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 «Программирование на языке ассемблера»

на тему «Создание видеоигры»

вариант №3

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 250504  Казаченко П.Е. |  | Проверил:  Туровец Н.О. |

Минск 2023

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

**Теоретические сведения** Для выполнения работы требуется рассмотреть следующие элементы языка ассемблера и операционной системы:

1. *Прямой доступ к видеопамяти.*

Кроме использования прерываний DOS, описанных в лабораторной рабо-те №2, программа может выводить текст на экран с помощью пересылки данных в специальную область памяти, связанную с видеоадаптером – видеопамять. Этот вариант вывода более быстр, чем при выводе символов через прерывания, а также позволяет формировать в консоли определенные эффекты, часто не используемые в режиме вывода в позицию курсора.

В большинстве текстовых видеорежимов под видеопамять отводится спе-циальная область памяти, начинающаяся с абсолютного адреса B800h:0000h и заканчивающаяся адресом B800h:FFFFh. Все, что программа запишет в эту область памяти, будет пересылаться в память видеоадаптера и отображаться на экране.

В текстовых режимах для хранения каждого изображенного символа ис-пользуются два байта:

-- байт с ASCII-кодом символа;

-- байт атрибута символа (указывает цвет символа и фона, мигание).

Байт атрибута символа имеет следующий формат (биты):

-- 7 – символ мигает (по умолчанию) или фон яркого цвета (если его действие было переопределено прерыванием 10h).

-- 6 – 4 – цвет фона.

-- 3 – символ яркого цвета (по умолчанию) или фон мигает

Для установки требуемого программе видеорежима используется прерывание 10h (видеосервис) BIOS. Видеорежимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов. В данной лабораторной работе использование графических режимов видеоадаптера не требуется, поэтому в описании прерываний эта информация будет опущена.

2. *Обработка нажатия кнопок клавиатуры.*

Обработка нажатий на клавиатуру может производиться различными спо-собами:

-- с помощью прерываний ввода символов DOS;

-- с помощью прерываний ввода символов BIOS;

-- с помощью прямого доступа к буферу клавиатуры

-- с помощью доступа к портам ввода-вывода клавиатуры.

Ввод символов с помощью функций прерывания DOS 21h рассмотрен ранее в лабораторной работе №2. По сравнению с функциями DOS, прерывание BIOS 16h предоставляет больше возможностей для считывания данных и управления клавиатурой и такой доступ практически эквивалентен по производительности прямому доступу к буферу клавиатуры.

Каждой клавише на клавиатуре соответствует уникальный код, называемый скан-код. Этот код посылается клавиатурой при каждом нажатии и отпускании клавиши и обрабатывается BIOS – записывается в кольцевой буфер клавиатуры.

К буферу клавиатуры также можно обратиться напрямую – буфер находится по адресу 0000h:041Eh и занимает 16 слов, по 0000h:043Dh включи-тельно. Каждый символ хранится в буфере в виде слова, в таком же виде, как возвращает функция 01h прерывания INT 16h.

По адресу 0000h:041Ah находится адрес (ближний) по которому будет расположен следующий введенный символ (указатель на начало буфера), а по адресу 0000h:041Ch лежит адрес конца буфера. Т.к. буфер клавиатуры – за-кольцован, то если эти адреса начала и конца буфера равны, то буфер пуст.

Иногда буфер клавиатуры размещается в другой области памяти, тогда адрес его начала хранится в области данных BIOS по адресу 0480h, а конца – по адресу 0482h.

3. *Доступ к системным часам.*

Персональный компьютер содержит два устройства для управления про-

цессами:

-- часы реального времени (RTC) – имеют автономное питание, используют-ся для чтения/установки текущих даты и времени, установки будильника и для вызова прерывания IRQ8 (INT 4Ah) каждую миллисекунду;

-- системный таймер – используется одновременно для управления контрол-лером прямого доступа к памяти, для управления динамиком и как генератор импульсов, вызывающий прерывание IRQ0 (INT 8h) *18,2* раза в секунду.

