МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

Отчет по предмету
Проектирование на языке ассемблера
Вариант 1

Лабораторная работа 8
«Создание видеоигры»

Выполнил:
Студент группы 150503
Шарай П.Ю.

Проверил:
Туровец Н.О.

Минск 2022

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 No2, 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 преры- вания, а также позволяет формировать в консоли определенные эффекты, часто не используемые в режиме вывода в позицию курсора.
В большинстве текстовых видеорежимов под видеопамять отводится специальная 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 используются 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 (если его дей-
25
ствие было переопределено прерыванием 11h). -- 2 – 0: цвет символа.
Кодировка цветов приведена в таблице 3.
Таким образом, по адресу B800h:0000h лежит байт с кодом символа, находящимся в верхнем левом углу экрана; по адресу B800h:0001h лежит атрибут этого символа; по адресу B800h:0002h лежит код второго символа в верхней строке экрана и т.д.
Коды цветов в текстовых видеорежимах

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.
--Прерывание BIOS 10h, функция00–установитьвидеорежим:
-- Прерывание BIOS 11h – конфигурация оборудования:
BIOS: --ПрерываниеBIOS 10h,функция01 –установитьразмеркурсора:

-- Прерывание BIOS 10h, функция 03 – получить положение и размер курсора (каждая страница использует собственный независимый курсор):
-- Прерывание BIOS 10h, функция 06 – прокрутка экрана вверх (вставка чистых строк снизу):
-- Прерывание BIOS 10h, функция 07 – прокрутка экрана вниз (вставка чистых строк сверху):
Прерывание 10h также обеспечивает функции вывода данных на уровне BIOS:
-- Прерывание BIOS 10h, функция 08 – считать символ и атрибут символа в текущей позиции курсора:
-- Прерывание BIOS 10h, функция 09 – вывести символ с заданным атрибутом на экран:
-- Прерывание BIOS 10h, функция 0Ah – вывести символ с текущим атрибутом на экран:
-- Прерывание BIOS 10h, функция 0Eh – вывести символ в режиме телетайпа:
-- Прерывание BIOS 10h, функция 13h – вывести строку символов с заданными атрибутами:
2. Обработка нажатия кнопок клавиатуры.
Обработка нажатий на клавиатуру может производиться различными способами:
-- 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 No2. 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.
Функции прерывания 16h:
-- Прерывание BIOS 16h, функция 00h (10h, 20h) – чтение символа с ожиданием:
-- Прерывание BIOS 16h, функция 01h (11h, 21h) – проверка наличия символа в буфере:
-- Прерывание BIOS 16h, функция 02h (12h, 22h) – получить состояние клавиатуры:
Байт состояния клавиатуры 1 (расположен в памяти DOS по адресу 0000h:0417h или 0040h:0017h):
-- бит 7 -- бит 6 -- бит 5 -- бит 4 -- бит 3
12h/22h), -- бит 2 -- бит 1 -- бит 0

– Ins включена,
– CapsLock включена, – NumLock включена, – ScrollLock включена,
– Alt нажата (любая Alt для функции 02h, и только левая Alt для
– Ctrl нажата (любая Ctrl), – левая Shift нажата,
– правая Shift нажата.
состояния клавиатуры 2 (расположен в памяти DOS по адресу -- бит 7 – SysRq нажата,
Байт
0000h:0418h или 0040h:0018h):
31
-- бит 6 – CapsLock нажата, -- бит 5 – NumLock нажата, -- бит 4 – ScrollLock нажата, -- бит 3 – правая Alt нажата, -- бит 2 – правая Ctrl нажата, -- бит 1 – левая Alt нажата, -- бит 0 – левая Ctrl нажата.
Т.к. эти байты расположены в памяти по фиксированному адресу, то вместо вызова прерывания удобнее просто считывать и даже перезаписывать значения этих байт напрямую, что изменит состояние клавиатуры.

