Московский государственный технический университет им. Н.Э. Баумана

Утверждаю:

Большаков С.А.
"__"_____ 2025 г.

Курсовая работа по курсу «Системное программирование»
«Резидентная программа (TSR)»

Листинг и исходный код программ
(вид документа)

писчая бумага
(вид носителя)

41
(количество листов)

ИСПОЛНИТЕЛЬ:

студенты группы ИУ5-42Б _____
Афонин И.И.
"__"_____ 2025 г.

Москва – 2025

СОДЕРЖАНИЕ

1. ИСХОДНЫЙ КОД (KR)

```
; ======================================
;  kr.asm
; 13.04.25
;  Сборка:
;  > tasm.exe /l kr.asm
;  > tlink /t /x kr.obj
; ======================================

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

_start:
   jmp _initTSR                        ; на начало программы

   ignoredChars             DB 'ЕЖЗИК'      ; игнорируемые символы
   ignoredLength             DB 5           ; длина строки ignoredChars
   ignoreEnabled             DB 0           ; флаг функции игнорирования ввода
   translateFrom            DB 'T;PBR'     ; заменяемые символы
   translateTo              DB 'ЕЖЗИК'     ; символы, на которые будет
происходить замена
   translateLength           DB 5          ; длина строки translateFrom
   translateEnabled          DB 0          ; флаг функции перевода

   signaturePrintingEnabled    DB 0          ; флаг вывода подписи
   counter                  DW 0
   printDelay                EQU 5         ; задержка перед выводом "подписи" в
секундах

   signatureLineLength        DW  52        ; длина одной строчки подписи
   signatureLine1             DB  179, 'Афонин Иван Игоревич                    ',
179
   signatureLine2             DB  179, 'ИУ5-42Б                                 ', 179
   signatureLine3             DB  179, 'Вариант #2                              ', 179
   tableTop                 DB  '┌', 50 dup ('─'), '┐'
   tableBottom               DB  '└', 50 dup ('─'), '┘'

   helpMsg     DB '> kr.com [/?] ' , 10, 13
            DB ' [/?] - вывод данной справки', 10, 13
```

```
        DB ' выгрузка резидента при повторном вызове программы без параметров', 10, 13
        DB '  F8  - вывод ФИО и группы по таймеру в верху экрана', 10, 13
        DB '  F9  - включение/отключения курсивного вывода русского символа Ъ', 10, 13
        DB '  F1  - включение/отключение частичной русификации клавиатуры: T;PBR -> ЕЖЗИК', 10, 13
        DB '  F2  - включение/отключение режима блокировки ввода букв ЕЖЗИК', 10, 13, 0

    helpMsgLength           EQU $-helpMsg
    commandLineResult       DB 0

    cursiveEnabled          DB 0          ; флаг перевода символа в курсив
    cursiveSymbol           DB 00000000b  ; символ, составленный из единиц (его курсивный вариант)
                            DB 00000000b
                            DB 00000000b
                            DB 00111110b
                            DB 00001100b
                            DB 00001100b
                            DB 00001000b
                            DB 00011110b
                            DB 00010011b
                            DB 00110011b
                            DB 00100011b
                            DB 01100011b
                            DB 01111110b
                            DB 00000000b
                            DB 00000000b
                            DB 00000000b

    charToCursiveIndex      DB 'Ъ'        ; символ для замены
    savedSymbol             DB 16 dup(0FFh) ; переменная для хранения старого символа

    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

    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

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

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

       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

   _test_Fx:                         ; проверка F8-F2
   _F8:
      cmp DH, 42h                       ; F8(?)
      jne _F9
      not signaturePrintingEnabled          ; Флаг печати ФИО
      mov ES:[1Ch], BX                  ; блокировка ввода символа
      jmp _quit
```

```
_F9:
    cmp DH, 43h                    ; F9(?)
    jne _F1
    mov ES:[1Ch], BX                ; блокировка ввода символа
    not cursiveEnabled              ; Установка или сброс курсива
    call toggleCursive              ; перевод символа в курсив и обратно
                                ; в зависимости от флага cursiveEnabled
    jmp _quit
_F1:
    cmp DH, 3Bh                    ; F1(?)
    jne _F2
    not translateEnabled            ; Включение перевода символов
    mov ES:[1Ch], BX                ; блокировка ввода символа
    jmp _quit
_F2:
    cmp DH, 3Ch                    ; F2 (?)
    jne _translateOrIgnore
    not ignoreEnabled               ; Включение блокировки символоа
    mov ES:[1Ch], BX                ; блокировка ввода символа
    jmp _quit

_translateOrIgnore:                 ; просто выводим набранный символ на
экран

    cmp ignoreEnabled, true         ; включен ли режим блокировки ввода?
    jne _checkTranslate

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

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

    _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 BP  ; //////////
        pop DS ES DX CX BX AX SI            ; восстанавливаем все регистры
        iret
    new_int9h endp

    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

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

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

    _2Fh_default:
```

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

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

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

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

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

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

    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

    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
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

   _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

   ; 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

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

```
        je _letsPrint

        jmp _dontPrint

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

    _dontPrint:
            inc counter                    ; увеличим значение счетчика на 1

    _notToPrint:
        pop AX
        iret
    new_int1Ch endp

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

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

  _actualPrint:
    mov AH, 0Fh                  ; чтение текущего видеорежима. в BH -
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

  ; Основная часть программы
  ; 1) установка видеорежима
  ; 2) проверка, запущен ли резидент
  ; 3) установка вектора прерываний
  _initTSR:
    mov AH, 03h
    int 10h
    push DX
    mov AH, 00h                  ; установка видеорежима
    mov AL, 83h
    int 10h
```

```
        pop DX
        mov AH, 02h
        int 10h

        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:
       mov AH, 0FFh
       mov AL, 0
       int 2Fh
       cmp AH, 'i'                        ; проверка того, загружена ли уже программа
       je _remove

        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

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

```
    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
_exit:                          ; выход
    int 20h