Для видеоигры, создаваемой в данной лабораторной работе, указанные выше устройства лучше всего использовать на уровне функций DOS или BIOS как средство для определения текущего времени, организации задержек и фор-мирования случайных чисел.

BIOS отслеживает каждый отсчет системного таймера с помощью своего обработчика прерывания IRQ0 (INT 8h) и увеличивает на 1 значение 32-битного счетчика, который располагается в памяти по адресу 0000h:046Ch, причем при переполнении этого счетчика байт по адресу 0000h:0470h уве-личивается на 1. Программа может считывать значение этого счетчика в цикле (например, просто командой MOV) и таким образом организовывать задержки,

(например, пока ждать пока счетчик не увеличится на 1 (минимальная задержка будет равна приблизительно 55 микросекундам)). Для работы со счетчиком времени в BIOS есть функции

**Код программы (.exe)**

.8086

.model small

.stack 100h

.data

;units

pac\_man equ 02h

ghost equ 40h

coin equ 03h

yellow\_color equ 0Eh

green\_color equ 0Ah

red\_color equ 04h

;score

score dw 0

info\_line db 'S', 3Fh, 'c', 3Fh,'o', 3Fh, 'r', 3Fh, 'e', 3Fh, ':', 3Fh, ' ', 3Fh, '0', 3Fh, 32 dup (' ', 3Fh)

score\_for\_operations dw 0

one\_digit\_of\_score dw 0

digits\_of\_score dw 0

ten db 10

;random

random\_number dw 0

lcg\_mul\_const equ 25173

lcg\_inc\_const equ 13849

;map

position dw 0

wall db ' ', 5Fh

red\_wall db ' ', 47h

black\_space db ' ', 00h

space db ' ', 0Fh

map\_width equ 40

map\_height equ 25

map db 40 dup ('w')

db 'w', 4 dup('s'), 2 dup('w'), 12 dup('s'), 2 dup('w'), 12 dup('s'), 2 dup('w'), 4 dup('s'), 'w'

db 'w', 's', 2 dup('w'), 's', 2 dup('w'), 's', 4 dup('w'), 's', 5 dup('w'), 's', 2 dup('w'), 's', 5 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w', 's', 2 dup('w'), 's', 2 dup('w'), 's', 4 dup('w'), 3 dup('s'), 'w', 8 dup('s'), 'w', 3 dup('s'), 4 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w', 12 dup('s'), 'w', 's', 'w', 's', 6 dup('w'), 's', 'w', 's', 'w', 12 dup('s'), 'w'

db 4 dup('w'), 's', 5 dup('w'), 's', 3 dup('w'), 's', 'w', 's', 6 dup('w'), 's', 'w', 's', 3 dup('w'), 's', 5 dup('w'), 's', 4 dup('w')

db 4 dup('w'), 's', 'w', 5 dup('s'), 'w', 16 dup('s'), 'w', 5 dup('s'), 'w', 's', 4 dup('w')

db 'w', 4 dup('s'), 'w', 's', 3 dup('w'), 3 dup('s'), 'w', 's', 10 dup('w'), 's', 'w', 3 dup('s'), 3 dup('w'), 's', 'w', 4 dup('s'), 'w'

db 'w', 's', 2 dup('w'), 's', 'w', 's', 3 dup('w'), 's', 'w', 's', 'w', 's', 10 dup('w'), 's', 'w', 's', 'w', 's', 3 dup('w'), 's', 'w', 's', 2 dup('w'), 's', 'w'

db 'w', 's', 2 dup('w'), 's', 'w', 's', 2 dup('w'), 2 dup('s'), 'w', 's', 'w', 12 dup('s'), 'w', 's', 'w', 2 dup('s'), 2 dup('w'), 's', 'w', 's', 2 dup('w'), 's', 'w'