К буферу клавиатуры также можно обратиться напрямую – буфер нахо- дится 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-
32
цессами:
-- часы реального времени (RTC) – имеют автономное питание, используют- ся для чтения/установки текущих даты и времени, установки будильника и для вызоваprерыванияIRQ8(INT 4Ah)каждуюмиллисекунду;
-- системный таймер – используется одновременно для управления контрол- лером прямого доступа к памяти, для управления динамиком и как генератор импульсов,вызывающийprерываниеIRQ0(INT 8h)18,2разавсекунду.
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.
Управление часами RTC и внутренними часами операционной системы средствами DOS:
-- Функция DOS 2Ah (INT 21h) – считать дату: Ввод: AH = 2Ah

Вывод: CX = год (1980 – 2099), DH =месяц,
DL =день,
AL =деньнедели(0–воскресенье,1–понедельникит.п.).
-- Функция DOS 2Bh (INT 21h) – установить дату:
Ввод:
AH = 2Bh,
CX =год(1980–2099), DH =месяц,
DL =день.
Вывод:
-- Функция DOS 2Ch (INT 21h) – считать время:
Если введена несуществующая дата, то AH Если дата установлена, то AH = 00h.
= FFh,
Ввод: AH = 2Ch
Вывод: CH = час,
CL =минута,
DH =секунда,
DL = сотая доля секунды.
-- Функция DOS 2Dh (INT 21h) – установить время:
Ввод:
Вывод:
AH = 2Dh,
CH =час,
CL =минута,
DH =секунда,
DL = сотая доля секунды.
Если введено несуществующее время, то AL
= FFh,
33
Ввод:
AH = 03h,
CH = час (в формате BCD),
CL = минута (в формате BCD),
DH = секунда (в формате BCD),
DL = 01h - если используется летнее время, DL = 00h - если не используется летнее время.
Если время установлено, то AL = 00h.
BIOS позволяет управлять часами PTC напрямую:
-- Прерывание BIOS 1Ah, функция 02h – считать время RTC:
Ввод: Вывод:
AH = 02h
CF = 1 – если часы не работают или попытка чтения пришлась на момент обновления,

0 – если время успешно считано, то: час (в формате BCD),
минута (в формате BCD),
секунда (в формате BCD),
CF =
CH =
CL =
DH =
DL = 01h – если действует летнее время,
DL = 00h - если не действует летнее время.
-- Прерывание BIOS 1Ah, функция 03h – установить время RTC:
-- Прерывание BIOS 1Ah, функция 04h – считать дату RTC:
Ввод: Вывод:
AH = 04h
CF = 1 – если часы не работают или попытка чтения пришлась на момент обновления,
CF = 0 – если дата успешно считана, то:
CX = год (в формате BCD, например, 1998h для 1998-го года),
DH = месяц (в формате BCD), DL = день (в формате BCD).
-- Прерывание BIOS 1Ah, функция 05h – установить дату RTC: Ввод: AH = 05h,
CX = год (в формате BCD), DH =месяц,
DL =день.
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 увеличивается 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:
-- Прерывание BIOS 1Ah, функция 00h – прочитать значение счетчика времени:
Ввод: AH = 00h.
Вывод: CX:DX = значение счетчика,
AL = байт переполнения счетчика.
--Прерывание BIOS 1Ah, функция 01h– установить значение счетчика времени:
Ввод: AH = 01h,
CX:DX =значениесчетчика.
Для изменения частоты работы таймера, BIOS имеет специальные функ- ции:
-- Прерывание BIOS 15h, функция 86h – формирование задержки таймера:
Ввод: Вывод:

AH = 86h,
CX:DX =длительностьзадержкивмикросекундах.
Если таймер был занят, то CF = 1,
Если задержка выполнена, то CF = 0:
AL = маска, записанная обработчиком в регистр управления преры- ваниями.
-- Прерывание BIOS 15h, функция 83h – управление работой счетчика:
Ввод:
Вывод:
AH = 83h,
AL = 1–прервать счетчик,
AL = 0–запустить счетчик:
CX:DX =длительность задержки в микросекундах,
ES:BX = адрес байта, старший бит которого по окончании работы счетчика будет установлен в 1.
Если таймер был занят, то CF = 1,
Если задержка выполнена, то CF = 0:
AL = маска, записанная обработчиком в регистр управления прерываниями.

