли резидент сейчас
mov AL, 0
int 2Fh
cmp AH, 'i'                  ; если запущена, то AH = 'i' (см. процедуру new_int2Fh)
je _alreadyInstalled
; @@@ конец блока @@@

```

; Адреса старых обработчиков

```

mov AX, 3509h                ; получить в ES:BX прерывания 09h
int 21h
mov word ptr CS:old_int9hOffset, BX ; обработчик прерывания 09h
mov word ptr CS:old_int9hSegment, ES
mov AX, 2509h                ; установим вектор на прерывание 09h
mov DX, offset new_int9h
int 21h

```

```

mov AX, 351Ch                ; получить в ES:BX прерывания 1Ch
int 21h
mov word ptr CS:old_int1ChOffset, BX ; обработчик прерывания 1Ch
mov word ptr CS:old_int1ChSegment, ES
mov AX, 251Ch                ; установим вектор на прерывание 1Ch
mov DX, offset new_int1Ch
int 21h

```

```

mov AX, 352Fh                ; получить в ES:BX прерывания 2Fh
int 21h
mov word ptr CS:old_int2FhOffset, BX ; обработчик прерывания 2Fh
mov word ptr CS:old_int2FhSegment, ES
mov AX, 252Fh                ; установим вектор на прерывание 2Fh
mov DX, offset new_int2Fh
int 21h

```

; Сообщение о загрузке резидента

```

lea BX, installedMsg        ; выводим сообщение, что всё ОК
call printStr

```

; Остаться в памяти РЕЗИДЕНТОМ!

```

mov DX, offset _initTSR     ; остаемся в памяти и выходим из основной части
int 27h

_remove:                    ; выгрузка из памяти
push ES

```

```

mov AX, DS:[2Ch]          ; PSP
mov ES, AX
mov AH, 49h               ; хватит памяти чтоб остаться резидентом?
int 21h
pop ES

mov AH, 0FFh
mov AL, 1
int 2Fh
jmp _exit

```

; TSR Уже в памяти !

```

_alreadyInstalled:        ; резидент уже запущен
    lea BX, alreadyInstalledMsg
    call printStr
    jmp _exit
_exit:                    ; выход
    int 20h

```

; парсер аргументов командной строки. выводит справку.
 ; устанавливает флаг commandLineResult:
 ; 0 = всё ОК; 1 = нужна выгрузка; 2 = была выведена справка, не нужно загружать
 резидент

; Процедура проверки параметров при запуске!

```

commandParamsParser proc
    push CS
    pop ES

    mov SI, 80h           ; SI = смещение командной строки
    lodsb                 ; получим кол-во символов
    or AL, AL             ; если 0 символов введено,
    jz _paramParsingEndWithUnload ; ### то дополнительная проверка, был ли уже
загружен                  ; ### резидент. в таком случае он выгружается

_nextChar:
    inc SI                ; теперь SI указывает на первый символ строки

    cmp [SI], BYTE ptr 0
    je _paramParsingEnd

    lodsw                 ; получаем два символа
    cmp AX, '?'
    je _displayHelp
; @@@@ следует раскомментировать, если нужно выгружать по аргументу /u @@@@
    cmp AX, 'u'
    je _finishTSR
    cmp AX, 'U'
    je _finishTSR

```

```

        jmp _nextChar

_finishTSR:
    mov commandLineResult, 1        ; флаг того, что необходимо выгрузить резидент
    jmp _nextChar
; @ @ @ конец блока @ @ @

        jmp _paramParsingEnd
_displayHelp:
    lea BX, helpMsg                ; выводим справку
    call printStr
    mov commandLineResult, 2        ; флаг того, что резидент загружать не надо

; ### далее - проверка: если резидент уже загружен, то выгрузить ###
    jmp _paramParsingEnd
_paramParsingEndWithUnload:
    mov AH, 0FFh
    mov AL, 0
    int 2Fh
    cmp AH, 'i'                    ; проверка того, загружена ли уже программа
    jne _paramParsingEnd

    mov commandLineResult, 1
; ### конец блока ###

_paramParsingEnd:
    ret
commandParamsParser endp

```

; Печать символа

```

; отображает символ из AL
printChar proc
    mov AH, 0EH
    int 010H
    ret
printChar endp

```

; Печать строки в резиденте (посимвольно)

```

; отображает нуль-терминированную строку из [BX]
printStr proc
    push DX AX
    mov AX, [BX]
_printStrLoop:
    cmp AL, 0
    je _printStrEnd
    call printChar
    inc BX
    mov AX, [BX]
    jmp _printStrLoop
_printStrEnd:

```

```

    pop AX DX
    ret
printStr endp

```

; Процедуры для курсива

; в зависимости от флага cursiveEnabled меняет начертание символа на курсив и обратно
; сама смена происходит в процедуре changeFont - здесь же подготавливаются данные

; Переключение на курсив

