Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 4

по дисциплине ПнаЯА

Вариант 1

Выполнил студент гр. 150501: Ковальчук Д.И.

Проверил: Туровец Н.О.

Минск 2022

Тема работы: Создание видеоигры

Цель работы: Ознакомиться в рамках создания видеоигры с обработкой нажатий кнопок клавиатуры, рассмотреть прямой доступ к видеопамяти с целью формирования игрового поля и информации для пользователя.

Теоретические сведения:

1. Видеорежимы и управление ими.

2. Прямой доступ к видеопамяти.

3. Управление курсором и вывод данных на экран прерыванием BIOS. 4. Обработка нажатий кнопок клавиатуры прерыванием BIOS.

Буфер клавиатуры.

5. Доступ к системным часам.

Задание:

Игра «Змейка».

Цель: ползаем, едим случайно появляющиеся в свободных местах яблочки и растем от этого в длину, выход за границы экрана означает возврат в поле с противоположной стороны (круглый мир).

Окончание: проигрыш – укус самого себя, выигрыш – нет.

Информация: счет.

Усложнение: смена уровня (увеличение скорости).

Код программы:

clearScreen MACRO ;

push ax ; save znachenie ax

mov ax, 0003h ; 00 - ustanovit videorezhim, clear the screen. 03h - rezhim 80x25

int 10h ; call for prerivaniye to make command

pop ax ; vosstanavit' znachenie ax

ENDM ;

.model small

.stack 100h

.data

;key bindings (configuration)

KUpSpeed equ 48h ; Up key

KDownSpeed equ 50h ; Down key

KMoveUp equ 11h ; W key

KMoveDown equ 1Fh ; S key

KMoveLeft equ 1Eh ; A key

KMoveRight equ 20h ; D key

KExit equ 01h ; ESC key

;

xSize equ 80 ; width of console

ySize equ 25 ; height of console

xField equ 50 ; width of field

yField equ 21 ; height of field

oneMemoBlock equ 2 ; size of one kletka of console

scoreSize equ 4 ; size of score table

;

videoStart dw 0B800h ; smeschenie videobufera

dataStart dw 0000h ; byte with code of symbol of left high corner

;

space equ 0020h ; emoty block with black background

snakeBodySymbol equ 0A40h ; symbol of snakes body

appleSymbol equ 0B0Fh ; symbol of apple

VWallSymbol equ 0FBAh ; symbol of vertical wall

HWallSymbol equ 0FCDh ; symbol of horizontal wall

BWallSymbol equ 4020h ;

VWallSpecialSymbol equ 0FCCh ; symbol of peresecheniye walls

fieldSpacingBad equ space, VWallSymbol, xField dup(space)

fieldSpacing equ fieldSpacingBad, VWallSymbol

rbSym equ 077DCh ; white block with white background

rbSpc equ 04F20h ; probel with red background and white sumbol colour

ylSym equ 06FDCh ; white block with yellow background

ylSpc equ 06F20h ; probel with yellow background

grSym equ 02FDBh ; white block with green backgroundм

grSpc equ 02F20h ; probel with green background

screen dw xSize dup(space)

dw space, 0FC9h, xField dup(HWallSymbol), 0FCBh, xSize - xField - 5 dup(HWallSymbol), 0FBBh, space

firstBl dw fieldSpacing, xSize - xField - 5 dup(rbSpc), VWallSymbol, space

dw fieldSpacing, rbSpc, 4 dup(rbSym), 15 dup(rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space

dw fieldSpacing, rbSpc, rbSym, 5 dup(rbSpc), 3 dup(rbSym), 2 dup(rbSpc), 3 dup(rbSym), rbSpc, rbSym, 3 dup(rbSpc), rbSym, 2 dup(rbSpc), rbSym, rbSpc, VWallSymbol, space

dw fieldSpacing, rbSpc, 4 dup(rbSym), rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), 3 dup(rbSym, rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space