Код программы:

```
.model small
.stack 100h
.data

KUpSpeed    equ 48h
KDownSpeed  equ 50h
KMoveUp     equ 11h
KMoveDown   equ 1Fh
KMoveLeft   equ 1Eh
KMoveRight  equ 20h
KExit       equ 01h

xSize       equ 80
ySize       equ 25
xField      equ 50
yField      equ 21
videoBufferCellSize equ 2
scoreSize equ 4

videoStart   dw 0B800h
dataStart    dw 0000h
timeStart    dw 0040h
timePosition dw 006Ch

space equ 0020h
spaceBlue equ 4020h ; (3020h), 4020h-red,
snakeBodySymbol    equ 0A11h ; 0A10h
appleSymbol        equ 0B25h      ;0B15h
VWallSymbol        equ 40BAh      ;30BAh
HWallSymbol        equ 40CDh      ;30CDh
BWallSymbol        equ 3023h      ;4023h

LeftTopWallSpecialSymbol equ 40C9h  ;30C9h
LeftBottomWallSpecialSymbol equ 40C8h  ;30C8h
RightTopWallSpecialSymbol equ 40BBh    ;30BBh
RightBottomWallSpecialSymbol equ 40BCh  ;30BCh

fieldSpacingBad equ spaceBlue, VWallSymbol, xField dup(space)
fieldSpacing equ fieldSpacingBad, VWallSymbol

screen     dw xSize dup(spaceBlue)
```

```
            dw spaceBlue, LeftTopWallSpecialSymbol, xField dup(HWallSymbol), RightTopWallSpecialSymbol, 27 dup
(spaceBlue)
      dw fieldSpacing, spaceBlue, 3053h, 3063h, 306Fh, 3072h, 3065h, 303Ah, spaceBlue
score      dw scoreSize dup(3030h), xSize - xField - scoreSize - 11 dup(spaceBlue)
      dw fieldSpacing, spaceBlue, 3053h, 3070h, 2 dup(3065h), 3064h, 303Ah, spaceBlue
speed      dw 3031h, 18 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3043h, 306Fh, 306Eh, 3074h, 3072h, 306Fh, 306Ch, 3073h, 303Ah, 17
dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3057h, spaceBlue, 30C4h, spaceBlue, 3055h, 3070h, 3018h, 19
dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3053h, spaceBlue, 30C4h, spaceBlue, 3044h, 306Fh, 3077h, 306Eh,
3019h, 17 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3041h, spaceBlue, 30C4h, spaceBlue, 304Ch, 3065h, 3066h, 3074h,
301Bh, 17 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3044h, spaceBlue, 30C4h, spaceBlue, 3052h, 3069h, 3067h, 3068h,
3074h, 301Ah, 16 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3045h, 3073h, 3063h, spaceBlue, 30C4h, spaceBlue, 3045h, 3078h,
3069h, 3074h, 3021h, xSize - xField - 15 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3018h, spaceBlue, 30C4h, spaceBlue, 3053h, 3070h, 3065h, 3065h,
3064h, spaceBlue, 3075h, 3070h, spaceBlue, 13 dup(spaceBlue)
                  dw fieldSpacing, spaceBlue, 3019h, spaceBlue, 30C4h, spaceBlue, 3053h, 3070h, 3065h, 3065h,
3064h, spaceBlue, 3064h, 306Fh, 3077h, 306Eh, 12 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw fieldSpacing, xSize - xField - 3 dup(spaceBlue)
                  dw spaceBlue, LeftBottomWallSpecialSymbol, xField dup(HWallSymbol),
RightBottomWallSpecialSymbol, 27 dup(spaceBlue)
      dw xSize dup(spaceBlue)


widthOfEndScreen   equ 20    ;
allWidth       equ 80    ;
black          equ 0020h  ;
white          equ 4020h  ;
black          equ 0020h  ;

blackVWallSymbol     equ 00FBAh
blackHWallSymbol     equ 00FCDh

theEnd      dw LeftTopWallSpecialSymbol, widthOfEndScreen-2 dup(HWallSymbol), RightTopWallSpecialSymbol
                        dw VWallSymbol, 18 dup(black), VWallSymbol
                        dw VWallSymbol, 5 dup(black) ,0F47h,0F61h,0F6Dh,0F65h,space,
0F6Fh,0F76h,0F65h,0F72h, 4 dup(black), VWallSymbol
                        dw VWallSymbol, 18 dup(black), VWallSymbol
                        dw LeftBottomWallSpecialSymbol, widthOfEndScreen-2 dup(HWallSymbol),
RightBottomWallSpecialSymbol

snakeMaxSize equ 30
snakeSize db 3