; парсер аргментов командной строки. выводит справку.
; устанавливает флаг commandLineResult:
; 0 = всё ОК; 1 = нужна выгрузка; 2 = была выведена справка, не нужно
загружать резидент
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 _paramParsingEnd                 ; то все в порядке

_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

    jmp _paramParsingEnd
```

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

   _paramParsingEnd:
      ret
  commandParamsParser endp

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

   ; 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

code ends
end _start
```

2. ЛИСТИНГ ПРОГРАММЫ (KR)

```
Turbo Assembler    Version 3.1    04/13/25 22:50:33          Page 1
kr.asm



    1                           ;
=====================================
    2                           ;    krbuff.asm
```

```
     3                          ; 14.04.23
     4                          ;    Сборка:
     5                          ;    > tasm.exe /l kr.asm
     6                          ;    > tlink         /t /x kr.obj
     7                          ;
=====================================
     8
     9  0000                    code segment 'code'
    10                              assume        CS:code, DS:code
    11                              org 100h
    12
    13 0100                     _start:
    14 0100  E9 0585                        jmp _initTSR                            ; на
начало программы
    15
    16 0103  85 86 87 88 8A                 ignoredChars                DB
        'ЕЖЗИК'          ; игнорируемые символы
    17 0108  05                     ignoredLength               DB        5          ;
длина строки ignoredChars
    18 0109  00                     ignoreEnabled               DB        0          ;
флаг функции игнорирования ввода
    19 010A  54 3B 50 42 52                 translateFrom               DB
        'T;PBR'          ; заменяемые символы
    20 010F  85 86 87 88 8A                 translateTo             DB        'ЕЖЗИК'
; символы, на которые будет  происходить замена
    21 0114  05                     translateLength         DB        5           ; длина
строки translateFrom
    22 0115  00                     translateEnabled        DB        0           ; флаг
функции перевода
    23
    24 0116  00                     signaturePrintingEnabled    DB        0          ;
флаг вывода подписи
    25 0117  0000                   counter                 DW        0
    26       =0005                  printDelay              EQU 5       ; задержка
        перед выводом "подписи"      в секундах
    27
    28 0119  0034                   signatureLineLength         DW        52         ;
длина одной строчки подписи
    29 011B  B3 80 E4 AE AD A8      AD+   signatureLine1                DB
179, 'Афонин Иван Игоревич                          ', +
    30       20 88 A2 A0 AD 20  88+  179
    31       A3 AE E0 A5 A2 A8E7+
    32       20 20 20 20 20 20      20+
    33       20 20 20 20 20 20      20+
```

```
    34      20 20 20 20 20 20     20+
    35      20 20 20 20 20 20     20+
    36      20 20 B3
    37 014F  B3 88 93 35 2D 34        32+    signatureLine2                DB
179, 'ИУ5-42Б                                        ', +
    38      81 20 20 20 20 20     20+  179
    39      20 20 20 20 20 20     20+
    40      20 20 20 20 20 20     20+
    41      20 20 20 20 20 20     20+
    42      20 20 20 20 20 20     20+
    43      20 20 20 20 20 20     20+
    44      20 20 B3
    45 0183  B3 82 A0 E0 A8 A0       AD+   signatureLine3                DB
179, 'Вариант #2                                     ', +
    46      E2 20 23 32 20 20     20+  179
    47      20 20 20 20 20 20     20+
    48      20 20 20 20 20 20     20+
    49      20 20 20 20 20 20     20+
    50      20 20 20 20 20 20     20+
    51      20 20 20 20 20 20     20+
    52      20 20 B3
    53 01B7  DA 32*(C4) BF            tableTop              DB        '┌', 50 dup
('─'), '┐'
    54 01EB  C0 32*(C4) D9            tableBottom                DB        '└', 50
dup ('─'), '┘'
    55
    56 021F  3E 20 6B 72 2E 63        6F+    helpMsg     DB    '> kr.com [/?] ' , 10,
13
    57      6D 20 5B 2F 3F 5D   20+
```

```
    58      0A 0D
    59 022F  20 5B 2F 3F 5D 20      2D+             DB      ' [/?] - вывод
данной справки',  10, 13
    60      20 A2 EB A2 AE A4 20+
    61      A4 A0 AD AD AE A9       20+
    62      E1 AF E0 A0 A2 AA       A8+
    63      0A 0D
    64 024D  20 A2 EB A3 E0 E3      A7+             DB      ' выгрузка
резидента при повторном вызове программы без      параметров', 10, 13
    65      AA A0 20 E0 A5 A7 A8+
    66      A4 A5 AD E2 A0 20 AF+
    67      E0 A8 20 AF AE A2 E2+
    68      AE E0 AD AE AC 20       A2+
    69      EB A7 AE A2 A5 20 AF+
    70      E0 AE A3 E0 A0 AC       AC+
    71      EB 20 A1 A5 A7 20  AF+
    72      A0 E0 A0 AC A5 E2 E0+
    73      AE A2 0A 0D
    74 0290  20 20 46 38 20 20  2D+             DB      '  F8  - вывод ФИО и
группы по таймеру в верху экрана', 10, 13
    75      20 A2 EB A2 AE A4 20+
    76      94 88 8E 20 A8 20    A3+
    77      E0 E3 AF AF EB 20  AF+
    78      AE 20 E2 A0 A9 AC A5+
    79      E0 E3 20 A2 20 A2   A5+
    80      E0 E5 E3 20 ED AA E0+
    81      A0 AD A0 0A 0D
    82 02C6  20 20 46 39 20 20 2D+              DB      '  F9  -
включение/отключения курсивного вывода      русского символа Ъ', 10, 13
    83      20 A2 AA AB EE E7       A5+
    84      AD A8 A5 2F AE E2       AA+
    85      AB EE E7 A5 AD A8       EF+
    86      20 AA E3 E0 E1 A8  A2+
    87      AD AE A3 AE 20 A2       EB+
    88      A2 AE A4 A0 20 E0 E3+
    89      E1 E1 AA AE A3 AE       20+
    90      E1 A8 AC A2 AE AB       A0+
    91      20 9A 0A 0D
```

```
   92 0309  20 20 46 31 20 20  2D+              DB        '  F1  -
включение/отключение частичной  русификации клавиатуры:      T;PBR ->    +
   93       20 A2 AA AB EE E7        A5+  ЕЖЗИК', 10, 13
   94       AD A8 A5 2F AE E2        AA+
   95       AB EE E7 A5 AD A8        A5+
   96       20 E7 A0 E1 E2 A8  E7+
   97       AD AE A9 20 E0 E3 E1+
   98       A8 E4 A8 AA A0 E6 A8+
   99       A8 20 AA AB A0 A2        A8+
  100       A0 E2 E3 E0 EB 3A  20+
  101       54 3B 50 42 52 20     2D+
  102       3E 20 85 86 87 88     8A+
  103       0A 0D
  1040358  20 20 46 32 20 20  2D+              DB        '  F2  -
включение/отключение режима блокировки      ввода букв ЕЖЗИК', 10, 13, 0
  105       20 A2 AA AB EE E7        A5+
  106       AD A8 A5 2F AE E2        AA+
  107       AB EE E7 A5 AD A8        A5+
  108       20 E0 A5 A6 A8 AC A0+
  109       20 A1 AB AE AA A8        E0+
  110       AE A2 AA A8 20 A2        A2+
  111       AE A4 A0 20 A1 E3 AA+
  112       A2 20 85 86 87 88     8A+
  113       0A 0D 00
  114
```

```
115     =017B                   helpMsgLength           EQU $-helpMsg
116039A  00                     commandLineResult       DB      0
117
118039B  00                     cursiveEnabled          DB      0       ;
флаг перевода символа в курсив
119039C  00                     cursiveSymbol           DB      00000000b
; символ, составленный из единиц (его курсивный+
120                             вариант)
121039D  00                                             DB      00000000b
122039E  00                                             DB      00000000b
123039F  3E                                             DB      00111110b
12403A0  0C                                             DB      00001100b
12503A1  0C                                             DB      00001100b
12603A2  08                                             DB      00001000b
12703A3  1E                                             DB      00011110b
12803A4  13                                             DB      00010011b
12903A5  33                                             DB      00110011b
13003A6  23                                             DB      00100011b
13103A7  63                                             DB      01100011b
13203A8  7E                                             DB      01111110b
13303A9  00                                             DB      00000000b
13403AA  00                                             DB      00000000b
13503AB  00                                             DB      00000000b
136
13703AC  9A                     charToCursiveIndex      DB      'Ь'     ;
символ для замены
13803AD  10*(FF)                savedSymbol             DB      16
dup(0FFh) ; переменная для хранения старого символа
139
14003BD  ????                   old_int9hOffset         DW      ?       ; адрес
старого обработчика int 9h
14103BF  ????                   old_int9hSegment        DW      ?       ; сегмент
старого обработчика int 9h
14203C1  ????                   old_int1ChOffset        DW      ?       ; адрес
старого обработчика int 1Ch
14303C3  ????                   old_int1ChSegment       DW      ?       ;
сегмент старого обработчика int 1Ch
14403C5  ????                   old_int2FhOffset        DW      ?       ; адрес
старого обработчика int 2Fh
```

```
   14503C7  ????                  old_int2FhSegment          DW        ?          ;
сегмент старого обработчика int 2Fh
   146
   14703C9  90 A5 A7 A8 A4 A5     AD+   installedMsg                    DB
'Резидент загружен.', 0
   148      E2 20 A7 A0 A3 E0  E3+
   149      A6 A5 AD 2E 00
   15003DC  90 A5 A7 A8 A4 A5     AD+   alreadyInstalledMsg            DB
'Резидент уже был загружен.', 0
   151      E2 20 E3 A6 A5 20   A1+
   152      EB AB 20 A7 A0 A3E0+
   153      E3 A6 A5 AD 2E 00
   15403F7  90 A5 A7 A8 A4 A5     AD+   notInstalledMsg         DB
'Резидент не был загружен.$'
   155      E2 20 AD A5 20 A1  EB+
   156      AB 20 A7 A0 A3 E0 E3+
   157      A6 A5 AD 2E 24
   158
   1590411  90 A5 A7 A8 A4 A5     AD+   removedMsg                    DB
'Резидент выгружен из памяти.'
   160      E2 20 A2 EB A3 E0  E3+
   161      A6 A5 AD 20 A8 A720+
   162      AF A0 AC EF E2 A8      2E
   163      =001C                  removedMsg_length          EQU $-removedMsg
   164
   165042D  8D A5 20 E3 A4 A0     AB+   noRemoveMsg                  DB
'Не удалось выгрузить резидент'
   166      AE E1 EC 20 A2 EB A3+
   167      E0 E3 A7 A8 E2 EC  20+
   168      E0 A5 A7 A8 A4 A5 AD+
   169      E2
   170      =001D                  noRemoveMsg_length         EQU $-noRemoveMsg
   171
```