dw fieldSpacing, 4 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, 4 dup(rbSym), rbSpc, 2 dup(rbSym), 2 dup(rbSpc), rbSym, 4 dup(rbSpc), VWallSymbol, space

dw fieldSpacing, rbSpc, 4 dup(rbSym), rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), 3 dup(rbSym, rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space

dw fieldSpacing, xSize - xField - 5 dup(rbSpc), VWallSymbol, space

delim1 dw fieldSpacingBad, 0FCCh, xSize - xField - 5 dup(HWallSymbol), 0FB9h, space

secondF dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space

dw fieldSpacing, ylSpc, 06F53h, 06F63h, 06F6Fh, 06F72h, 06F65h, 06F3Ah, ylSpc

score dw scoreSize dup(06F30h), xSize - xField - scoreSize - 13 dup(ylSpc), VWallSymbol, space

dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space

dw fieldSpacing, ylSpc, 06F53h, 06F70h, 2 dup(06F65h), 06F64h, 06F3Ah, ylSpc

speed dw 06F31h, 16 dup(ylSpc), VWallSymbol, space

dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space

delim2 dw fieldSpacingBad, 0FCCh, xSize - xField - 5 dup(HWallSymbol), 0FB9h, space

thirdF dw fieldSpacing, xSize - xField - 5 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F43h, 02F6Fh, 02F6Eh, 02F74h,02F72h, 02F6Fh, 02F6Ch,02F73h, 02F3Ah, 15 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F57h, grSpc, 02FC4h, grSpc, 02F55h, 02F70h, 02F18h, 17 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F53h, grSpc, 02FC4h, grSpc, 02F44h, 02F6Fh, 02F77h ,02F6Eh, 02F19h, 15 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F41h, grSpc, 02FC4h, grSpc, 02F4Ch, 02F65h, 02F66h ,02F74h, 02F1Bh, 15 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F44h, grSpc, 02FC4h, grSpc, 02F52h, 02F69h, 02F67h ,02F68h, 02F74h, 02F1Ah, 14 dup(grSpc), VWallSymbol, space

dw fieldSpacing, grSpc, 02F45h, 02F73h,02F63h, grSpc, 02FC4h, grSpc, 02F45h, 02F78h, 02F69h ,02F74h, 02F13h, xSize - xField - 17 dup(grSpc), VWallSymbol, space

dw space, 0FC8h, xField dup(HWallSymbol), 0FCAh, xSize - xField - 5 dup(HWallSymbol), 0FBCh, space

dw xSize dup(space)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; BANNER

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

widthOfBanner equ 40 ;

allWidth equ 80 ;

black equ 0020h ;

white equ 4020h ;

blackVWallSymbol equ 00FBAh

blackHWallSymbol equ 00FCDh

wastedBanner dw 00FC9h, widthOfBanner-2 dup(blackHWallSymbol), 0FBBh

dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol

dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, 2 dup(black), 2 dup(white), black, 4 dup(white), black, 3 dup(white), black, 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol

dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, black, white, black, white, black,white, black, black, black, black, black, white, 2 dup(black), white, 2 dup(black), black, white, black, black, white, 5 dup(black), blackVWallSymbol

dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, 3 dup(white), black, 4 dup(white), 2 dup(black), white, 2 dup(black), 2 dup(white), 2 dup(black), white, 2 dup(black), white, 5 dup(black), blackVWallSymbol

dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, white, black, white, 4 dup(black), white, 2 dup(black), white, 2 dup(black), white, 2 dup(black), black, white, 2 dup(black), white, 5 dup(black), blackVWallSymbol

dw blackVWallSymbol, 6 dup(black), 2 dup(white, black), 2 dup(black), white, black, white, black, 4 dup(white), 2 dup(black), white, 2 dup(black), 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol

dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol

dw blackVWallSymbol, 7 dup(black) ,08F50h, 08F72h, 08F65h, 08F73h, 08F73h, 08F00h, 08F61h, 08F6Eh, 08F79h, 08F00h, 08F6Bh, 08F65h, 08F79h, 08F00h, 08F74h, 08F6Fh, 08F00h, 08F65h, 08F78h, 08F69h, 08F74h, 10 dup(black), blackVWallSymbol

dw 0FC8h, widthOfBanner-2 dup(blackHWallSymbol), 0FBCh

snakeMaxSize equ 30

snakeSize db 3

PointSize equ 2

snakeBody dw 1D0Dh, 1C0Dh, 1B0Dh, snakeMaxSize-2 dup(0000h)

brickWallSize equ 9

brickWall1 dw 0303h, 0302h, 0301h, 0300h, 02FFh, 0203h, 0103h, 0003h, 0FF03h

brickWall2 dw 0103h, 0003h, 0FF03h, 0FE03h, 0FD03h, 0FD02h, 0FD01h, 0FD00h, 0FCFFh

brickWall3 dw 01FEh, 00FEh, 0FFFEh, 0FEFEh, 0FDFEh, 0FD01h, 0FD00h, 0FCFFh, 0FCFEh

brickWall4 dw 01FEh, 00FEh, 0FFFEh, 0FEFEh, 002FEh, 00401h, 00400h, 003FFh, 003FEh

brickWallTemplate dw brickWallSize dup(0)

brickWallTrue dw brickWallSize dup(0)

stopVal equ 00h

forwardVal equ 01h

backwardVal equ -1

Bmoveright db 01h

Bmovedown db 00h

minWaitTime equ 1

maxWaitTime equ 9

waitTime dw maxWaitTime

deltaTime equ 1

.code

main:

mov ax, @data ;

mov ds, ax ;

mov dataStart, ax ; upload data

mov ax, videoStart ; put in ax beggining of output in videobuffer

mov es, ax ; move ax to es-segment register

xor ax, ax ;

;

clearScreen ; clear console

;

call initAllScreen ; initialize screen

;

call mainGame ; go to main cycle of game

;

to\_close: ;

call printBanner ;

mov ah,7h ; 7h - console input w/o eho(wait for input to close app)

int 21h ;

esc\_exit:

clearScreen ; clear console

;

mov ah, 4ch ; 4ch- function to exit

int 21h ;

;

;

;ZF = 1 - buffer is empty ;

;AH = scan-code ;

CheckBuffer MACRO ; check if there was input

mov ah, 01h ; 01h- read symbol with waiting and check for ctrl-break

int 16h ;

ENDM ;

;

ReadFromBuffer MACRO ; read what was inputed

mov ah, 00h ; 00h - wait for next input

int 16h ;

ENDM ;

;

;result in cx:dx ;

GetTimerValue MACRO ;

push ax ; save what is in ax

;

mov ax, 00h ; get time

int 1Ah ; preryvaniye for time

;

pop ax ; vosstanovlenye what is in ax

ENDM ;

;

printBanner PROC

push es ;

push 0B800h ;

; 0b800h

pop es ; ES=0B800h

;

mov di, 7\*allWidth\*2 + (allWidth - widthOfBanner) ;

mov si, offset wastedBanner ;

mov cx, 10 ;

cld ; DF =0

loopPrintBanner: ;

;

push cx ;

;

mov cx, widthOfBanner ;

;rep - repeat

rep movsw ; movsw - what is in DS:(E)SI to ES:(E)D

;

add di, 2\*(allWidth - widthOfBanner);

;

pop cx ;

loop loopPrintBanner ;

std ;DF=1

pop es ;

ret ;

ENDP

drawBrickWall PROC

push cx

push bx

mov cx, brickWallSize

mov si, offset brickWallTrue ;SMESCHENIE OF BRICKWALLTRUE TO SI

loopBrickWall:

mov bx, [si] ; put in si ocherednoy symbol

add si, PointSize ;