PointSize equ 2

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

brickWallSize equ 5

brickWall1 dw 0202h,  0201h, 0200h,  01FFh, 01FEh
brickWall2 dw 0202h,  0102h, 0002h, 0FF02h, 0FE02h
brickWall3 dw 01FEh,  00FEh, 0FFFEh, 0FEFEh, 0FDFEh
brickWall4 dw 0FE02h,  0FE01h, 0FE00h, 0FDFFh, 0FDFEh

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

clearScreen MACRO
            push ax
            mov ax, 0003h
            int 10h
            pop ax
ENDM


main:
            mov ax, @data
            mov ds, ax
            mov dataStart, ax
            mov ax, videoStart
            mov es, ax
            xor ax, ax

            clearScreen
            call HideCursor

            call initAllScreen

            call mainGame

to_close:
            call printEndScreen
            mov ah,7h
   int 21h

esc_exit:

            clearScreen

            mov ah, 4ch
            int 21h

GetTimerValue MACRO
            push ax

            mov ax, 00h
            int 1Ah

            pop ax
ENDM


printEndScreen PROC
            push es
            push 0B800h

            pop es

            mov di, 20*80 + 50
            mov si, offset theEnd
            mov cx, 5
            cld
loopPrintEndScreen:

            push cx
```

```
            mov cx, widthOfEndScreen
            rep movsw

            add di, 2*(allWidth - widthOfEndScreen)

            pop cx
            loop loopPrintEndScreen
    std
            pop es
            ret
ENDP

HideCursor proc
    push ax
    push bx
    push cx
        mov ah,3
        mov bh,0
        int 10h
        mov ch,20h
        mov ah,1
        int 10h
        pop cx
        pop bx
        pop ax
        ret
endp

drawBrickWall PROC
 push cx
 push bx
 mov cx, brickWallSize

 mov si, offset brickWallTrue
 loopBrickWall:
            mov bx, [si]
            add si, PointSize

            call CalcOffsetByPoint

            mov di, bx

            mov ax, BWallSymbol
            stosw
            loop loopBrickWall
 pop bx
 pop cx
 ret
ENDP

destroyWall PROC
    push cx
    mov cx, brickWallSize

    mov si, offset brickWallTrue
loopDestroyWall:
            mov bx, [si]
            add si, PointSize
            call CalcOffsetByPoint
            mov di, bx
            mov ax, space
            stosw
            loop loopDestroyWall
    pop cx
    ret
ENDP

initAllScreen PROC
            mov si, offset screen
            xor di, di
            mov cx, xSize*ySize
```

```
        rep movsw

        xor ch, ch
        mov cl, snakeSize
        mov si, offset snakeBody

loopInitSnake:
        mov bx, [si]
        add si, PointSize

        call CalcOffsetByPoint

        mov di, bx

        mov ax, snakeBodySymbol
        stosw

        loop loopInitSnake

        call GenerateRandomApple

        ret
ENDP

CalcOffsetByPoint PROC

        push ax
        push dx

        xor ah, ah
        mov al, bl
        mov dl, xSize
        mul dl
        mov dl, bh
        xor dh, dh
        add ax, dx
        mov dx, videoBufferCellSize
        mul dx
        mov bx, ax

        pop dx
        pop ax
        ret
ENDP

MoveSnake PROC
        push ax
        push bx
        push cx
        push si
        push di
        push es

   xor ax, ax
        mov al, snakeSize
        mov cx, ax
        mov bx, PointSize
        mul bx
        mov di, offset snakeBody
        add di, ax
        mov si, di
        sub si, PointSize

        push di

        mov es, videoStart
        mov bx, ds:[si]
        call CalcOffsetByPoint