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 4
kr.asm

```
172  =00FF                true                EQU 0FFh          ; нужно для удобства использования  not с флагами
173                                                              ; 0FFh = 11111111b  = инверсия 00000000b
174
175044A                   new_int9h proc    far
176044A  56 50 53 51 52 06 1E       push SI AX      BX CX DX ES DS
                  ; сохраняем значения всех,        изменяемых регистров+
177                       в стеке
1780451  55                  push BP ; /////////
1790452  0E                  push CS                    ;
синхронизируем CS     и DS
1800453  1F                   pop DS
181
1820454  9C                   pushf
1830455  2E: FF 1E   03BDr                call dword       ptr
CS:[old_int9hOffset]    ; вызываем  стандартный обработчик прерывания
184045A  B8 0040             mov AX, 40h                ; 40h -
сегмент, где хранятся флаги состояния  +
185                       клавиатуры
186045D  8E C0               mov ES, AX
187045F  26: 8B 1E   001C    mov BX, ES:[1Ch]                ;
адрес хвоста
1880464  83 EB 02            sub BX, 2h                 ;
сместимся назад к последнему введённому        +
189                       символу
1900467  83 FB 1E            cmp BX, 1Eh                ; не
вышли       ли мы за пределы буфера?
191046A  73 03                jae _go
192046C  BB 003C             mov BX, 3Ch                ; хвост
вышел за пределы буфера: значит,     +
193                       последний
194                                                      ; введённый
символ       находится в конце буфера
195046F                   _go:
196046F  26: 8B 17           mov DX, ES:[BX]                ; в
DX 0 введённый  символ
197
```

```
1980472                 _test_Fx:                       ; проверка
    F8-F2
1990472                 _F8:
2000472  80 FE 42              cmp DH, 42h             ; F8(?)
2010475  75 0C                 jne _F9
2020477  F6 16 0116r           not signaturePrintingEnabled    ; Флаг
печати ФИО
203047B  26: 89 1E   001C      mov ES:[1Ch], BX        ;
блокировка вводасимвола
2040480  EB 7C 90              jmp _quit
2050483                 _F9:
2060483  80 FE 43              cmp DH, 43h             ; F9(?)
2070486  75 0F                 jne _F1
2080488  26: 89 1E   001C      mov ES:[1Ch], BX        ;
блокировка вводасимвола
209048D  F6 16 039Br           not cursiveEnabled      ;
Установка или сброс курсива
2100491  E8 0073               call toggleCursive      ;
перевод символа в курсив      и обратно
211                                                    ; в зависимости
от      флага cursiveEnabled
2120494  EB 68 90              jmp _quit
2130497                 _F1:
2140497  80 FE 3B              cmp DH, 3Bh             ; F1(?)
215049A  75 0C                 jne _F2
216049C  F6 16 0115r           not translateEnabled    ; Включение
перевода символов
21704A0  26: 89 1E   001C      mov ES:[1Ch], BX        ;
блокировка вводасимвола
21804A5  EB 57 90              jmp _quit
21904A8                 _F2:
22004A8  80 FE 3C              cmp DH, 3Ch             ; F2 (?)
22104AB  75 0C                 jne _translateOrIgnore
22204AD  F6 16 0109r           not ignoreEnabled       ;
Включение блокировки символоа
22304B1  26: 89 1E   001C      mov ES:[1Ch], BX        ;
блокировка вводасимвола
22404B6  EB 46 90              jmp _quit
225
22604B9                 _translateOrIgnore:             ; просто
выводим набранный      символ на экран
227
22804B9  80 3E 0109r FF        cmp ignoreEnabled,   true       ;
включен ли режим       блокировки ввода?
```

```
22904BE  75 1B                  jne _checkTranslate
230
23104C0  BE 0000                mov SI, 0                       ; да, включен
23204C3  8A 0E 0108r            mov CL, ignoredLength           ; количество игнорируемых символов
233
23404C7                 _checkIgnored:
23504C7  3A 94 0103r            cmp DL, ignoredChars[SI]        ; проверяем, присутствует ли текущий символ в  +
236                     списке игнорируемых
23704CB  74 06                  je  _block
23804CD  46                 inc SI
23904CE  E2 F7                  loop _checkIgnored              ; зацикливаем ignoredLength раз
24004D0  EB 09 90               jmp _checkTranslate
241
242                     ; блокируем
24304D3                 _block:
24404D3  26: 89 1E    001C      mov ES:[1Ch], BX                ; блокировка вводасимвола
245                     ; если по варианту    нужно не блокировать ввод символа,
246                     ; а заменять одни символы другими,
      замените строку   выше строкой
247                     ;     mov ES:[BX], AX
248                     ; на месте        AX может быть '*' для замены всех символов множества ignoredChars на          +
249                     звёздочки
250                     ; или, для        перевода одних символов      в другие - завести массив
251                     ; replaceWith DB '...', где перечислить символы, на которые пойдёт  замена
252                     ; и раскомментировать строки ниже:
253                     ;     xor AX, AX
254                     ;     mov AL, replaceWith[SI]
255                     ;     mov ES:[BX], AX                 ; замена символа
25604D8  EB 24 90                jmp _quit
```

```
   257
   25804DB                    _checkTranslate:
   25904DB  80 3E 0115r FF          cmp translateEnabled, true          ;
включен ли режим       перевода?
   26004E0  75 1C                jne _quit
   261
   26204E2  BE 0000                 mov SI, 0                           ; да,
включен
   26304E5  8A 0E 0114r             mov CL, translateLength             ;
кол-во символов для перевода
   264
   26504E9                    _checkTranslateLoop:
   26604E9  3A 94 010Ar             cmp DL, translateFrom[SI]           ;
присутствует ли текущий символ в списке для  +
   267                     перевода?
   26804ED  74 06                   je  _translate
   26904EF  46                   inc SI
   27004F0  E2 F7                loop _checkTranslateLoop            ;
продолжаем, пока       не закончим проверять каждый+
   271                     символ
   27204F2  EB 0A 90                jmp _quit
   273
   27404F5                    _translate:
   27504F5  33 C0                xor AX, AX                          ;
переводим
   27604F7  8A 84 010Fr             mov AL, translateTo[SI]
   27704FB  26: 89 07               mov ES:[BX], AX                     ;
замена символа
   278
   27904FE                    _quit:
   28004FE  5D                   pop BP  ; //////////
   28104FF  1F 07 5A 59 5B 58       5E       pop DS ES DX CX BX       AX SI
                ; восстанавливаем все регистры
   2820506  CF                   iret
   2830507                    new_int9h endp
   284
   2850507                    toggleCursive proc
```

```
   2860507  06 50                push ES AX                      ;
сохраняем регистры
   2870509  0E                  push CS
   288050A  07                  pop ES
   289
   290050B  80 3E 039Br FF          cmp cursiveEnabled, true            ;
если флаг равен true,
   2910510  75 30               jne _restoreSymbol              ;
выполняем замену      символа     на курсивный вариант,
   292                                                          ;
предварительно сохраняя старый символ в          +
   293                  savedSymbol
   294
   2950512  E8 004E                 call saveFont
   2960515  8A 0E 03ACr             mov CL, charToCursiveIndex
   2970519                      _shiftTable:
   2980519  83 C5 10                add BP, 16                      ;
получаем    в BP таблицу всех символов. адрес   +
   299                  указывает на символ 0
   300                                                          ; поэтому
нужно совершить сдвиг 16*X - где  X - +
   301                  код символа
   302051C  E2 FB                   loop _shiftTable
   303
   304051E  1E                  push DS                         ; при savefont
смещается регистр ES
   305051F  58                  pop AX                          ; поэтому