db 'w', 9 dup('s'), 2 dup('w'), 's', 'w', 's', 10 dup('w'), 's', 'w', 's', 2 dup('w'), 9 dup('s'), 'w'

db 'w','s', 2 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 's', 'w', 's', 10 dup('w'), 's', 'w', 's', 2 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w','s', 2 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 3 dup('s'), 10 dup('w'), 3 dup('s'), 2 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w', 10 dup('s'), 'w', 's', 'w', 's', 10 dup('w'), 's', 'w', 's', 'w', 10 dup('s'), 'w'

db 'w', 's', 2 dup('w'), 's', 'w', 's', 'w', 's', 'w', 's', 'w', 's', 'w', 12 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 's', 'w', 's', 2 dup('w'), 's', 'w'

db 'w', 's', 2 dup('w'), 's', 'w', 's', 'w', 5 dup('s'), 'w', 's', 10 dup('w'), 's', 'w', 5 dup('s'), 'w', 's', 'w', 's', 2 dup('w'), 's', 'w'

db 'w', 4 dup('s'), 'w', 's', 2 dup('w'), 's', 4 dup('w'), 's', 10 dup('w'), 's', 4 dup('w'), 's', 2 dup('w'), 's', 'w', 4 dup('s'), 'w'

db 4 dup('w'), 's', 'w', 28 dup('s'), 'w', 's', 4 dup('w')

db 4 dup('w'), 's', 5 dup('w'), 's', 3 dup('w'), 's', 'w', 's', 6 dup('w'), 's', 'w', 's', 3 dup('w'), 's', 5 dup('w'), 's', 4 dup('w')

db 'w', 10 dup('s'), 3 dup('w'), 's', 'w', 's', 6 dup('w'), 's', 'w', 's', 3 dup('w'), 10 dup('s'), 'w'

db 'w', 's', 2 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 5 dup('s'), 'w', 8 dup('s'), 'w', 5 dup('s'), 2 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w', 's', 2 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 's', 'w', 's', 5 dup('w'), 's', 2 dup('w'), 's', 5 dup('w'), 's', 'w', 's', 2 dup('w'), 's', 2 dup('w'), 's', 2 dup('w'), 's', 'w'

db 'w', 4 dup('s'), 2 dup('w'), 4 dup('s'), 'w', 16 dup('s'), 'w', 4 dup('s'), 2 dup('w'), 4 dup('s'), 'w'

db 40 dup ('w')

;death screen

death\_screen db 80 dup('w')

db 280 dup('s')

db 3 dup('s'), 'w', 5 dup('s'), 'w', 's', 3 dup('w'), 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 3 dup('w'), 's', 3 dup('w'), 's', 3 dup('w'), 5 dup('s')

db 4 dup('s'), 'w', 3 dup('s'), 'w', 2 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 's', 3 dup('w'), 's', 3 dup('w'), 5 dup('s')

db 5 dup('s'), 'w', 's', 'w', 3 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 's', 2 dup('w'), 2 dup('s'), 'w', 7 dup('s')

db 6 dup('s'), 'w', 4 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 's', 2 dup('w'), 2 dup('s'), 3 dup('w'), 5 dup('s')

db 6 dup('s'), 'w', 4 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 2 dup('s'), 2 dup('w'), 's', 3 dup('w'), 5 dup('s')

db 6 dup('s'), 'w', 4 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 2 dup('s'), 2 dup('w'), 's', 'w', 7 dup('s')

db 6 dup('s'), 'w', 4 dup('s'), 'w', 's', 'w', 's', 'w', 's', 'w', 2 dup('s'), 'w', 3 dup('s'), 'w', 's', 'w', 's', 3 dup('w'), 's', 3 dup('w'), 5 dup('s')

db 6 dup('s'), 'w', 4 dup('s'), 3 dup('w'), 's', 3 dup('w'), 2 dup('s'), 3 dup('w'), 's', 3 dup('w'), 's', 3 dup('w'), 's', 3 dup('w'), 5 dup('s')

db 200 dup('s')

db 80 dup('w')

