Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №5**

**По дисциплине «Программирование на языке ассемблера»**

**Тема: «Создание видеоигры»**

Выполнил: Купаленко А.Д.

Группа 21ИТ1

Проверила: Рогулев В.С.

Преподаватель кафедры ТП

Полоцк 2022

**Тема:** **Создание видеоигры**

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

**Задание**

Игра «Пакман».

Цель: ходим по лабиринту и собираем случайно появляющиеся яблочки,

убегаем от противника (один вид лабиринта, два или три противника).

Окончание: проигрыш – столкновение с противником, выигрыш – нет.

Информация: счет.

Усложнение: добавление бонусов (атака, увеличение скорости и т. п.).

**Код:**

IDEAL

MODEL small

STACK 100h

screen\_RAM\_graphics equ 0A000h

RIGHT\_KEY equ 77

LEFT\_KEY equ 75

UP\_KEY equ 72

DOWN\_KEY equ 80

ESCAPE equ 1

DATASEG

PAReturnAddress dw 0 ;Return addresses variables

PMAReturnAddress dw 0

CHK4PMRA dw 0

PACMANRA dw 0

DLTRA dw 0

RowLoop dw 0 ;variables for procedure uses (so the values inside the important variables wont be changed)

ColumnLoop dw 0

offsetStart dw 0

Color dw 0

checkDotsRA dw 0

DLTDOTRA dw 0

Direction dw 0

;Xs and Ys for procedures

X dw 0

Y dw 0

Y2 dw 0

X2 dw 0

X3 dw 0

Y3 dw 0

Bool dw 0

DotCounter dw 0 ;counts the amount of dots that were eaten by pacman

WinBool dw 0 ;boolean if user won

finalBool dw 0 ;boolean for procedures

moveBool dw 0 ;another boolean for procedures

StartGameBool dw 0 ;boolean to check if the game should start

EndGameBool dw 0 ;boolean to see if the game needs to be ended

StartInstructionsBool dw 0

TouchedPMBool dw 0

;PacMan XY and others:

PACMAN\_X dw 153

PACMAN\_Y dw 95

PMRightCounter dw 0

PMLeftCounter dw 0

PMUpCounter dw 0

PMDownCounter dw 0

;Ghost1 XY:

GHOST1\_X dw 100

GHOST1\_Y dw 11

GHOST1RIGHT1ST dw 107 ;These variables contain the amount of steps the ghost moves in each direction

GHOST1DOWN1ST dw 30

GHOST1LEFT1ST dw 108

GHOST1DOWN2ND dw 55

GHOST1RIGHT2ND dw 108

GHOST1DOWN3RD dw 50

GHOST1LEFT2ND dw 108

GHOST1DOWN4TH dw 30

GHOST1RIGHT3RD dw 108

GHOST1UP1ST dw 30

GHOST1LEFT3RD dw 108

GHOST1UP2ND dw 50

GHOST1RIGHT4TH dw 108

GHOST1UP3RD dw 55

GHOST1LEFT4TH dw 108

GHOST1UP4TH dw 30

;Ghost2 XY:

GHOST2\_X dw 12

GHOST2\_Y dw 11

GHOST2RIGHT1ST dw 58

GHOST2DOWN1ST dw 30

GHOST2RIGHT2ND dw 14

GHOST2LEFT1ST dw 42

GHOST2DOWN2ND dw 50

GHOST2RIGHT3RD dw 42

GHOST2LEFT2ND dw 72

GHOST2UP1ST dw 80

;Ghost3 XY:

GHOST3\_X dw 12

GHOST3\_Y dw 175

GHOST3UP1ST dw 80

GHOST3RIGHT1ST dw 72

GHOST3LEFT1ST dw 42

GHOST3DOWN1ST dw 50

GHOST3RIGHT2ND dw 42

GHOST3LEFT2ND dw 13

GHOST3DOWN2ND dw 30

GHOST3LEFT3RD dw 58

;Ghost4 XY:

GHOST4\_X dw 295

GHOST4\_Y dw 11

GHOST4DOWN1ST dw 80

GHOST4LEFT1ST dw 72

GHOST4RIGHT1ST dw 42

GHOST4UP1ST dw 50

GHOST4LEFT2ND dw 42

GHOST4RIGHT2ND dw 13

GHOST4UP2ND dw 30

GHOST4RIGHT3RD dw 58

;Ghost5 XY:

GHOST5\_X dw 295

GHOST5\_Y dw 175

GHOST5LEFT1ST dw 59

GHOST5UP1ST dw 30

GHOST5LEFT2ND dw 14

GHOST5RIGHT1ST dw 43

GHOST5UP2ND dw 50

GHOST5LEFT3RD dw 42

GHOST5RIGHT2ND dw 72

GHOST5DOWN1ST dw 80

;Xs and Ys for procedure uses

CHK4PM\_X dw 0

CHK4PM\_Y dw 0

;Main Menu's Letters' Xs and Ys

PixelatedC\_X dw 70

PixelatedC\_Y dw 40

PixelatedR\_X dw 96

PixelatedR\_Y dw 40

PixelatedE\_Y dw 40

PixelatedE\_X dw 122

PixelatedN\_Y dw 40

PixelatedN\_X dw 148

PixelatedN2\_Y dw 40

PixelatedN2\_X dw 176

PixelatedU\_Y dw 40

PixelatedU\_X dw 202

PixelatedI\_Y dw 40

PixelatedI\_X dw 228

PixelatedS\_Y dw 40

PixelatedS\_X dw 254

PixelatedApostrophe\_Y dw 35

PixelatedApostrophe\_X dw 241

PixelatedL\_Y dw 72

PixelatedL\_X dw 105

PixelatedE2\_Y dw 72

PixelatedE2\_X dw 129

PixelatedG\_Y dw 72

PixelatedG\_X dw 155

PixelatedE3\_Y dw 72

PixelatedE3\_X dw 181

PixelatedN3\_Y dw 72

PixelatedN3\_X dw 207

PixelatedD\_Y dw 72

PixelatedD\_X dw 235

MMMessage1 db "1 - Start Game "

MMMessage2 db "2 - Instructions "

MMMessage3 db "3 - Exit Game "

Lore1 db "When Tamir was a little boy, he wanted to be called Barak.$"

Lore2 db "If you type barak mammon in Hebrew when the keyboard is set to English,$"

Lore3 db "You get crennui! Our little Tamir is very fat and his goal is to eat all$"

Lore4 db "the food in the world, but many skinny people want to stop him!$"

Lore5 db "Help little Tamir eat all the food in the world!$"

Lore6 db "Use the arrow keys to move little Tamir around,$"

Lore7 db "and make sure you avoid all the skinny people,which look like ghosts$"

Lore8 db "Press any key to go back to the main menu . . .$"

ColorValueIncrease dw 0

include 'PROJFC.asm'

CODESEG ;This program is a pacman based game. first of all it brings the user to the main menu, where he can choose to exit the game, go to the instructions and story of the game and start the game.

start: ;If the user decides to go to the game, it begins the game loop of the pacman game and the user needs to go over all the dots in the map without touching the ghosts.

mov ax, @data ;if the user touches a ghost, he looses. If the user eats all the dots he wins.

mov ds, ax

mov ax,13h

int 10h

mainMenuLoop:;loop that keeps changing the colors of the main menu text while the user is at the main menu

call mainMenu

cmp [StartGameBool],1;checks to see where the user wants to go according to the booleans

je StartGame

cmp [EndGameBool],1

je exit

cmp [StartInstructionsBool],1

je StartInstructions

jmp mainMenuLoop

StartGame:

call Clear\_Screen

call Print\_Starting\_Locs

call Print\_Map

gameLoop:;the game loop that keeps re printing everything that needs to be reprinted

