

```
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
```

```
charToCursiveIndex DB 'B'
```

;@** СИМВОЛ ДЛЯ ЗАМЕНЫ**

savedSymbol

DB 16 dup(0FFh)

; переменная для хранения старого символа

```
true equ 0FFh
; константа истинности
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

unloadTSR DB 0
; 1 - выгрузить резидент
notLoadTSR DB 0
; 1 - не загружать
counter DW 0
printDelay equ 2
;@ задержка перед выводом "подписи" в секундах
printPos DW 1
;@ положение подписи на экране. 0 - верх, 1 - центр, 2 - низ
```

;@ заменить на собственные данные. формирование таблицы идет по строке большей длины (1я строка).

signatureLine1

DB 179, 'Игорь Латкин, Алексей

Леонтьев,

Константин Назаров', 179

```
Line1_length equ $-signatureLine1
signatureLine2 DB 179, 'ИУ5-44
', 179
Line2_length equ $-signatureLine2
signatureLine3 DB 179, 'Вариант #0
', 179
Line3_length equ $-signatureLine3
```

;Справка****

```
helpMsg DB '>tsr.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 и отключения курсивного вывода русского символа В',
10, 13
DB ' F3 - включение и отключение частичной русификации клавиатуры (F<DUL
-> АБВГД)', 10, 13
DB ' F4 - включение и отключение режима блокировки ввода латинских
строчных букв', 10, 13
```

```
helpMsg_length equ $-helpMsg
errorParamMsg DB 'Ошибка параметров командной
строки', 10, 13
errorParamMsg_length equ $-errorParamMsg

tableTop DB 218, Line1_length-2 dup
(196), 191
tableTop_length equ $-tableTop
tableBottom DB 192, Line1_length-2 dup (196), 217
tableBottom_length equ $-tableBottom

; сообщения
installedMsg DB 'Резидент загружен!$'
```

```

alreadyInstalledMsg      DB  'Резидент уже загружен$'
noMemMsg                 DB  'Недостаточно памяти$'
notInstalledMsg          DB  'Не удалось загрузить резидент$'

removedMsg               DB  'Резидент выгружен'
removedMsg_length        equ  $-removedMsg

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

f1_txt                   DB      'F1'
f2_txt                   DB      'F2'
f3_txt                   DB      'F3'
f4_txt                   DB      'F4'
fx_length                equ     $-f4_txt

```

; Проверка клавиш

```

changeFx proc
    push AX
    push BX
    push CX
    push DX
    push BP
    push ES
    xor BX, BX

    mov AH, 03h
    int 10h
    push DX

    push CS
    pop ES

_checkF1:
    lea BP, f1_txt
    mov CX, fx_length
    mov BH, 0
    mov DH, 0
    mov DL, 78
    mov AX, 1301h

    cmp signaturePrintingEnabled, true
    je _greenF1

    _redF1:
        mov BL, 01001111b ; red
        int 10h
        jmp _checkF2

    _greenF1:
        lea BP, f1_txt
        mov BL, 00101111b ; green
        int 10h

_checkF2:
    lea BP, f2_txt
    mov CX, fx_length
    mov BH, 0
    mov DH, 1
    mov DL, 78
    mov AX, 1301h

    cmp cursiveEnabled, true
    je _greenF2

    _redF2:
        mov BL, 01001111b ; red
        int 10h
        jmp _checkF3

    _greenF2:
        mov BL, 00101111b ; green
        int 10h

_checkF3:
    lea BP, f3_txt
    mov CX, fx_length
    mov BH, 0
    mov DH, 2

```

```

mov DL, 78
mov AX, 1301h

cmp translateEnabled, true
je _greenF3

_redF3:
    mov BL, 01001111b ; red
    int 10h
    jmp _checkF4

_greenF3:
    mov BL, 00101111b ; green
    int 10h

_checkF4:
    lea BP, f4_txt
    mov CX, fx_length
    mov BH, 0
    mov DH, 3
    mov DL, 78
    mov AX, 1301h

    cmp ignoreEnabled, true
    je _greenF4

_redF4:
    mov BL, 01001111b ; red
    int 10h
    jmp _outFx

_greenF4:
    mov BL, 00101111b ; green
    int 10h

_outFx:
    pop DX
    mov AH, 02h
    int 10h

    pop ES
    pop BP
    pop DX
    pop CX
    pop BX
    pop AX
    ret
changeFx endp

```

; новый обработчик new_int9h

```

;новый обработчик
new_int9h proc far
    ; сохраняем значения всех, изменяемых регистров в стеке
    push SI
    push AX
    push BX
    push CX
    push DX
    push ES
    push DS
    ; синхронизируем CS и DS
    push CS
    pop DS

    mov AX, 40h ; 40h-сегмент, где хранятся флаги сост-я клавиатуры, кольц. буфер
ввода

    mov ES, AX
    in AL, 60h ; записываем в AL скан-код нажатой клавиши

    ;@ проверка на Ctrl+U, только для ИУ5-41
    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
    ; выгрузка
    mov AH, 0FFh

```

```

mov AL, 01h
int 2Fh
; завершаем обработку нажатия

```

; Работа с портом в/в

```

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

; выходим
jmp _quit

```

;@ далее - код для всех вариантов

;проверка F1-F4

_test_Fx:

sub AL, 58 ; в AL теперь номер функциональной клавиши

_F1:

```

cmp AL, 1 ; F1
jne _F2
not signaturePrintingEnabled
call changeFx
jmp _translate_or_ignore

```

_F2:

```

cmp AL, 2 ; F2
jne _F3
not cursiveEnabled
call changeFx
call setCursive ; перевод символа в курсив и обратно в зависимости от
флага cursiveEnabled
jmp _translate_or_ignore

```

_F3:

```

cmp AL, 3 ; F3
jne _F4
not translateEnabled
call changeFx
jmp _translate_or_ignore

```

_F4:

```

cmp AL, 4 ; F4
jne _translate_or_ignore
not ignoreEnabled
call changeFx
jmp _translate_or_ignore

```

;игнорирование и перевод

_translate_or_ignore:

; Вызов старого обработчика old_int9hOffset

Pushf

```

call dword ptr CS:[old_int9hOffset] ; вызываем стандартный обработчик прерывания
mov     AX, 40h ; 40h-сегмент, где хранятся флаги сост-я клави, кольцо. буфер

```

ввода

; Работа с клавиатурой

```

mov     ES, AX
mov     BX, ES:[1Ch] ; адрес хвоста
dec     BX ; сместимся назад к последнему
dec     BX ; 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
;включен ли режим блокировки ввода?
cmp ignoreEnabled, true
jne _check_translate

```

; Блокировка ввода символов

```
; да, включен
mov SI, 0
mov CX, ignoredLength ;кол-во игнорируемых символов

; проверяем, присутствует ли текущий символ в списке игнорируемых
_check_ignored:
    cmp DL, ignoredChars[SI]
    je _block
    inc SI
loop _check_ignored
    jmp _check_translate

; 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
```

; Замена символов

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

    ; да, включен
    mov SI, 0
    mov CX, translateLength ; кол-во символов для перевода
    ; проверяем, присутствует ли текущий символ в списке для перевода
    _check_translate_loop:
        cmp DL, translateFrom[SI]
        je _translate
        inc SI
    loop _check_translate_loop
    jmp _quit

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

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

new_int9h endp

;=== Обработчик прерывания int 1Ch ===;
;=== Вызывается каждые 55 мс ===;
```

; Новый обработчик new_int1Ch

```
new_int1Ch proc far
    push AX
    push CS
    pop DS
```

; Вызов старого обработчика old_int1ChOffset

```
    pushf
    call dword ptr CS:[old_int1ChOffset]

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

; Контроль счетчика циклов

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

    jmp _dontPrint

    _letsPrint:
        not signaturePrintingEnabled
        mov counter, 0
        call printSignature

    _dontPrint:
        add counter, 1

    _notToPrint:

    pop AX

    iret
new_int1Ch endp

;=== 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)
;===
```

; Новый обработчик new_int2Fh

```
new_int2Fh proc
    cmp     AH, 0FFh      ;наша функция?
    jne     _2Fh_std      ;нет - на старый обработчик
    cmp     AL, 0         ;подфункция проверки, загружен ли резидент в память?
    je      _already_installed
    cmp     AL, 1         ;подфункция выгрузки из памяти?
    je      _uninstall
    jmp     _2Fh_std      ;нет - на старый обработчик

    _2Fh_std:
```

; Вызов старого обработчика old_int2FhOffset

```
    jmp     dword ptr CS:[old_int2FhOffset] ;вызов старого обработчика

    _already_installed:
        mov     AH, 'i';вернём 'i', если резидент загружен в память
        iret

    _uninstall:
        push    DS
        push    ES
        push    DX
        push    BX

        xor     BX, BX

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

; выгрузка резидента

```

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

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

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

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

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

_notRemove: ; не удалось выполнить выгрузку
        ; вывод сообщения о неудачной выгрузке
        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

_unloaded: ; выгрузка прошла успешно
        ; вывод сообщения об удачной выгрузке
        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
        pop     DX
        pop     ES
        pop     DS
        iret
new_int2Fh endp

;=== Процедура вывода подписи (ФИО, группа)
;=== Настраивается значениями переменных в начале исходника
;===

```

; Вывод подписи

```

printSignature proc
        push    AX
        push    DX
        push    CX
        push    BX
        push    ES
        push    SP
        push    BP
        push    SI
        push    DI

        xor     AX, AX
        xor     BX, BX
        xor     DX, DX

        mov     AH, 03h                                ; чтение текущей позиции курсора
        int     10h

```



```

push DX
курсора в стек                                ; помещаем информацию о положении

cmp printPos, 0
je _printTop

cmp printPos, 1
je _printCenter

cmp printPos, 2
je _printBottom

; все числа подобраны на глаз...
_printTop:
    mov DH, 0
    mov DL, 15
    jmp _actualPrint

_printCenter:
    mov DH, 9
    mov DL, 15
    jmp _actualPrint

_printBottom:
    mov DH, 19
    mov DL, 15
    jmp _actualPrint

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

    push CS
    pop ES                                     ; указываем ES на CS

    ; вывод 'верхушки' таблицы
    push DX
    lea BP, tableTop                           ; помещаем в BP указатель на выводимую
строку                                         ;
    mov CX, tableTop_length                     ; в CX - длина строки
    mov BL, 0111b                               ; цвет выводимого текста ref:
http://en.wikipedia.org/wiki/BIOS\_color\_attributes
    mov AX, 1301h                               ; AH=13h - номер ф-ии, AL=01h - курсор
перемещается при выводе каждого из символов строки
    int 10h
    pop DX
    inc DH

    ; вывод первой линии
    push DX
    lea BP, signatureLine1
    mov CX, Line1_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h
    pop DX
    inc DH

    ; вывод второй линии
    push DX
    lea BP, signatureLine2
    mov CX, Line2_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h
    pop DX
    inc DH

    ; вывод третьей линии
    push DX
    lea BP, signatureLine3
    mov CX, Line3_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h
    pop DX
    inc DH

```

```

        ;вывод 'низа' таблицы
        push DX
        lea BP, tableBottom
        mov CX, tableBottom_length
        mov BL, 0111b
        mov AX, 1301h
        int 10h
        pop DX
        inc DH

        xor BX, BX
        pop DX
положение курсора
        mov AH, 02h
первоначальное
        int 10h
        call changeFx

        pop DI
        pop SI
        pop BP
        pop SP
        pop ES
        pop BX
        pop CX
        pop DX
        pop AX

        ret
printSignature endp

```

; Смена шрифта

=== Функция, которая в зависимости от флага `cursiveEnabled` меняет начертание символа с курсива на обычное и наоборот

=== Сама смена происходит в процедуре `changeFont`, а здесь подготавливаются данные

```

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

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

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

    ; при savefont смещается регистр ES
    ; поэтому приходится делать такие махинации, чтобы
    ; записать полученный элемент в savedSymbol
    ; swap(ES, DS) и сохранение старого значения DS
    push DS
    pop AX
    push ES
    pop DS
    push AX
    pop ES
    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

    ; заменим написание символа на курсив
    mov CX, 1

```

```

mov DH, 0
mov DL, charToCursiveIndex
lea BP, cursiveSymbol
call changeFont
jmp _exitSetCursive

```

; Восстановление шрифта

```

_restoreSymbol:
    ; если флаг равен 0, выполняем замену курсивного символа на старый вариант

    mov CX, 1
    mov DH, 0
    mov DL, charToCursiveIndex
    lea bp, savedSymbol
    call changeFont

_exitSetCursive:
    pop AX
    pop ES
    ret
setCursive endp

;=== Функция смены начертания символа (курсив/нормальное)
;===
; *** входные данные
; DL = номер символа для замены
; CX = Кол-во символов заменяемых изображений символов
; (начиная с символа указанного в DX)
; ES:bp = адрес таблицы
;
; *** описание работы процедуры
; 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)
; Параметр BH = 0Eh сообщает, что на определение каждого изображения символа
; расходуется по 14 байт (режим 8x14 бит как раз 14 байт)
; 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,
; то есть если изображение заменено, старый вариант нигде уже не проявится

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

;=== Функция сохранения нормального начертания символа
;===
; *** входные данные
; BH - 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 AH = 11h (функции знакогенератора)
; Параметр AL = 30 - подфункция получения информации о EGA
;
; *** результат
; в ES:BP находится таблица символов (полная)
; в CX находится байт на символ
; в DL количество экранных строк
; ВАЖНО! Происходит сдвиг регистра ES
; ( ES становится равным C000h )

```

```

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

```

```

;=== Отсюда начинается выполнение основной части программы ===;
;===

```

; Часть Инициализации

```

_initTSR:                                ; старт резидента
    mov AH, 03h
    int 10h
    push DX
    mov AH, 00h                                ; установка видеорежима (83h текст 80x25
16/8 CGA, EGA b800 Comp, RGB, Enhanced), без очистки экрана
    mov AL, 83h
    int 10h
    pop DX
    mov AH, 02h
    int 10h

```

; Новые вектора Инициализации

```

    call commandParamsParser
    mov AX, 3509h                                ; получить в ES:BX вектор 09
    int 21h                                ; прерывания

;@ === Удаление резидента из памяти ===
;@ Если по варианту необходимо выгружать резидент по повторному запуску приложений,
;@ нужно закомментировать следующие 3 строки, а также
;@ содержимое метки _finishTSR ф-ии commandParamsParser, но не саму метку!
    cmp unloadTSR, true
    je _removingOnParameter
    jmp _notRemovingNow

```

; Проверка загрузки

```

    _removingOnParameter:
        mov AH, 0FFh
        mov AL, 0
        int 2Fh
        cmp AH, 'i' ; проверка того, загружена ли уже программа
        je _remove
        mov AH, 09h                                ;@ для выгрузки резидента по повторному
запуску закомментировать эту строку
        lea DX, notInstalledMsg                    ;@ для выгрузки резидента по повторному запуску
закомментировать эту строку
        int 21h                                ;@ для выгрузки резидента по повторному
запуску закомментировать эту строку
        int 20h                                ;@ для выгрузки резидента по повторному
запуску закомментировать эту строку

    _notRemovingNow:

    cmp notLoadTSR, true                        ; если была выведена справка
    je _exit_tmp                                ; просто выходим

;@ Если по варианту необходимо выгружать резидент по повторному запуску, то комментируем
5 строк ниже
;@ если необходимо выгружать по параметру командной строки, то оставляем их
    mov AH, 0FFh
    mov AL, 0
    int 2Fh
    cmp AH, 'i' ; проверка того, загружена ли уже программа
    je _alreadyInstalled

    jmp _tmp

```

```

_exit_tmp:
    jmp _exit

_tmp:
    push ES

```

; Проверка наличия памяти

```

mov AX, DS:[2Ch]          ; psp
mov ES, AX
mov AH, 49h              ; хватит памяти чтоб остаться
int 21h                  ; резидентом?
pop ES
jc _notMem                ; не хватило - выходим

```

; Сохранение старых векторов и установка новых

```

;== int 09h ==;

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

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

;== int 2Fh ==;
mov AX, 352Fh            ; получить в ES:BX вектор 1C
int 21h                  ; прерывания
mov     word ptr CS:old_int2FhOffset, BX
mov     word ptr CS:old_int2FhSegment, ES
mov AX, 252Fh            ; установим вектор на 2F
mov DX, offset new_int2Fh ; прерывание
int 21h

call changeFx
mov DX, offset installedMsg ; выводим что все ок
mov AH, 9
int 21h

```

; Оставить в ОП резидентом (027H)

```

mov DX, offset _initTSR    ; остаемся в памяти резидентом
int 27h                    ; и выходим
; конец основной программы

```

; Выгрузка резидента (сигнал в TSR)

```

_remove: ; выгрузка программы из памяти
    mov AH, 0FFh
    mov AL, 1
    int 2Fh
    jmp _exit
_alreadyInstalled:
    mov AH, 09h
    lea DX, alreadyInstalledMsg
    int 21h
    jmp _exit
_notMem: ; не хватает памяти, чтобы остаться резидентом
    mov DX, offset noMemMsg
    mov AH, 9
    int 21h
_exit: ; выход
    int 20h

```

;== Процедура проверки параметров ком. строки ==;

```
;===
```

; Проверка и разбор параметров

```
commandParamsParser proc
    push CS
    pop ES
    mov unloadTSR, 0
    mov notLoadTSR, 0

    mov SI, 80h                ;SI=смещение командной строки.
    lodsb                     ;Получим кол-во символов.
    or AL, AL                 ;Если 0 символов введено,
    jz _exitHelp              ;то все в порядке.

    _nextChar:

    inc SI                    ;Теперь SI указывает на первый символ
строки.

    cmp [SI], BYTE ptr 13
    je _exitHelp

    lodsw                     ;Получаем два символа
    cmp AX, '?/'              ;Это '/'? (данные расположены в обратном
порядк, т.е. AL:AH вместо AH:AL)
    je _question
    cmp AX, 'u/'
    je _finishTSR

    ;cmp AH, '/'
    ;je _errorParam

    jmp _exitHelp
```

; Вывод справки

```
_question:
    ; вывод строки помощи
    mov AH, 03
    int 10h
    lea BP, helpMsg
    mov CX, helpMsg_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h
    ; конец вывода строки помощи
    not notLoadTSR            ;флаг того, что необходимо не загружать резидент
    jmp _nextChar

;@ === Удаление резидента из памяти ===
;@ Если по варианту необходимо выгружать резидент по параметру '/u' командной строки,
;@ нужно использовать следующий код, в остальных случаях необходимо закомментировать
;@ этот код, кроме названия метки! (по желанию можно избавиться и от метки, но аккуратно
просмотреть использование)
_finishTSR:
    not unloadTSR            ;флаг того, что необходимо выгрузить резидент
    jmp _nextChar

jmp _exitHelp

_errorParam:
    ;вывод строки
    mov AH, 03
    int 10h
    lea BP, CS:errorParamMsg
    mov CX, errorParamMsg_length
    mov BL, 0111b
    mov AX, 1301h
    int 10h
    ;конец вывода строки
_exitHelp:
    ret
commandParamsParser endp
```

code ends

`end _start`

; Конец части инициализации