        mov di, bx
        mov ax, space
        stosw
```

```
        pop di

        mov es, dataStart
        std
        rep movsw
        mov bx, snakeBody

        add bh, Bmoveright
        add bl, Bmovedown
        mov snakeBody, bx

        pop es
        pop di
        pop si
        pop cx
        pop bx
        pop ax
        ret
ENDP

mainGame PROC
        push ax
        push bx
        push cx
        push dx
        push ds
        push es

checkAndMoveLoop:
        mov ah, 01h
        int 16h
        jnz checkEnteredSymbol
        jmp noSymbolInBuff

checkEnteredSymbol:
   mov ah, 00h
        int 16h
        cmp ah, KExit
        jne skipJmp

        jmp esc_exit

skipJmp:
        cmp ah, KMoveLeft
        je setMoveLeft

        cmp ah, KMoveRight
        je setMoveRight

        cmp ah, KMoveUp
        je setMoveUp

        cmp ah, KMoveDown
        je setMoveDown

        cmp ah, KUpSpeed
        je setSpeedUp

        cmp ah, KDownSpeed
        je setSpeedDown

        jmp noSymbolInBuff

setMoveLeft:
   mov al, Bmoveright
   cmp al, forwardVal
   jne setMoveLeft_ok
   jmp noSymbolInBuff

   setMoveLeft_ok:

        mov Bmoveright, backwardVal
```

```
            mov Bmovedown,  stopVal
            jmp noSymbolInBuff

setMoveRight:
   mov al, Bmoveright
   cmp al, backwardVal
   jne setMoveRight_ok
   jmp noSymbolInBuff

   setMoveRight_ok:

            mov Bmoveright, forwardVal
            mov Bmovedown, stopVal
            jmp noSymbolInBuff

setMoveUp:
   mov al, Bmovedown
   cmp al, forwardVal
   jne setMoveUp_ok
   jmp noSymbolInBuff

   setMoveUp_ok:

            mov Bmoveright, stopVal
            mov Bmovedown, backwardVal
            jmp noSymbolInBuff

setMoveDown:
   mov al, Bmovedown
   cmp al, backwardVal
   jne setMoveDown_ok
   jmp noSymbolInBuff

   setMoveDown_ok:

            mov Bmoveright, stopVal
            mov Bmovedown, forwardVal
            jmp noSymbolInBuff

setSpeedUp:
            mov ax, waitTime
            cmp ax, minWaitTime
            je noSymbolInBuff

            sub ax, deltaTime
            mov waitTime, ax

            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

noSymbolInBuff:
            call MoveSnake
```

```
        mov bx, snakeBody

checkSymbolAgain:
        call CalcOffsetByPoint

        mov es, videoStart
        mov ax, es:[bx]

        cmp ax, appleSymbol
        je AppleIsNext

        cmp ax, snakeBodySymbol
        je SnakeIsNext

        cmp ax, HWallSymbol
        je PortalUpDown

        cmp ax, VWallSymbol
        je PortalLeftRight

        cmp ax, BWallSymbol
        je SnakeIsNext

        jmp GoNextIteration

AppleIsNext:
   call destroyWall
        call incSnake
        call GenerateRandomApple
        call incScore
        jmp GoNextIteration
SnakeIsNext:
        jmp endLoop
PortalUpDown:
        mov bx, snakeBody
        sub bl, yField
        cmp bl, 0
        jg writeNewHeadPos

        add bl, yField*2

writeNewHeadPos:
        mov snakeBody, bx
        jmp checkSymbolAgain

PortalLeftRight:
        mov bx, snakeBody
        sub bh, xField
        cmp bh, 0
        jg writeNewHeadPos

        add bh, xField*2
        jmp writeNewHeadPos

GoNextIteration:
        mov bx, snakeBody
        call CalcOffsetByPoint
        mov di, bx
        mov ax, snakeBodySymbol
        stosw

        call Sleep

        jmp checkAndMoveLoop

endLoop:
        pop es
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
```

```
        ret
ENDP

Sleep PROC
        push ax