call PacMan\_Move

call Ghost1\_Move

call Ghost2\_Move

call Ghost3\_Move

call Ghost4\_Move

call Ghost5\_Move

cmp [TouchedPMBool],0

jne exit

call PrintDots

cmp [WinBool],1

je exit

jmp gameLoop

StartInstructions:;prints the instructions

call Clear\_Screen

call Print\_Instructions

call Clear\_Screen

mov [StartInstructionsBool],0

mov ax,13h

int 10h

jmp mainMenuLoop

exit:

mov ax,03h

int 10h

mov ax, 4c00h

int 21h

proc Clear\_Screen;clears the screen

push es

push cx

push di

push ax

mov ax,13h

int 10h

mov ax,screen\_RAM\_graphics

mov es,ax ; es:di - video memory

xor di,di

mov cx,320\*200/2

xor ax,ax ;color

rep stosw ; mov es:[di],ax add di,2

pop ax

pop di

pop cx

pop es

ret

endp Clear\_Screen

proc Print\_Anything;According to the parameters given it prints any character.

pop [PAReturnAddress]

pop [x];pops the pushed parameters

pop [y]

pop [RowLoop]

pop [ColumnLoop]

pop [offsetStart]

push ax

push bx

push cx

push dx

mov bx,[offsetStart];moves the offset where the color array values start at

mov cx,[RowLoop];big loop

Print1:

push cx

mov cx,[ColumnLoop];small loop

Print2:

push cx

mov cx,[x];x

mov dx,[y];y

mov al,[byte ptr bx];color

mov ah,0Ch

int 10h

pop cx

inc [x]

inc bx

loop Print2;small loop loop.

inc [y];increases the y by 1 to start at a new row in the big loop

push ax

mov ax,[ColumnLoop]

sub [x],ax;returns x back to the starting value to start printing the new row

pop ax

pop cx

loop Print1;big loop loop

pop dx

pop cx

pop bx

pop ax

push [PAReturnAddress]

ret

endp Print\_Anything

proc Print\_Map\_Anything;Same as Print\_Anything but meant to print the map since there is only one color and not an array of colors.

pop [PMAReturnAddress]

pop [x]

pop [y]

pop [RowLoop]

pop [ColumnLoop]

pop [Color]

push ax

push bx

push cx

push dx

mov cx,[RowLoop]

Print1\_1:

push cx

mov cx,[ColumnLoop]

Print2\_1:

push cx

mov cx,[x]

mov dx,[y]

mov ax,[color]

mov ah,0Ch

int 10h

pop cx

inc [x]

loop Print2\_1

inc [y]

push ax

mov ax,[ColumnLoop]

sub [x],ax

pop ax

pop cx

loop Print1\_1

pop dx

pop cx

pop bx

pop ax

push [PMAReturnAddress]

ret

endp Print\_Map\_Anything

proc Print\_Starting\_Locs;prints the starting locations of all the characters

Right\_PacMan\_Print:

push offset RIGHT\_PM;starting offset

push 12 ;columns

push 13 ;rows

push [PACMAN\_Y] ;y

push [PACMAN\_X] ;x

call Print\_Anything

Ghost1\_Print:

push offset Ghost1;starting offset

push 13 ;columns

push 14 ;rows

push [GHOST1\_Y] ;y

push [GHOST1\_X] ;x

call Print\_Anything

Ghost2\_Print:

push offset Ghost2;starting offset

push 13 ;columns

push 14 ;rows

push [GHOST2\_Y] ;y

push [GHOST2\_X] ;x

call Print\_Anything

Ghost3\_Print:

push offset Ghost3;starting offset

push 13 ;columns

push 14 ;rows

push [GHOST3\_Y] ;y

push [GHOST3\_X] ;x

call Print\_Anything

Ghost4\_Print:

push offset Ghost4;starting offset

push 13 ;columns

push 14 ;rows

push [GHOST4\_Y] ;y

push [GHOST4\_X] ;x

call Print\_Anything

Ghost5\_Print:

push offset Ghost5;starting offset

push 13 ;columns

push 14 ;rows

push [GHOST5\_Y] ;y

push [GHOST5\_X] ;x

call Print\_Anything

ret

endp Print\_Starting\_Locs

proc PacMan\_Move;moves pacman according to the direction received and checks for borders

pop [PACMANRA]

push ax

push bx

push cx

push dx

Start\_Move:

mov ah,0Bh ;checks if a key is pressed

int 21h

cmp al,0

je Receive\_EndCP

Move\_PacMan:

;receives a key from the user

mov ah,07h

int 21h

;Checking the pressed key

mov cx,ax

mov ch,0

cmp cx,[Direction]

je Receive\_EndCP

cmp al,UP\_KEY

je Up

cmp al,DOWN\_KEY

je Down

cmp al,RIGHT\_KEY

je RightCP

cmp al,LEFT\_KEY

je Left

jmp Receive\_End

Left:

checkPMLeft: ;checks if pacman can be moved

call check\_PM\_left

cmp [moveBool],0

jne dontPrintLeft

call Move\_PM\_Left

mov [Direction],LEFT\_KEY

dontPrintLeft:

mov [moveBool],0

jmp Receive\_End

RightCP:

jmp Right

Receive\_EndCP:

cmp [Direction],LEFT\_KEY

je Left

cmp [Direction],RIGHT\_KEY

je Right

cmp [Direction],DOWN\_KEY

je Down

cmp [Direction],UP\_KEY

je Up

jmp Receive\_End

Up:

checkPmUP:

call check\_PM\_up

cmp [moveBool],0

jne dontPrintUp

call Move\_PM\_up

mov [Direction],UP\_KEY

dontPrintUp:

mov [moveBool],0

jmp Receive\_End

Down:

checkPMDown:

call check\_PM\_down

cmp [moveBool],0

jne dontPrintDown

call Move\_PM\_Down

mov [Direction],DOWN\_KEY

dontPrintDown:

mov [moveBool],0

jmp Receive\_End

Right:

checkPMRight:

call check\_PM\_right

cmp [moveBool],0

jne dontPrintRight

call Move\_PM\_Right

mov [Direction],RIGHT\_KEY

dontPrintRight:

mov [moveBool],0

jmp Receive\_End

Receive\_End:

pop dx

pop cx

pop bx

pop ax

push [PACMANRA]

ret

endp PacMan\_Move

proc Move\_PM\_up ;moves pacman upwards

push ax

mov [PMRightCounter],0;resets other counters

mov [PMLeftCounter],0

mov [PMDownCounter],0

dec [PACMAN\_Y]

cmp [PMUpCounter],0;checks which part of the up pacman should be printed

je PrintFirstUpPM

cmp [PMUpCounter],1

je PrintSecondUpPM

cmp [PMUpCounter],2

je PrintThirdUpPM

PrintFirstUpPM:

push offset UP\_PacMan

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMUpCounter]

jmp deleteBottomSide

PrintSecondUpPM:

push offset UP\_PM2

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMUpCounter]

jmp deleteBottomSide

PrintThirdUpPM:

push offset UP\_PM3

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

mov [PMUpCounter],0

deleteBottomSide:

push 0

push 13

push 2

mov ax,[PACMAN\_Y]

add ax,12

push ax

push [PACMAN\_X]

call Print\_Map\_Anything

pop ax

ret

endp Move\_PM\_up

proc Move\_PM\_Left;moves pacman to the left same like the last procedure

push ax

mov [PMRightCounter],0

mov [PMUpCounter],0

mov [PMDownCounter],0

dec [PACMAN\_X]

cmp [PMLeftCounter],0

je PrintFirstLeftPM

cmp [PMLeftCounter],1

je PrintSecondLeftPM

cmp [PMLeftCounter],2

je PrintThirdLeftPM

PrintFirstLeftPM:

push offset LEFT\_PM

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMLeftCounter]

jmp deleteRightSide

PrintSecondLeftPM:

push offset LEFT\_PM2

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMLeftCounter]

jmp deleteRightSide

PrintThirdLeftPM:

push offset LEFT\_PM3

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

mov [PMLeftCounter],0

deleteRightSide:

push 0

push 2

push 13

push [PACMAN\_Y]

mov ax,[PACMAN\_X]

add ax,12

push ax

call Print\_Map\_Anything

pop ax

ret

endp Move\_PM\_Left

proc Move\_PM\_Right;same as before but to the right

push ax

mov [PMLeftCounter],0

mov [PMUpCounter],0

mov [PMDownCounter],0

inc [PACMAN\_X]

cmp [PMRightCounter],0

je PrintFirstRightPM

cmp [PMRightCounter],1

je PrintSecondRightPM

cmp [PMRightCounter],2

je PrintThirdRightPM

PrintFirstRightPM:

push offset RIGHT\_PM

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMRightCounter]

jmp deleteLeftSide

PrintSecondRightPM:

push offset RIGHT\_PM2

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMRightCounter]

jmp deleteLeftSide

PrintThirdRightPM:

push offset RIGHT\_PM3

push 12

push 13

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

mov [PMRightCounter],0

deleteLeftSide:

push 0

push 1

push 13

push [PACMAN\_Y]

mov ax,[PACMAN\_X]

dec ax

push ax

call Print\_Map\_Anything

pop ax

ret

endp Move\_PM\_Right

proc Move\_PM\_Down;same as before but to the bottom

push ax

mov [PMUpCounter],0

mov [PMRightCounter],0

mov [PMLeftCounter],0

inc [PACMAN\_Y]

cmp [PMDownCounter],0

je PrintFirstDownPM

cmp [PMDownCounter],1

je PrintSecondDownPM

cmp [PMDownCounter],2

je PrintThirdDownPM

PrintFirstDownPM:

push offset DOWN\_PM

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMDownCounter]

jmp deleteTopSide

PrintSecondDownPM:

push offset DOWN\_PM2

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

inc [PMDownCounter]

jmp deleteTopSide

PrintThirdDownPM:

push offset DOWN\_PM3

push 13

push 12

push [PACMAN\_Y]

push [PACMAN\_X]

call Print\_Anything

mov [PMDownCounter],0

deleteTopSide:

push 0

push 13

push 1

mov ax,[PACMAN\_Y]

dec ax

push ax

push [PACMAN\_X]

call Print\_Map\_Anything

pop ax

ret

endp Move\_PM\_Down

include "PROJFMM.asm"

include "PROJFMAP.asm"

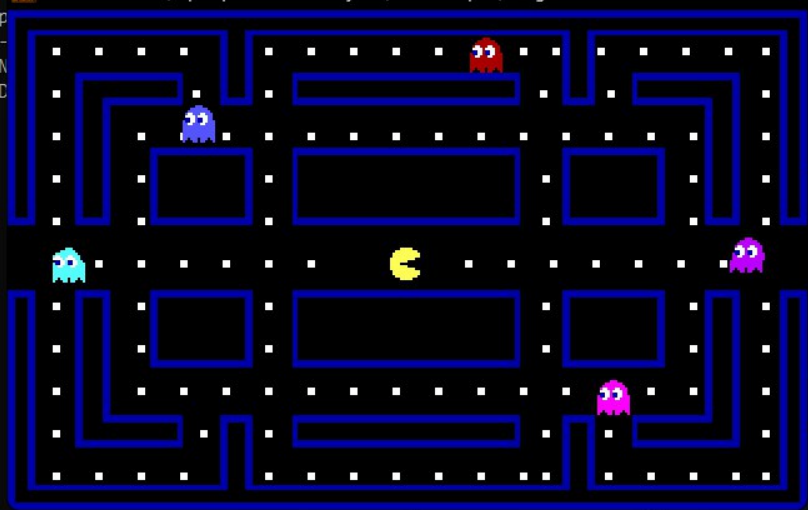
include "PROJFDT.asm"

include "CHECKPM.asm"

include "PROJFG1.asm"

END start

**Работа программы:**

****