приходится делать такие махинации,   +
   306                  чтобы
   3070520  06                  push ES                         ; записать
        полученный элемент в savedSymbol
   3080521  1F                  pop DS
   3090522  50                  push AX                         ; DS -> AX, ES
-> DS, AX -> ES => ES и DS      +
   310                  поменялись     местами
   3110523  07                  pop ES                          ; + сохранение
старого значения DS     в AX
   3120524  50                  push AX
   313
```

```
   3140525  8B F5                mov SI, BP
   3150527  BF 03ADr                lea DI, savedSymbol             ;
сохраняем в переменную savedSymbol таблицу   +
   316                        нужного символа
   317
   318052A  B9 0010                 mov CX, 16                      ;
movsb из     DS:SI в       ES:DI
   319
   320052D  F3> A4                  rep movsb                       ;
исходные    позиции      сегментов возвращены
   321052F  1F                  pop DS                      ;
восстановление DS
   322
   3230530  B9 0001                 mov CX, 1                       ;
заменим написание символа на курсив
   3240533  B6 00                mov DH, 0
   3250535  8A 16 03ACr             mov DL, charToCursiveIndex
   3260539  BD 039Cr                lea BP, cursiveSymbol
   327053C  E8 0015                 call changeFont
   328053F  EB 10 90                jmp _exitToggleCursive
   329
   3300542                    _restoreSymbol:
   3310542  B9 0001                 mov CX, 1                       ;
если флаг равен 0, заменяем курсивный символ +
   332                        на старый вариант
   3330545  B6 00                mov DH, 0
   3340547  8A 16 03ACr             mov DL, charToCursiveIndex
   335054B  BD 03ADr                lea BP, savedSymbol
   336054E  E8 0003                 call changeFont
   337
   3380551                    _exitToggleCursive:
   3390551  58                   pop AX
   3400552  07                   pop ES
   3410553  C3                   ret
   3420554                    toggleCursive endp
```

```
343
3440554                 changeFont proc
3450554  50 53 52                  push AX BX    DX
3460557  B8 1100                   mov AX, 1100h
347055A  BB 1000                   mov BX, 1000h
348055D  CD 10                     int 10h
349055F  5A 5B 58                  pop DX BX AX
3500562  C3                    ret
3510563                 changeFont endp
352
3530563                 saveFont proc
3540563  50 53 52                  push AX BX    DX
3550566  B8 1130                   mov AX, 1130h
3560569  BB 0600                   mov BX, 0600h
357056C  CD 10                     int 10h
358056E  5B 58 5A                  pop BX AX DX
3590571  C3                    ret
3600572                 saveFont endp
361
362                     ; обработчик прерывания int 2Fh
363                     ; служит для:
364                     ;  1) проверки      факта присутствия TSR в
    памяти (при AH=0FFh, AL=0)
365                     ;     будет возвращён AH='i' в случае,       если
TSR уже загружен
366                     ;  2) выгрузки      TSR из памяти (при AH=0FFh,
AL=1)
3670572                 new_int2Fh proc
3680572  80 FC FF                  cmp AH, 0FFh                ; наша
процедура?
3690575  75 0B                 jne _2Fh_default                ; нет - на
    стандартный обработчик
3700577  3C 00                 cmp AL, 0                       ;
подпроцедура проверки, загружен ли резидент в+
371                         память?
3720579  74 0C                 je  _alreadyInstalled2Fh
373057B  3C 01                     cmp AL, 1                       ;
подпроцедура выгрузки из    памяти?
374057D  74 0B                     je  _uninstall
```

```
375057F  EB 01 90                jmp _2Fh_default
376
3770582                    _2Fh_default:
3780582  2E: FF 2E   03C5r      jmp dword ptr CS:[old_int2FhOffset]    ;
вызов стандартного обработчика
379
3800587                    _alreadyInstalled2Fh:
3810587  B4 69                   mov AH, 'i'                    ; пусть AH
    = 'i', если резидент уже загружен в +
382                          память
3830589  CF                      iret                           ; конечно,
    вместо 'i' может быть любое значение
384
385058A                    _uninstall:                    ; подпроцедура
выгрузки из      памяти
386058A  1E 06 52 53         push DS ES     DX BX
387058E  33 DB                   xor BX, BX
388
3890590  0E                      push CS                        ; CS = ES,
    для доступа к переменным
3900591  07                      pop ES
391
3920592  B8 2509                   mov AX, 2509h
3930595  26: 8B 16   03BDr            mov DX, ES:old_int9hOffset
        ; возвращаем вектор прерывания 09h   на место
394059A  26: 8E 1E   03BFr            mov DS, ES:old_int9hSegment
395059F  CD 21                   int 21h
396
39705A1  B8 251C                 mov AX, 251Ch
39805A4  26: 8B 16   03C1r       mov DX, ES:old_int1ChOffset        ;
возвращаем вектор прерывания 1Ch      на место
39905A9  26: 8E 1E   03C3r       mov DS, ES:old_int1ChSegment
```

```
   40005AE  CD 21                  int 21h
   401
   40205B0  B8 252F                mov AX, 252Fh
   40305B3  26: 8B 16  03C5r       mov DX, ES:old_int2FhOffset     ;
возвращаем вектор прерывания 2Fh    на место
   40405B8  26: 8E 1E  03C7r       mov DS, ES:old_int2FhSegment
   40505BD  CD 21                  int 21h
   406
   40705BF  2E: 8E 06  002C        mov ES, CS:2Ch                  ;
загрузим    в ES адрес окружения
   40805C4  B4 49                  mov AH, 49h                     ;
выгрузим    из памяти окружение
   40905C6  CD 21                  int 21h
   41005C8  72 0B                  jc  _notRemove
   411
   41205CA  0E                 push CS
   41305CB  07                 pop ES                      ; в ES - адрес
резидентной     программы
   41405CC  B4 49                  mov AH, 49h                     ;
выгрузим    из памяти резидент
   41505CE  CD 21                  int 21h
   416
   41705D0  72 03              jc  _notRemove
   41805D2  EB 15 90               jmp _unloaded
   419
   42005D5                 _notRemove:                     ; не
удалось выполнить выгрузку =>   вывод ошибки
   42105D5  B4 03                  mov AH, 03h                     ;
получаем    позицию     курсора
   42205D7  CD 10                  int 10h
   42305D9  BD 042Dr               lea BP, noRemoveMsg
   42405DC  B9 001D                mov CX, noRemoveMsg_length
   42505DF  B3 07                  mov BL, 0111b
   42605E1  B8 1301                mov AX, 1301h
   42705E4  CD 10                  int 10h
   42805E6  EB 12 90               jmp _2Fh_exit
   429
   43005E9                 _unloaded:                  ; выгрузка
      прошла успешно => вывод  сообщения
```

```
43105E9  B4 03                mov AH, 03h                ; получаем
    позицию    курсора
43205EB  CD 10                    int 10h
43305ED  BD 0411r                 lea BP, removedMsg
43405F0  B9 001C                  mov CX, removedMsg_length
43505F3  B3 07                mov BL, 0111b
43605F5  B8 1301                  mov AX, 1301h
43705F8  CD 10                    int 10h
438
43905FA                       _2Fh_exit:
44005FA  5B 5A 07 1F              pop BX DX ES DS
44105FE  CF                    iret
44205FF                       new_int2Fh endp
443
444                           ; обработчик прерывания int 1Ch
445                           ; вызывается каждые 55       мс
44605FF                       new_int1Ch proc far
44705FF  50                    push AX
4480600  0E                    push CS
4490601  1F                    pop DS
450
4510602  9C                    pushf
4520603  2E: FF 1E   03C1r        call dword      ptr CS:[old_int1ChOffset]
; вызываем стандартный обработчик прерывания
453
4540608  80 3E 0116r FF          cmp signaturePrintingEnabled, true
; если нажата управляющая клавиша (в данном   +
455                          случае F1)
456060D  75 1B                   jne _notToPrint
```

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 9
kr.asm

```
   457
   458060F  83 3E 0117r 5B              cmp counter, printDelay*1000/55 + 1    ;
если кол-во "тактов" равно printDelay секундам
   4590614  74 03                  je  _letsPrint
   460
   4610616  EB 0E 90                    jmp _dontPrint
   462
   4630619                        _letsPrint:
   4640619  F6 16 0116r              not signaturePrintingEnabled
   465061D  C7 06 0117r 0000            mov counter, 0
   4660623  E8 0016                     call printSignature                ;
выводим подпись на экран
   467
   4680626                        _dontPrint:
   4690626  FF 06 0117r                 inc counter                  ; увеличим
       значение счетчика на 1
   470
   471062A                        _notToPrint:
   472062A  58                       pop AX
   473062B  CF                       iret
   474062C                        new_int1Ch endp
   475
   476                            ; выводит одну    строку подписи
   477062C                        printSignatureLine proc
   478062C  52                       push DX
   479062D  8B 0E 0119r                 mov CX, signatureLineLength
   4800631  B3 07                    mov BL, 0111b                ; цвет
выводимого текста
   4810633  B8 1301                     mov AX, 1301h                          ;
AH = 13h    - номер       ф-ии, AL = 01h - перемещение+
   482                                курсора
   4830636  CD 10                       int 10h
   4840638  5A                       pop DX
   4850639  FE C6                       inc DH
   486063B  C3                       ret
   487063C                        printSignatureLine endp
   488
   489                            ; процедура вывода подписи
   490063C                        printSignature      proc
```

```
491063C  50 52 51 53 06 54 55+      push AX DX    CX BX ES SP BP SI DI
492      56 57
493
4940645  33 C0                 xor AX, AX                    ;
обнуляем    значения регистров
4950647  33 DB                      xor BX, BX
4960649  33 D2                 xor DX, DX
497
498064B  B4 03                      mov AH, 03h              ; чтение
текущей позиции курсора
499064D  CD 10                      int 10h
500064F  52                    push DX                  ; помещаем
     информацию о положении курсора в стек
501
5020650  BA 000F                    mov DX, 000Fh                 ;
NB! вверху: 000Fh, посередине: 090Fh, внизу: +
503                  130Fh
504
5050653                        _actualPrint:
5060653  B4 0F                   mov AH, 0Fh              ; чтение
текущего видеорежима. в BH - текущая  +
507                  страница
5080655  CD 10                      int 10h
509
5100657  0E                    push CS
5110658  07                    pop ES                   ; указываем ES
на CS
512
5130659  BD 01B7r                   lea BP, tableTop
```