; get position in videobuffer

call CalcOffsetByPoint ; get smeschenie of outputed symbol

;

mov di, bx ; put in di position

;

mov ax, BWallSymbol ; put entered symbol to ax

stosw ; output stosw - save es:di with what is in al or ax

loop loopBrickWall

pop bx

pop cx

ret

ENDP

destroyWall PROC

push cx

mov cx, brickWallSize

mov si, offset brickWallTrue

loopDestroyWall:

mov bx, [si] ; put in si ocherednoy symbol

add si, PointSize ;

call CalcOffsetByPoint ; get smeschenie of outputed symbol

;

mov di, bx ; put in di position

;

mov ax, space ; put entered symbol to ax

stosw ; output stosw - save es:di with what is in al or ax

loop loopDestroyWall

pop cx

ret

ENDP

;

initAllScreen PROC ;

mov si, offset screen ; put to di

xor di, di ;

; now ds:si point on symbols we want to input

; aand es:di on di'yi sembol of console

mov cx, xSize\*ySize ; put to dz amount of symbols in console

rep movsw ; rewrite all cx symbols from ds:si to console es:di

;

;

xor ch, ch ;

mov cl, snakeSize ; put to cl size of snake

mov si, offset snakeBody; to si upload smescheniye of begining of snake

;

loopInitSnake: ; cycle to uotput snakes body

mov bx, [si] ; put to si ocherednoy symbol of snake's body

add si, PointSize ; add to si PointSize (2), because every point is 2 bytes (colour+symbol)

call CalcOffsetByPoint ; ; get smeschenie of outputed symbol

;

mov di, bx ; ; put in di position

;

mov ax, snakeBodySymbol ; put to ax outputed symbol

stosw ; output stosw - save es:di with what is in al or ax

loop loopInitSnake ;

;

call GenerateRandomApple; generate random apple

;

ret ;

ENDP ;

;get smescheniye of videobuffer like (bh + (bl \* xSize))\*oneMemoBlock

;input: coordinates (x,y) in bx

;output: smescheniye in bx

CalcOffsetByPoint PROC ;

push ax ; save what is in ax and dx

push dx ;

;

xor ah, ah ;

mov al, bl ;

mov dl, xSize ; xSize - size of string

mul dl ; al\*dl

mov dl, bh ;

xor dh, dh ;

add ax, dx ; ax+dx

mov dx, oneMemoBlock ; oneMemoBlock - length of every block

mul dx ;

mov bx, ax ;

;

pop dx ; vosstanovleniye ax and dx

pop ax ;

ret ;

ENDP ;

;move body of snale in massive

;delete last element

;colour last element

MoveSnake PROC ;

push ax ;

push bx ;

push cx ;

push si ; save what is in registers

push di ;

push es ;

;

mov al, snakeSize ;

xor ah, ah ;

mov cx, ax ;

mov bx, PointSize ; put size of one dot to bx

mul bx ; now ax-real position in memory otnositelno nachala massiva

mov di, offset snakeBody; put to di smescheniye of head af snake

add di, ax ; di - adress of next element after last element of massiv

mov si, di ;

sub si, PointSize ; si-adress of last massiv element

;

push di ; save what is in di

; delete end of snake from screen

mov es, videoStart ; put to es smeschenie videobufera

mov bx, ds:[si] ; put to bx last element of snake

call CalcOffsetByPoint ; calculate its position on screen

mov di, bx ; put to di position we need to clear

mov ax, space ; put empty point to ax

stosw ; write(put what is in ax to es:di)

;

pop di ; vosstanavlivayem di

;

mov es, dataStart ; to work with data(before es pointed to videobufer)

std ; df=1; go from end to start

rep movsw ; rewrite frfom ds:si to es:di (si - prelast elem of snake, di - last element)