;player

player\_skin db pac\_man, yellow\_color

player\_direction db ?

player\_position dw 881

;ghosts

ghost\_skin db ghost, green\_color

first\_ghost\_position dw 187

second\_ghost\_position dw 246

third\_ghost\_position dw 823

;coin

coin\_skin db coin, red\_color

coin\_position dw 764

.code

set\_field PROC

mov bx, 0 ;i

mov cx, 25

setting\_rows\_map:

push cx

mov cx, map\_width

mov di, position

add di, di

setting\_cols\_map:

cmp map[bx], 'w'

jne skip1\_map

lea si, wall

movsw

jmp skip2\_map

skip1\_map:

add si, 2

add di, 2

skip2\_map:

inc position

inc bx

loop setting\_cols\_map

pop cx

loop setting\_rows\_map

mov position, 0

ret

set\_field ENDP

set\_death\_screen proc

mov bx, 0 ;i

mov cx, 25

setting\_rows\_screen:

push cx

mov cx, map\_width

mov di, position

add di, di

setting\_cols\_screen:

cmp death\_screen[bx], 'w'

jne skip1\_screen

lea si, red\_wall

movsw

jmp skip2\_screen

skip1\_screen:

lea si, black\_space

movsw

skip2\_screen:

inc position

inc bx

loop setting\_cols\_screen

pop cx

loop setting\_rows\_screen

mov position, 0

ret

set\_death\_screen endp

set\_field\_info PROC

push cx

mov cx, map\_width

add cx, cx

lea si, info\_line

mov di, 1920

rep movsb

pop cx

ret

set\_field\_info ENDP

set\_player\_and\_ghosts PROC

lea si, player\_skin

mov di, player\_position

add di, di

movsw

lea si, ghost\_skin

mov di, first\_ghost\_position

add di, di

movsw

lea si, ghost\_skin

mov di, second\_ghost\_position

add di, di

movsw

lea si, ghost\_skin

mov di, third\_ghost\_position

add di, di

movsw

ret

set\_player\_and\_ghosts ENDP

set\_coin PROC

lea si, coin\_skin

mov di, coin\_position

add di, di

movsw

ret

set\_coin ENDP

clear\_player\_position PROC

lea si, black\_space

mov di, player\_position

add di, di

movsb

ret

clear\_player\_position ENDP

clear\_ghost\_position PROC

lea si, black\_space

mov di, bx

add di, di

movsb

ret

clear\_ghost\_position ENDP

inc\_score PROC

inc score

ret

inc\_score ENDP

set\_score proc

push cx

mov bx, score

mov score\_for\_operations, bx

mov digits\_of\_score, 0

mov ax, score\_for\_operations

mul ten

mov score\_for\_operations, ax

setting\_digits\_of\_score:

inc digits\_of\_score

mov ax, score\_for\_operations

div ten

xor ah,ah

mov score\_for\_operations, ax

cmp score\_for\_operations, 10

jae setting\_digits\_of\_score

mov di,1932

add di, digits\_of\_score

add di, digits\_of\_score

mov ax, score

mov score\_for\_operations, ax

writing\_score:

mov ax, score\_for\_operations

div ten

mov bx, ax

xor bh, bh

mov score\_for\_operations, bx

mov al, ah

xor ah, ah

mov one\_digit\_of\_score, ax

add one\_digit\_of\_score, 48

dec digits\_of\_score

mov si, offset one\_digit\_of\_score

movsb

sub di, 3

cmp digits\_of\_score, 0

jne writing\_score

pop cx

ret

set\_score endp

random PROC

mov ax, lcg\_mul\_const

mul random\_number

add ax, lcg\_inc\_const

mov random\_number, ax

xor dx, dx

mov cx, 1000

div cx

ret

random ENDP

generate\_coin PROC

push cx

generating:

mov cx, 0

mov coin\_position, cx

call random

cmp dx, 919

jae generating

mov coin\_position, dx

mov bx, coin\_position

cmp map[bx], 'w'

je generating

pop cx

ret

generate\_coin ENDP

ghosts\_movements PROC

moving\_first\_ghost:

mov bx, first\_ghost\_position

call clear\_ghost\_position

call random

cmp dx, 750

jae try\_up\_first

cmp dx, 500

jae try\_down\_first

cmp dx, 250

jae try\_right\_first

sub bx, 1

cmp map[bx], 'w'

je moving\_first\_ghost

jmp moved\_first

try\_up\_first:

sub bx, 40

cmp map[bx], 'w'

je moving\_first\_ghost

jmp moved\_first

try\_down\_first:

add bx, 40

cmp map[bx], 'w'

je moving\_first\_ghost

jmp moved\_first

try\_right\_first:

add bx, 1

cmp map[bx], 'w'

je moving\_first\_ghost

jmp moved\_first

moved\_first:

mov first\_ghost\_position, bx

moving\_second\_ghost:

mov bx, second\_ghost\_position

call clear\_ghost\_position

call random

cmp dx, 750

jae try\_up\_second

cmp dx, 500

jae try\_down\_second

cmp dx, 250

jae try\_right\_second

sub bx, 1

cmp map[bx], 'w'

je moving\_second\_ghost

jmp moved\_second

try\_up\_second:

sub bx, 40

cmp map[bx], 'w'

je moving\_second\_ghost

jmp moved\_second

try\_down\_second:

add bx, 40

cmp map[bx], 'w'

je moving\_second\_ghost

jmp moved\_second

try\_right\_second:

add bx, 1

cmp map[bx], 'w'

je moving\_second\_ghost

jmp moved\_second

moved\_second:

mov second\_ghost\_position, bx

moving\_third\_ghost:

mov bx, third\_ghost\_position

call clear\_ghost\_position

call random

cmp dx, 750

jae try\_up\_third

cmp dx, 500

jae try\_down\_third

cmp dx, 250

jae try\_right\_third

sub bx, 1

cmp map[bx], 'w'

je moving\_third\_ghost

jmp moved\_third

try\_up\_third:

sub bx, 40

cmp map[bx], 'w'

je moving\_third\_ghost

jmp moved\_third

try\_down\_third:

add bx, 40

cmp map[bx], 'w'

je moving\_third\_ghost

jmp moved\_third

try\_right\_third:

add bx, 1

cmp map[bx], 'w'

je moving\_third\_ghost

jmp moved\_third

moved\_third:

mov third\_ghost\_position, bx

ret

ghosts\_movements ENDP

death:

call set\_death\_screen

mov ax, 4C00h

int 21h

start:

mov ax, @data

mov ds, ax

mov ah, 00h

mov al, 01h

int 10h

mov ah, 01h

mov cx, 2607h

int 10h

mov ax, 0B800h

mov es, ax

call set\_field

call set\_field\_info

call set\_player\_and\_ghosts

call set\_coin

main\_cycle:

mov cx, 0

mov ah, 86h

mov dx, 65535

int 15h

int 15h

int 15h

mov dx, first\_ghost\_position

cmp player\_position, dx

je death

mov dx, second\_ghost\_position

cmp player\_position, dx

je death

mov dx, third\_ghost\_position

cmp player\_position, dx

je death

mov dx, coin\_position

cmp player\_position, dx

jne skip\_coin\_operations

call inc\_score

call generate\_coin

call set\_coin

call set\_score

skip\_coin\_operations:

call ghosts\_movements

call set\_player\_and\_ghosts

mov ah, 01h

int 16h

jz no\_key

mov ah, 00h

int 16h

cmp ah, 11h

jne not\_w

mov player\_direction, 'w'