        push bx
        push cx
        push dx

        GetTimerValue

        add dx, waitTime
        mov bx, dx

checkTimeLoop:
        GetTimerValue
        cmp dx, bx
        jl checkTimeLoop

        pop dx
        pop cx
        pop bx
        pop ax
        ret
ENDP

GenerateRandomApple PROC
        push ax
        push bx
        push cx
        push dx
        push es

        mov ah, 2Ch
        int 21h

        mov al, dl
   mul dh

        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
```

```
            push ax
            movswToTemplate:
               mov ax, [si]
               mov [di],ax
               add di, 2
               add si, 2
            loop movswToTemplate
            pop ax

generateRandomApplePosition:
            mov ah, 2Ch
            int 21h

            mov al, dl
            mul dh

            xor dx, dx
            mov cx, xField
            div cx
            add dx, 2
            mov bh, dl

            xor dx, dx
            mov cx, yField
            div cx
            add dx, 2
            mov bl, dl

   push bx
            call CalcOffsetByPoint
            mov es, videoStart
            mov ax, es:[bx]
   pop bx

            cmp ax, space
            jne generateRandomApplePosition

   mov cx, brickWallSize
   mov si, offset brickWallTemplate

   checkWallPlace:
      push bx
               add bx, [si]

      push bx
               call CalcOffsetByPoint
               mov es, videoStart
               mov ax, es:[bx]
      pop bx

      pop bx

               cmp ax, space

               jne generateRandomApplePosition

               add si, PointSize
            loop checkWallPlace

   mov cx, brickWallSize
   mov si, offset brickWallTemplate
   mov di, offset brickWallTrue

            push ax
            copyTrueCoordinateOfWall:
               mov ax, [si]
               add ax, bx
               mov [di], ax

               add si, PointSize
               add di, PointSize
```

```
            loop copyTrueCoordinateOfWall
            pop ax

            call drawBrickWall

            push bx
            call CalcOffsetByPoint
            mov es, videoStart
            mov ax, appleSymbol;
            mov es:[bx], ax
    pop bx

            pop es
            pop dx
            pop cx
            pop bx
            pop ax
            ret
ENDP

incSnake PROC
            push ax
            push bx
            push di
            push es

            mov al, snakeSize
            cmp al, snakeMaxSize
            je return
            inc al
            mov snakeSize, al
            dec al

            mov bl, PointSize
            mul bl

            mov di, offset snakeBody
            add di, ax

            mov es, dataStart
            mov bx, es:[di]

            call CalcOffsetByPoint
            mov es, videoStart
            mov es:[bx], snakeBodySymbol

return:
            pop es
            pop di
            pop bx
            pop ax
            ret
ENDP

incScore PROC
            push ax
            push es
            push si
            push di
            mov es, videoStart
            mov cx, scoreSize
            mov di, offset score + (scoreSize - 1) * videoBufferCellSize - offset screen

loop_score:
            mov ax, es:[di]
            cmp al, '9'
            jne nineNotNow

            sub al, 9
            mov es:[di], ax

            sub di, videoBufferCellSize
```

```
            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
```

Результат работы программы: