# ; 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 ; на начало программы

; данные

; данные программы резидента

\_start:  
  
jmp \_initTSR ; на начало программы  
;@Если по варианту нужно, чтобы происходила замена символов  
; данные  
ignoredChars DB 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя' ; список игнорируемых символов  
replaceWith DB 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЬЭЮЯ' ; список заменяемых символов  
ignoredLength equ $-ignoredChars ; длина строки ignoredChars  
ignoreEnabled DB 0 ; флаг функции игнорирования ввода  
translateFrom DB 'KVYJG' ; символы для замены  
translateTo DB 'ЛМНОП' ; символы на которые будет идти замена  
translateLength equ $-translateTo ; длина строки trasnlateFrom  
translateEnabled DB 0 ; флаг функции перевода  
  
signaturePrintingEnabled DB 0 ; флаг функции вывода информации об авторе  
cursiveEnabled DB 0 ; флаг перевода символа в курсив

; Текст справки

helpMsg DB '>tsr.com [/?]', 10, 13

DB ' [/?] - вывод данной справки', 10, 13

DB ' F3 - вывод ФИО и группы по таймеру по центру экрана', 10, 13

DB ' F4 - включение и отключения курсивного вывода русского символа В', 10, 13

DB ' F5 - включение и отключение частичной русификации клавиатуры(]sm".z ЛМНОП)', 10, 13

DB ' F6 - включение и отключение режима блокировки цифр', 10, 13

# ; Курсивный образ

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

DB 00111111b

DB 00100001b

DB 00100010b

DB 00100100b

DB 01111000b

DB 01000000b

DB 01000000b

DB 01110000b

DB 01001000b

DB 01001000b

DB 01000100b

DB 01000100b

DB 01000010b

DB 10000010b

DB 10000100b

DB 11111000b; Код символа для курсива

charToCursiveIndex DB 'В' ;@ символ для замены

# ; Сохраненный образ символа

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

true equ 0FFh ; константа истинности

; 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

notLoadTSR DB 0 ; 1 - не загружать

counter DW 0

printDelay equ 2 ;@ задержка перед выводом "подписи" в секундах

signatureLine1 DB 179, ‘Кожиев Таймураз Казбекович ', 179

Line1\_length equ $-signatureLine1

signatureLine2 DB 179, 'ИУ5-45 ', 179

Line2\_length equ $-signatureLine2

signatureLine3 DB 179, 'Вариант #7 ', 179

Line3\_length

# ; сообщения части резидента

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

# ; Новый 09h

;новый обработчик

new\_int9h proc far

; сохраняем значения всех, изменяемых регистров в стэке

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 нажатия клавиш F3-F6

;проверка F1-F4

\_test\_Fx:

sub AL, 58 ; в AL теперь номер функциональной клавиши

# ;F1

\_F1:

cmp AL, 3 ; F1

jne \_F2

not signaturePrintingEnabled

call changeFx

jmp \_translate\_or\_ignore

# ;F2

\_F2:

cmp AL, 4 ; 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

\_F3:

cmp AL, 5 ; F3

jne \_F4

not translateEnabled

call changeFx

jmp \_translate\_or\_ignore

# ;F4

\_F4:

cmp AL, 6 ; F4

jne \_translate\_or\_ignore

not ignoreEnabled

call changeFx

jmp \_translate\_or\_ignore

;игнорирование и перевод

# ; Вывод символа без изменения

\_translate\_or\_ignore:

pushf

call dword ptr CS:[old\_int9hOffset] ; вызываем стандартный обработчик прерывания

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

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

; 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

# ; Работа с буфером клавиатуры

\_block:

mov ES:[1Ch], BX ;блокировка ввода символа

;@mov ES:[BX], AX

;@xor AX, AX

;@mov AL, replaceWith[SI]

;@mov ES:[BX], AX ; замена символа

jmp \_quit

\_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

# ; перевод символов (translateFrom-> translateTo)

\_translate:

xor AX, AX ; переводим

mov AL, translateTo[SI]

mov ES:[BX], AX ; замена символа в буфере клавиатуры

\_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 ; 0- режим проверки, загружен ли резидент в память?

je \_alreadyInstalled2Fh

cmp AL, 1 ; 1- режим выгрузки и вызов процедуры 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: ; подпроцедура выгрузки из памяти

push DS ES DX BX

xor BX, BX

push CS ; CS = ES, **для доступа к перемен**ным

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

# ; Выгрузка окружения процесса из памяти

mov ES, CS:2Ch ; загрузим в ES адрес окружения

mov AH, 49h ; выгрузим из памяти окружение

int 21h

jc \_notRemove

# ; Освобождение памяти, занятой под резидент

# ; Выгрузка процедур резидента из памяти

push CS

pop ES ; в ES - адрес резидентной программы

mov AH, 49h ; выгрузим из памяти резидент

int 21h

jc \_notRemove ; Проверка флага освобождения памяти.

; **флаг CF установлен, если ошибка**!!!

jmp \_unloaded

# ; не удалось выполнить выгрузку

\_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

jmp \_dontPrint

# ; Выводим сообщение по времени

\_letsPrint:

not signaturePrintingEnabled

mov counter, 0

call printSignature ; выводим подпись (сообщение по времени) 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

# ; Часть ИНИЦИАЛИЗАЦИИ

; Основная часть программы

; 1) установка видеорежима

; 2) проверка, запущен ли резидент

; 3) установка вектора прерываний

\_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 ###

# ; Проверка загрузки резидента (обращение к 2FH)

; @@@ 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 @@@

# ; Запоминание Адресов старых обработчиков

# ; 9H

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

# ; 1СH

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

# ; 2FH

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

mov DX, offset \_initTSR ; остаемся в памяти и выходим из основной части

# ; Остаться в памяти РЕЗИДЕНТОМ!

**; Прерывание TSR (27H)**

int 27h

\_remove: ; выгрузка из памяти области окружения

# ; Предварительное Освобождение области окружения из ОП !

; это осталось их обобщенной программы TSR.asm, а здесь не используется (!!)

; окружение выгружается в резиденте в процедуре выгрузки по 2Fh (смотри выше)

push ES

mov AX, DS:[2Ch] ; PSP область окружения в регистр

mov ES, AX

mov AH, 49h ; хватит памяти чтоб остаться резидентом?

int 21h

pop ES

mov AH, 0FFh

mov AL, 1 ; 1- режим проверки наличия в памяти

int 2Fh

jmp \_exit

# ; TSR Уже в памяти !

\_alreadyInstalled: ; резидент уже запущен

lea BX, alreadyInstalledMsg

call printStr

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 Процедура проверки параметров при запуске!

commandParamsParser proc

push CS

pop ES

mov SI, 80h ; SI = смещение командной строки

lodsb ; получим кол-во символов в AL

or AL, AL ; если 0 символов введено, число символов параметров

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, '?/' ; проверка "/?" – так как в памяти байты наоборот!!!!!!

je \_displayHelp

; @@@ следует раскомментировать, если нужно выгружать по аргументу /u @@@

cmp AX, 'u/' ; проверка "/u" – так как в памяти байты наоборот!!!!!!

je \_finishTSR

cmp AX, 'U/' ; проверка "/U" – так как в памяти байты наоборот!!!!!!

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

; ВЫЗРУЗКА резидентв через 2FH

\_paramParsingEndWithUnload:

mov AH, 0FFh

mov AL, 0 ; 0 - **Режим выгрузки**

int **2Fh**

cmp AH, 'i' ; проверка того, загружена ли уже программа 'i' – в памяти!!

jne \_paramParsingEnd

mov commandLineResult, 1

; ### конец блока ###

\_paramParsingEnd:

ret

commandParamsParser endp

; Процедура печати одного символа через видеосервис 10Н

; отображает символ из AL

# ; printChar Печать символа

printChar proc

mov AH, 0EH

int 010H

ret

printChar endp

; Печать строки через видеосервис 10Н (нуль в конце строки)

# ; printStr Печать строки в резиденте (посимвольно)

; отображает нуль-терминированную строку из [BX]

printStr proc

push DX AX

mov AX, [BX]

; цикл печати строки

\_printStrLoop:

cmp AL, 0 ; проверка нуля в строке

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

# ; toggleCursive Переключение на курсив (процедура)

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,

; Если нет то восстанавливаем символ из сохраненного ранее

; предварительно сохраняя старый символ в savedSymbol

call saveFont ; получаем в BP таблицу всех символов. адрес указывает на символ 0

mov CL, charToCursiveIndex ; код символа для курсива

\_shiftTable:

add BP, 16 ; Двигаемся по таблице со сдвигом 16

; поэтому нужно совершить сдвиг 16\*X - где X - код символа

loop \_shiftTable

push DS ; пpи savefont смещается pегистp ES

pop AX ; поэтомy пpиходится 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 ; сохpаняем в пеpеменнyю savedSymbol таблицу нужного символа

mov CX, 16 ; movsb из DS:SI в ES:DI

rep movsb ; исходные позиции сегментов возвращены перезапись в

savedSymbol

pop DS ; восстановление DS

mov CX, 1 ; заменим написание символа на курсив

mov DH, 0

mov DL, charToCursiveIndex ; кодировка курсивного символа

lea BP, cursiveSymbol ; образ курсива

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 ; кодировка сохраненного символа

call changeFont

\_exitToggleCursive:

pop AX

pop ES

ret

toggleCursive endp

; функция смены начертания символа (курсив/нормальное)

;

; входные данные:

; 1) **DL** = номер символа для замены (charToCursiveIndex)

; 2) **CX** = количество символов заменяемых изображений символов (СХ =1 - один символ)

; (начиная с символа указанного в DX)

; 3) **ES:BP** = адрес таблицы (получаем подфункцией 1130Н видеочервиса)

;

; 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

; замена изображения одного символа через видео сервис (10Н)

# ; changeFont 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:

; 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:

; 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)

# ; saveFont сохранение образа для символа шрифта

saveFont proc

push AX BX DX

mov AX, 1130h

mov BX, 0600h

int 10h ; Получим (см. спр.) ES:BP – адрес таблицы описания буквы шрифта

pop BX AX DX

ret

saveFont endp

# ; printSignatureLine 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

# ; printSignature процедура вывода подписи (сообщения)

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 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

code ends

end \_start