jmp no\_key

not\_w:

cmp ah, 1Fh

jne not\_s

mov player\_direction, 's'

jmp no\_key

not\_s:

cmp ah, 1Eh

jne not\_a

mov player\_direction, 'a'

jmp no\_key

not\_a:

cmp ah, 20h

jne not\_d

mov player\_direction, 'd'

jmp no\_key

not\_d:

no\_key:

cmp player\_direction, 'w'

je is\_w

cmp player\_direction, 's'

je is\_s

cmp player\_direction, 'a'

je is\_a

cmp player\_direction, 'd'

je is\_d

jmp main\_cycle

is\_w:

mov si, player\_position

sub si, 40

cmp map[si], 'w'

je wall\_

call clear\_player\_position

sub player\_position, 40

jmp wall\_

is\_s:

mov si, player\_position

add si, 40

cmp map[si], 'w'

je wall\_

call clear\_player\_position

add player\_position, 40

jmp wall\_

is\_a:

mov si, player\_position

sub si, 1

cmp map[si], 'w'

je wall\_

call clear\_player\_position

sub player\_position, 1

jmp wall\_

is\_d:

mov si, player\_position

add si, 1

cmp map[si], 'w'

je wall\_

call clear\_player\_position

add player\_position, 1

jmp wall\_

wall\_:

call set\_player\_and\_ghosts

jmp main\_cycle

mov ax, 4C00h

int 21h

end start

**Вывод программы**

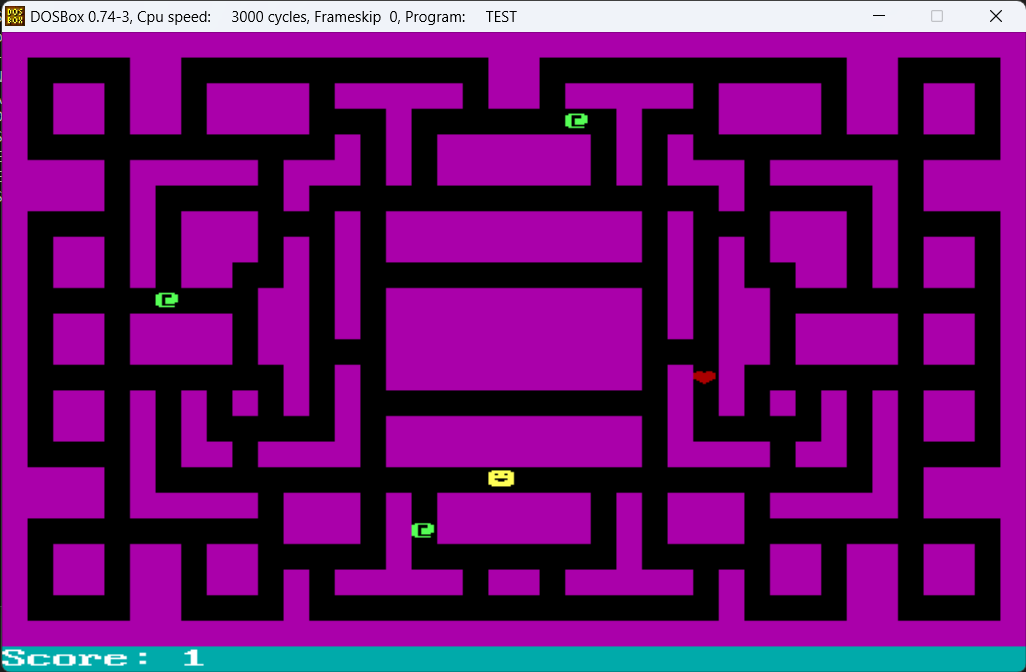


Рисунок 1 – Результат работы программы (игра)



Рисунок 2 – Результат работы программы (проигрыш)