```

toggleCursive proc
    push ES AX                ; сохраняем регистры
    push CS
    pop ES

    cmp cursiveEnabled, true  ; если флаг равен true,
    jne _restoreSymbol        ; выполняем замену символа на курсивный вариант,
                                ; предварительно сохраняя старый символ в savedSymbol

    call saveFont
    mov CL, charToCursiveIndex

_shiftTable:
    add BP, 16                ; получаем в BP таблицу всех символов. адрес указывает
на символ 0                    ; поэтому нужно совершить сдвиг 16*X - где X - код символа

    loop _shiftTable

    push DS                    ; при savefont смещается регистр ES
    pop AX                     ; поэтому приходится делать такие махинации, чтобы
    push ES                    ; записать полученный элемент в savedSymbol
    pop DS
    push AX                    ; DS -> AX, ES -> DS, AX -> ES => ES и DS поменялись
местами
    pop ES                      ; + сохранение старого значения DS в AX
    push AX

    mov SI, BP
    lea DI, savedSymbol        ; сохраняем в переменную savedSymbol таблицу
нужного символа

    mov CX, 16                 ; movsb из DS:SI в ES:DI

    rep movsb                  ; исходные позиции сегментов возвращены
    pop DS                     ; восстановление DS

    mov CX, 1                  ; заменим написание символа на курсив
    mov DH, 0
    mov DL, charToCursiveIndex
    lea BP, cursiveSymbol
    call changeFont
    jmp _exitToggleCursive

```

```

_restoreSymbol:
    mov CX, 1                ; если флаг равен 0, заменяем курсивный символ на
старый вариант
    mov DH, 0
    mov DL, charToCursiveIndex
    lea BP, savedSymbol
    call changeFont

_exitToggleCursive:
    pop AX
    pop ES
    ret
toggleCursive endp

; функция смены начертания символа (курсив/нормальное)
;
; входные данные:
; 1) DL = номер символа для замены
; 2) CX = количество символов заменяемых изображений символов
;   (начиная с символа указанного в DX)
; 3) ES:BP = адрес таблицы
;
; описание работы процедуры:
; 1) происходит вызов int 10h (видеосервис)
;   с функцией AH = 11h (функции знакогенератора)
;   параметр AL = 0 сообщает, что будет заменено изображение
;   символа для текущего шрифта.
;   в случаях, когда AL = 1 или 2, будет заменено изображение
;   только для определенного шрифта (8x14 и 8x8 соответственно)
; 2) параметр BH = 0Eh сообщает, что на определение каждого изображения символа
;   расходуется по 14 байт (режим 8x14 бит как раз 14 байт)
; 3) параметр BL = 0 - блок шрифта для загрузки (от 0 до 4)
;
; результат:
; изображение указанного(ых) символа(ов) будет заменено
; на предложенное пользователем.
; изменению подвергнутся все символы, находящиеся на экране:
; таким образом, если изображение заменено, старый вариант нигде уже не проявится

```

; Изменение шрифта в таблице

```

changeFont proc
    push AX BX DX
    mov AX, 1100h
    mov BX, 1000h
    int 10h
    pop DX BX AX
    ret
changeFont endp

```

```

; функция сохранения нормального начертания символа

```

```

;
; входные данные:
; ВН - тип возвращаемой символьной таблицы
; = 0 - таблица из int 1fh
; = 1 - таблица из int 44h
; = 2..5 - таблица из 8x14, 8x8, 8x8 (top), 9x14
; = 6 - 8x16
;
; описание работы процедуры:
; происходит вызов int 10h (видеосервис)
; с функцией АН = 11h (функции знакогенератора)
; параметр AL = 30 - подфункция получения информации о EGA
;
; результат:
; 1) в ES:BP находится таблица символов (полная)
; 2) в CX находится байт на символ
; 3) в DL количество экранных строк
; важно! происходит сдвиг регистра ES (ES = C000h)

```

; сохранение образа для символа шрифта

```

saveFont proc
    push AX BX DX
    mov AX, 1130h
    mov BX, 0600h
    int 10h
    pop BX AX DX
    ret
saveFont endp

; выводит одну строку подписи
printSignatureLine proc
    push DX
    mov CX, signatureLineLength
    mov BL, 0111b                ; цвет выводимого текста
    mov AX, 1301h                ; АН = 13h - номер ф-ии, AL = 01h - перемещение
    курсора
    int 10h
    pop DX
    inc DH
    ret
printSignatureLine endp

```

; процедура вывода подписи (сообщения)

```

; процедура вывода подписи
printSignature proc
    push AX DX CX BX ES SP BP SI DI

    xor AX, AX                  ; обнуляем значения регистров
    xor BX, BX
    xor DX, DX

```

```

    mov AH, 03h                ; чтение текущей позиции курсора
    int 10h
    push DX                    ; помещаем информацию о положении курсора в стек

    mov DX, 090Fh              ; NB! вверху: 000Fh, посередине: 090Fh, внизу: 130Fh

_actualPrint:
    mov AH, 0Fh                ; чтение текущего видеорежима. в ВН - текущая
страница
    int 10h

    push CS
    pop ES                     ; указываем ES на CS

    lea BP, tableTop
    call printSignatureLine     ; выводим верх таблицы
    lea BP, signatureLine1
    call printSignatureLine     ; выводим первую строку
    lea BP, signatureLine2
    call printSignatureLine     ; выводим вторую строку
    lea BP, signatureLine3
    call printSignatureLine     ; выводим третью строку
    lea BP, tableBottom
    call printSignatureLine     ; выводим низ таблицы

    xor BX, BX
    pop DX                     ; восстанавливаем из стека прежнее положение курсора
    mov AH, 02h                ; меняем положение курсора на первоначальное
    int 10h

    pop DI SI BP SP ES BX CX DX AX
    ret
printSignature endp

code ends
end _start

```