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 10
kr.asm

```
   514065C  E8 FFCD                call printSignatureLine                 ;
выводим верх таблицы
   515065F  BD 011Br               lea BP, signatureLine1
   5160662  E8 FFC7                call printSignatureLine                 ;
выводим первую строку
   5170665  BD 014Fr               lea BP, signatureLine2
   5180668  E8 FFC1                call printSignatureLine                 ;
выводим вторую строку
   519066B  BD 0183r               lea BP, signatureLine3
   520066E  E8 FFBB                call printSignatureLine                 ;
выводим третью строку
   5210671  BD 01EBr               lea BP, tableBottom
   5220674  E8 FFB5                call printSignatureLine                 ;
выводим низ таблицы
   523
   5240677  33 DB                  xor BX, BX
   5250679  5A                  pop DX                          ;
восстанавливаем из стека      прежнее      положение   +
   526                        курсора
   527067A  B4 02                  mov AH, 02h                     ; меняем
положение  курсора      на первоначальное
   528067C  CD 10                  int 10h
   529
   530067E  5F 5E 5D 5C 07 5B       59+        pop DI SI BP SP ES   BX CX DX
AX
   531      5A 58
   5320687  C3                     ret
   5330688                      printSignature      endp
   534
   535                          ; Основная часть программы
   536                          ; 1) установка      видеорежима
   537                          ; 2) проверка,      запущен      ли резидент
   538                          ; 3) установка      вектора      прерываний
   5390688                      _initTSR:
   5400688  B4 03                  mov AH, 03h
   541068A  CD 10                      int 10h
   542068C  52                     push DX
   543068D  B4 00                      mov AH, 00h                     ;
установка видеорежима
```

```
   544068F  B0 83                  mov AL, 83h
   5450691  CD 10                      int 10h
   5460693  5A                    pop DX
   5470694  B4 02                  mov AH, 02h
   5480696  CD 10                      int 10h
   549
   5500698  E8 007B                    call commandParamsParser            ;
читаем аргументы        командной строки
   551069B  80 3E 039Ar 02             cmp commandLineResult, 2            ;
если результат =  2, значит была выведена          +
   552                      справка
   55306A0  75 03                  jne _shouldContinue             ;
соответственно, никаких других действий          +
   554                      делать не нужно
   55506A2  EB 70 90                   jmp _exit
   55606A5                      _shouldContinue:
   55706A5  B4 FF                      mov     AH, 0FFh
   55806A7  B0 00                      mov     AL, 0
   55906A9  CD 2F                      int  2Fh
   56006AB  80 FC 69                   cmp     AH, 'i'
; проверка того, загружена ли уже программа
   56106AE  74 50                      je _remove
   562
   56306B0  B8 3509                    mov AX, 3509h                       ;
получить    в ES:BX     прерывания 09h
   56406B3  CD 21                      int 21h
   56506B5  2E: 89 1E   03BDr              mov word ptr CS:old_int9hOffset,
BX     ; обработчик прерывания 09h
   56606BA  2E: 8C 06   03BFr              mov word ptr
CS:old_int9hSegment, ES
   56706BF  B8 2509                    mov AX, 2509h                       ;
установим векторна прерывание 09h
   56806C2  BA 044Ar                   mov DX, offset new_int9h
   56906C5  CD 21                      int 21h
   570
```

```
   57106C7  B8 351C                mov AX, 351Ch                   ;
получить    в ES:BX     прерывания 1Ch
   57206CA  CD 21                  int 21h
   57306CC  2E: 89 1E   03C1r      mov word ptr CS:old_int1ChOffset, BX
; обработчик прерывания 1Ch
   57406D1  2E: 8C 06   03C3r      mov word ptr CS:old_int1ChSegment,
        ES
   57506D6  B8 251C                mov AX, 251Ch                   ;
установим векторна прерывание 1Ch
   57606D9  BA 05FFr               mov DX, offset new_int1Ch
   57706DC  CD 21                  int 21h
   578
   57906DE  B8 352F                mov AX, 352Fh                   ;
получить    в ES:BX     прерывания 2Fh
   58006E1  CD 21                  int 21h
   58106E3  2E: 89 1E   03C5r      mov word ptr CS:old_int2FhOffset, BX
; обработчик прерывания 2Fh
   58206E8  2E: 8C 06   03C7r      mov word ptr CS:old_int2FhSegment,
        ES
   58306ED  B8 252F                mov AX, 252Fh                   ;
установим векторна прерывание 2Fh
   58406F0  BA 0572r               mov DX, offset new_int2Fh
   58506F3  CD 21                  int 21h
   586
   58706F5  BB 03C9r               lea BX, installedMsg            ;
выводим сообщение, что всё ОК
   58806F8  E8 0045                call printStr
   589
   59006FB  BA 0688r               mov DX, offset _initTSR         ;
остаемся    в памяти и выходим из основной части
   59106FE  CD 27                  int 27h
   592
   5930700                    _remove:                             ; выгрузка
        из памяти
   5940700  06                   push ES
   5950701  A1 002C                mov AX, DS:[2Ch]                ;
PSP
   5960704  8E C0                mov ES, AX
```

```
   5970706  B4 49                mov AH, 49h                ; хватит памяти
чтоб остаться резидентом?
   5980708  CD 21                    int 21h
   599070A  07                  pop ES
   600
   601070B  B4 FF                    mov AH, 0FFh
   602070D  B0 01                    mov AL, 1
   603070F  CD 2F                    int 2Fh
   6040711  EB 01 90                 jmp _exit
   6050714                      _exit:                     ; выход
   6060714  CD 20                    int 20h
   607
   608                          ; парсер аргментов командной строки. выводит
справку.
   609                          ; устанавливает флаг commandLineResult:
   610                          ; 0 = всё ОК; 1 = нужна выгрузка; 2 = была
выведена справка, не нужно загружать резидент
   6110716                      commandParamsParser proc
   6120716  0E                    push CS
   6130717  07                    pop ES
   614
   6150718  BE 0080                    mov SI, 80h              ; SI =
смещение командной строки
   616071B  AC                  lodsb                      ; получим кол-
во символов
   617071C  0A C0                    or  AL, AL                 ;
если 0 символов введено,
   618071E  74 1A                    jz  _paramParsingEnd       ; то
все в  порядке
   619
   6200720                      _nextChar:
   6210720  46                    inc SI                   ; теперь SI
указывает на первый символ строки
   622
   6230721  80 3C 00                 cmp [SI], BYTE ptr    0
   6240724  74 14                 je  _paramParsingEnd
   625
   6260726  AD                  lodsw                      ; получаем
       два символа
   6270727  3D 3F2F                  cmp AX, '?/'
```

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 12
kr.asm

```
628072A  74 03                  je  _displayHelp
629
630072C  EB 0C 90                       jmp _paramParsingEnd
631072F                         _displayHelp:
632072F  BB 021Fr                       lea BX, helpMsg                 ;
выводим справку
6330732  E8 000B                        call printStr
6340735  C6 06 039Ar 02                 mov commandLineResult, 2        ;
флаг того, что резидент загружать не надо
635
636073A                         _paramParsingEnd:
637073A  C3                        ret
638073B                        commandParamsParser endp
639
640                             ; отображает символ из        AL
641073B                         printChar proc
642073B  B4 0E                          mov AH, 0EH
643073D  CD 10                          int 010H
644073F  C3                        ret
6450740                         printChar endp
646
647                             ; отображает нуль-терминированную строку из
[BX]
6480740                         printStr proc
6490740  52 50                     push DX AX
6500742  8B 07                     mov AX, [BX]
6510744                         _printStrLoop:
6520744  3C 00                     cmp AL, 0
6530746  74 08                     je  _printStrEnd
6540748  E8 FFF0                        call printChar
655074B  43                        inc BX
656074C  8B 07                          mov AX, [BX]
657074E  EB F4                          jmp _printStrLoop
6580750                         _printStrEnd:
6590750  58 5A                     pop AX DX
6600752  C3                        ret
6610753                         printStr endp
662
6630753                    code ends
664                        end _start
```

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 13
Symbol Table

Symbol Name	Type	Value
??DATE	Text	"04/13/25"
??FILENAME	Text	"kr "
??TIME	Text	"22:50:33"
??VERSION	Number	030A
@CPU	Text	0101H
@CURSEG	Text	CODE
@FILENAME	Text	KR
@WORDSIZE	Text	2
ALREADYINSTALLEDMSG	Byte	CODE:03DC
CHANGEFONT	Near	CODE:0554
CHARTOCURSIVEINDEX	Byte	CODE:03AC
COMMANDLINERESULT	Byte	CODE:039A
COMMANDPARAMSPARSER	Near	CODE:0716
COUNTER	Word	CODE:0117
CURSIVEENABLED	Byte	CODE:039B
CURSIVESYMBOL	Byte	CODE:039C
HELPMSG	Byte	CODE:021F
HELPMSGLENGTH	Number	017B
IGNOREDCHARS	Byte	CODE:0103
IGNOREDLENGTH	Byte	CODE:0108
IGNOREENABLED	Byte	CODE:0109
INSTALLEDMSG	Byte	CODE:03C9
NEW_INT1CH	Far	CODE:05FF
NEW_INT2FH	Near	CODE:0572
NEW_INT9H	Far	CODE:044A
NOREMOVEMSG	Byte	CODE:042D
NOREMOVEMSG_LENGTH	Number	001D
NOTINSTALLEDMSG	Byte	CODE:03F7
OLD_INT1CHOFFSET	Word	CODE:03C1
OLD_INT1CHSEGMENT	Word	CODE:03C3
OLD_INT2FHOFFSET	Word	CODE:03C5
OLD_INT2FHSEGMENT	Word	CODE:03C7
OLD_INT9HOFFSET	Word	CODE:03BD
OLD_INT9HSEGMENT	Word	CODE:03BF
PRINTCHAR	Near	CODE:073B
PRINTDELAY	Number	0005

PRINTSIGNATURE Near CODE:063C
PRINTSIGNATURELINE Near CODE:062C
PRINTSTR Near CODE:0740
REMOVEDMSG Byte CODE:0411
REMOVEDMSG_LENGTH Number 001C
SAVEDSYMBOL Byte CODE:03AD
SAVEFONT Near CODE:0563
SIGNATURELINE1 Byte CODE:011B
SIGNATURELINE2 Byte CODE:014F
SIGNATURELINE3 Byte CODE:0183
SIGNATURELINELENGTH Word CODE:0119
SIGNATUREPRINTINGENABLED Byte CODE:0116
TABLEBOTTOM Byte CODE:01EB
TABLETOP Byte CODE:01B7
TOGGLECURSIVE Near CODE:0507
TRANSLATEENABLED Byte CODE:0115
TRANSLATEFROM Byte CODE:010A
TRANSLATELENGTH Byte CODE:0114

Turbo Assembler Version 3.1 04/13/25 22:50:33 Page 14
Symbol Table

TRANSLATETO Byte CODE:010F
TRUE Number 00FF
_2FH_DEFAULT Near CODE:0582
_2FH_EXIT Near CODE:05FA
_ACTUALPRINT Near CODE:0653
_ALREADYINSTALLED2FH Near CODE:0587
_BLOCK Near CODE:04D3
_CHECKIGNORED Near CODE:04C7
_CHECKTRANSLATE Near CODE:04DB
_CHECKTRANSLATELOOP Near CODE:04E9
_DISPLAYHELP Near CODE:072F
_DONTPRINT Near CODE:0626
_EXIT Near CODE:0714
_EXITTOGGLECURSIVE Near CODE:0551
_F1 Near CODE:0497
_F2 Near CODE:04A8
_F8 Near CODE:0472
_F9 Near CODE:0483
_GO Near CODE:046F
_INITTSR Near CODE:0688
_LETSPRINT Near CODE:0619
_NEXTCHAR Near CODE:0720
_NOTREMOVE Near CODE:05D5
_NOTTOPRINT Near CODE:062A
_PARAMPARSINGEND Near CODE:073A
_PRINTSTREND Near CODE:0750
_PRINTSTRLOOP Near CODE:0744
_QUIT Near CODE:04FE
_REMOVE Near CODE:0700
_RESTORESYMBOL Near CODE:0542
_SHIFTTABLE Near CODE:0519
_SHOULDCONTINUE Near CODE:06A5
_START Near CODE:0100
_TEST_FX Near CODE:0472
_TRANSLATE Near CODE:04F5
_TRANSLATEORIGNORE Near CODE:04B9
_UNINSTALL Near CODE:058A
_UNLOADED Near CODE:05E9

Groups & Segments Bit Size Align Combine Class

CODE 16 0753 Para none CODE