;smeschaem all snake on 1 step

;

mov bx, snakeBody ; put to dx head of snake

;

add bh, Bmoveright ; renew head coordinates

add bl, Bmovedown ;

mov snakeBody, bx ; save new head

;

;now all body is moved ;

pop es ;

pop di ;

pop si ;

pop cx ; vosstanavlivayem registers

pop bx ;

pop ax ;

ret ;

ENDP ;

mainGame PROC

push ax ;

push bx ;

push cx ;

push dx ; save what is in registers

push ds ;

push es ;

;

checkAndMoveLoop: ;

;

CheckBuffer ; check if there was input

jnz skipJmp2 ; if yes - skipJmp2

jmp far ptr noSymbolInBuff ; if no- noSymbolInBuff

;

skipJmp2: ;

ReadFromBuffer ; read symbol for buffer

;

cmp ah, KExit ; if exit was pressed

jne skipJmp ; else skipJmp

;

jmp far ptr esc\_exit ; go to endf game endLoop

;

skipJmp: ;

cmp ah, KMoveLeft ; if "left" was pressed

je setMoveLeft ;

;

cmp ah, KMoveRight ; if "right" was pressed

je setMoveRight ;

;

cmp ah, KMoveUp ;if "up" was pressed

je setMoveUp ;

;

cmp ah, KMoveDown ;if "down" was pressed

je setMoveDown ;

;

cmp ah, KUpSpeed ; move up key is pressed

je setSpeedUp ;

;

cmp ah, KDownSpeed ; move down key is pressed

je setSpeedDown ;

;

jmp noSymbolInBuff ;

;

setMoveLeft: ;

mov al, Bmoveright ; check if tried to switch directions

cmp al, forwardVal ;

jne setMoveLeft\_ok ;

jmp noSymbolInBuff ;

;

setMoveLeft\_ok: ;

;

mov Bmoveright, backwardVal ; direction right - negative

mov Bmovedown, stopVal ; direction down - null

jmp noSymbolInBuff ;

;

setMoveRight: ;

mov al, Bmoveright ; check if tried to switch directions

cmp al, backwardVal ;

jne setMoveRight\_ok ;

jmp noSymbolInBuff ;

;

setMoveRight\_ok: ;

;

mov Bmoveright, forwardVal ; direction right - positive

mov Bmovedown, stopVal ; direction down - null

jmp noSymbolInBuff ;

;

setMoveUp: ;

mov al, Bmovedown ;check if switch directions

cmp al, forwardVal ;

jne setMoveUp\_ok ;

jmp noSymbolInBuff ;

;

setMoveUp\_ok: ;

;

mov Bmoveright, stopVal ;direction down-negative

mov Bmovedown, backwardVal ;

jmp noSymbolInBuff ;

;

setMoveDown: ;

mov al, Bmovedown ;check if switch direction

cmp al, backwardVal ;

jne setMoveDown\_ok ;

jmp noSymbolInBuff ;

;

setMoveDown\_ok: ;

;

mov Bmoveright, stopVal ;direction right - null

mov Bmovedown, forwardVal ;direction down - positive

jmp noSymbolInBuff ;

;

setSpeedUp: ;

mov ax, waitTime ;move delay to ax

cmp ax, minWaitTime ;compare it to min

je noSymbolInBuff ;if min - let it go

;

sub ax, deltaTime ;make time less

mov waitTime, ax ;new delay

;

mov es, videoStart ;

mov di, offset speed - offset screen ;

mov ax, es:[di] ;

inc ax ;

mov es:[di], ax ;

;

jmp noSymbolInBuff ;

;

setSpeedDown: ;

mov ax, waitTime ;

cmp ax, maxWaitTime ;

je noSymbolInBuff ;

;

add ax, deltaTime ;

mov waitTime, ax ;

;

mov es, videoStart ;

mov di, offset speed - offset screen ;

mov ax, es:[di] ;

dec ax ;

mov es:[di], ax ;

;

jmp noSymbolInBuff ;

;

noSymbolInBuff: ;

call MoveSnake ;move snake on screen

;

mov bx, snakeBody ;move to bx head of snake

checkSymbolAgain: ;

call CalcOffsetByPoint ;in bx now- smescheniye yacheiki konsoli with new head

;

mov es, videoStart ;to es smescheniye videobufera

mov ax, es:[bx] ;put to ax symbol where snake should be

;

cmp ax, appleSymbol ;if this symbol is apple

je AppleIsNext ;

;

cmp ax, snakeBodySymbol ;if this symbol is body

je SnakeIsNext ;

;

cmp ax, HWallSymbol ;if symbol is horizontal wall

je PortalUpDown ;

;

cmp ax, VWallSymbol ;if symbol is vertical wall

je PortalLeftRight ;

;

cmp ax, BWallSymbol ;if symbol is horizontal wall

je SnakeIsNext ;

;

cmp ax, VWallSpecialSymbol ;if it is cross

je PortalLeftRight ;

;

jmp GoNextIteration ;

;

AppleIsNext: ;

call destroyWall

call incSnake ;make snake longer

call GenerateRandomApple ;new apple

call incScore ;make score bigger

jmp GoNextIteration ;next step

SnakeIsNext: ;

jmp endLoop ;end of game

PortalUpDown: ;

mov bx, snakeBody ;put head to bx

sub bl, yField ;y-height of console

cmp bl, 0 ;check if high or low board

jg writeNewHeadPos ;redraw head of snake

;

; if upper wall

add bl, yField\*2 ; new coordinates

;

writeNewHeadPos: ;

mov snakeBody, bx ; new head

jmp checkSymbolAgain ; check again

;

PortalLeftRight: ;

mov bx, snakeBody ;

sub bh, xField ;

cmp bh, 0 ;

jg writeNewHeadPos ; same analogy with vertical wall

;

add bh, xField\*2 ;

jmp writeNewHeadPos ;

;

GoNextIteration: ;

mov bx, snakeBody ;bx-new head

call CalcOffsetByPoint ;count position

mov di, bx ; now in di new smeschniye bx in console

mov ax, snakeBodySymbol ;put in ax symbol of snake

stosw ;input in console

;

call Sleep ; delay

;

jmp checkAndMoveLoop ;

;

endLoop: ;

pop es ;

pop ds ;

pop dx ;vosstanavlivayem registers

pop cx ;

pop bx ;

pop ax ;

ret ;

ENDP ;

;

Sleep PROC ;

push ax ;

push bx ; save registers

push cx ;

push dx ;

;

GetTimerValue ; get current time

;

add dx, waitTime ; dx+waittime

mov bx, dx ; put it to bx

;

checkTimeLoop: ;

GetTimerValue ;get current time

cmp dx, bx ; ax - current value, bx - needed value

jl checkTimeLoop ;if early-go to next iteration

;

pop dx ;

pop cx ;

pop bx ;vosstanovleniye registers

pop ax ;

ret ;

ENDP ;

GenerateRandomApple PROC ;

push ax ;

push bx ;

push cx ; save registers

push dx ;

push es ;

;

mov ah, 2Ch ; read current time

int 21h ; ch - hour, cl - min, dh - sec, dl - msec

mov al, dl

mul dh ; now to ax random number

xor dx, dx

mov cx, 04h

div cx

mov bh, dl

cmp bh, 0

jne rnd1

mov si, offset brickWall1

jmp writeToTemplate

rnd1:

cmp bh, 1

jne rnd2

mov si, offset brickWall2

jmp writeToTemplate

rnd2:

cmp bh, 2

jne rnd3

mov si, offset brickWall3

jmp writeToTemplate

rnd3:

mov si, offset brickWall4

jmp writeToTemplate

writeToTemplate:

mov di, offset brickWallTemplate

mov cx, brickWallSize

toTemplate:

push ax

mov ax, [si]

mov [di],ax

pop ax

add di, PointSize

add si, PointSize

loop toTemplate

loop\_random: ;

mov ah, 2Ch ; read current time

int 21h ; ch - h, cl - min, dh -sec, dl-msec

;

mov al, dl ; get random number

mul dh ;now in ax random number

;

xor dx, dx ;

mov cx, xField ;in dx width of field

div cx ;get num of string of apple

add dx, 2 ;add smescheniye from start of os'

mov bh, dl ;save x

;

xor dx, dx ;

mov cx, yField ;

div cx ; analogically for y

add dx, 2 ;

mov bl, dl ;now in bx - apple coordinates

;

push bx

call CalcOffsetByPoint;count smescheniye

mov es, videoStart ;to es - start of videobufer

mov ax, es:[bx] ;to ax - symbol where apple will be

pop bx

;

cmp ax, space ;compare with empty dot

jne loop\_random ;if not empty -new make coordinates

mov cx, brickWallSize

mov si, offset brickWallTemplate

loopRandomWall:

push bx ;cycle with snake output

add bx, [si] ;to si ocherednoy symbol of snake

push bx

call CalcOffsetByPoint;count smescheniye

mov es, videoStart ;to es start of videobufer

mov ax, es:[bx] ;to ax - symbol where apple will be

pop bx

pop bx

cmp ax, space

jne loop\_random

add si, PointSize ;add to si PointSize(2), because every dot takes 2 bytes(color + symbol)

loop loopRandomWall

mov cx, brickWallSize

mov si, offset brickWallTemplate

mov di, offset brickWallTrue

loopCreateWall:

push ax ;cycle to output snake

mov ax, [si] ;in si ocherednoy symbol of snake

add ax, bx

mov [di], ax

add si, PointSize

add di, PointSize

pop ax ;output

loop loopCreateWall

call drawBrickWall

push bx

call CalcOffsetByPoint;count smescheniye

mov es, videoStart ;in es start of videobufer

mov ax, appleSymbol;

mov es:[bx], ax ;output apple symbol

pop bx

;

pop es ;

pop dx ;

pop cx ;vosstanavlivayem registers

pop bx ;

pop ax ;

ret ;

ENDP ;

;save tail of snake if no overloading

incSnake PROC ;

push ax ;

push bx ; save registers

push di ;

push es ;

;

mov al, snakeSize ;

cmp al, snakeMaxSize ;

je return ;if max - exit

;

; make snake longer

inc al ; al+1

mov snakeSize, al ; new size

dec al ; al-1

;

;

mov bl, PointSize ; vosstanavlivayem end

mul bl ;now in ax nuzhnoye smescheniye

;

mov di, offset snakeBody

add di, ax ; di now - point of vosstanovlenie

;

mov es, dataStart ;put data to es

mov bx, es:[di] ;in bx - point to vosstanovit'

call CalcOffsetByPoint;get coordinates

;

mov es, videoStart ;in es-smeschenie videobufera

mov es:[bx], snakeBodySymbol ;put one dot of body

;

return: ;

pop es ;

pop di ;vosstanavlivayem registers

pop bx ;

pop ax ;

ret ;

ENDP ;

;

incScore PROC ;

push ax ;

push es ;

push si ;

push di ;

mov es, videoStart ;

mov cx, scoreSize ; ;max pos value

mov di, offset score + (scoreSize - 1)\*oneMemoBlock - offset screen

;

loop\_score: ;

mov ax, es:[di] ;

cmp al, 39h ;'9' symbol

jne nineNotNow ;

;

sub al, 9 ;

mov es:[di], ax ;

;

sub di, oneMemoBlock ;return to symbol back

loop loop\_score

jmp return\_incScore

nineNotNow:

inc ax

mov es:[di], ax

return\_incScore:

pop di

pop si

pop es

pop ax

ret

ENDP

end main