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»

Вариант 2

Выполнил студент гр. 150502: Альхимович Н.Г.

Проверил:        Туровец Н.О.

Минск 2022

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.

Вариант задания:

Игра «Тетрис».

Цель: заполнение случайными падающими сверху вниз фигурами (например, 3 вида фигур) игрового поля, игрок может сдвигать и вращать фигуру, если нижняя строка заполнена полностью, то она очищается (все сдвигается на строку вниз).

Окончание: проигрыш – нет места для новой фигуры, выигрыш – нет. Информация: счет.

Усложнение: смена уровня (увеличение скорости), увеличение числа видов фигур.

Теоретические сведения:

1. Прямой доступ к видеопамяти.

Кроме использования прерываний DOS, описанных в лабораторной рабо- те No2, программа может выводить текст на экран с помощью пересылки дан- ных в специальную область памяти, связанную с видеоадаптером – видеопа- мять. 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 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:

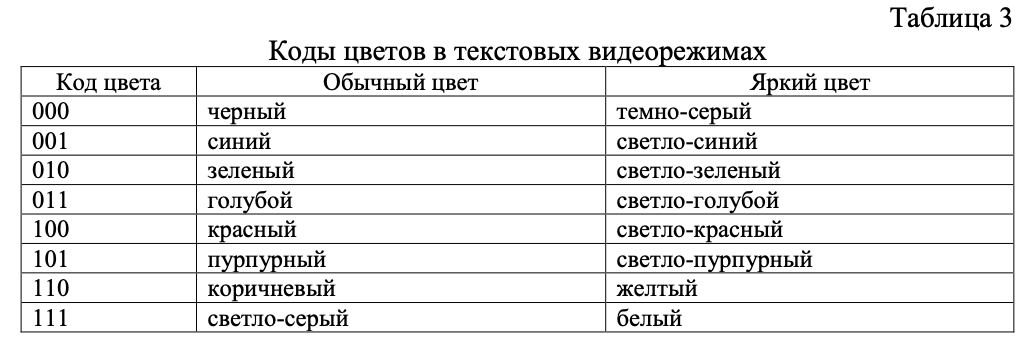
-- байт с ASCII-кодом символа;  
-- байт атрибута символа (указывает цвет символа и фона, мигание).

Байт атрибута символа имеет следующий формат (биты):  
-- 7 – символ мигает (по умолчанию) или фон яркого цвета (если его дей-

ствие было переопределено прерыванием 10h).  
-- 6 – 4 – цвет фона.  
-- 3 – символ яркого цвета (по умолчанию) или фон мигает (если его действие 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.



Фрагмент программы использующей прямой доступ к видеопамяти:

page26image116324160page26image116323952page26image116323744

...

mov cx, output\_line\_size ; число байт в строке - в СХ

push 0B800h

pop es ; адрес в видеопамяти

mov di,word ptr start\_position ; в ES:DI

mov si,offset output\_line ; адрес строки в DS:SI

cld

rep movsb ; скопировать строку

...

start\_position dw 0 ; позиция символа на экране

; строка " 00:00 " с атрибутом символа 1Fh (белый на синем фоне)

output\_line db ' ',1Fh,

db '0',1Fh,'0',1Fh,':',1Fh

db '0',1Fh,'0',1Fh,' ',1Fh

output\_line\_size equ 14

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

--ПрерываниеBIOS 10h,функция00–установитьвидеорежим: Ввод: AH = 00,

AL =номеррежимавмладших7битах:  
-- 00 – 40х25 черно-белый текстовый режим;  
-- 01 – 40х25 стандартный 16-цветный текстовый режим;

Вывод:

-- 02 – 80 х 25 черно-белый текстовый режим;  
-- 03 – 80 х 25 стандартный 16-цветный текстовый режим; -- 07 – 80 х 25 черно-белый стандартный монохромный.

Видеорежим номер 3 используется в DOS по умолчанию. Если старший бит AL не установлен в 1, то экран очищается.

Обычно отсутствует, но некоторые BIOS помещают в AL 30Н для

текстовых режимов и 20h для графических  
Следующий пример показывает установку стандартного 16-цветового

текстового режима:

MOV AH,00 ; функция установки видеорежима

MOV AL,03 ; цветной текст 80х25 с очисткой экрана

INT 10H ; вызвать прерывание BIOS

-- Прерывание BIOS 11h – конфигурация оборудования:

Ввод: Вывод:

AH = 11

AX = состояние оборудования (биты 5 и 4 указывают текущий ви- деорежим):

-- 00 – не используется,  
-- 01 – 40х25 цветной режим,  
-- 10 – 80х25 цветной режим,  
-- 11 – 80х25 черно-белый режим.

Курсор не является символом из набора ASCII-кодов. Компьютер имеет собственное аппаратное обеспечение для управления видом курсора. Обычно символ курсора похож на символ подчеркивания и всегда мерцает.

Для работы с курсором используются следующие функции BIOS:

--Прерывание BIOS 10h, функция 01 –установить размер курсора:

DL = номерстолбца(считаяот0).  
Отсчет номера строки и столбца ведется от верхнего левого угла экрана (символ в левой верхней позиции имеет координаты 0, 0). Номера страниц 0 – 3 (для видеорежимов 2 и 3) и 0 – 7 (для видеоре- жимов 1 и 2) соответствуют области памяти, содержимое которой в данный момент отображается на экране.  
Можно вывести текст в неактивную в настоящий момент страницу, а затем переключиться на нее, чтобы изображение изменилось мгно- венно.

-- Прерывание BIOS 10h, функция 03 – получить положение и размер кур- сора (каждая страница использует собственный независимый курсор):

Ввод: Вывод:

АН = 03,  
ВН = номер страницы.

DH = строка текущей позиции курсора (считая от 0), DL = столбец текущей позиции курсора (считая от 0), СН = первая строка размера курсора,  
CL = последняя строка размера курсора.

-- Прерывание BIOS 10h, функция 05 – установить активную страницу(доступно для цветных текстовых режимов):  
Ввод: АН = 05,

AL = номер страницы (обычно 0).  
Для режима 40х25 можно устанавливать до 8 страниц (от 0 до 7), а для режима 80х25 – до 4 страниц (от 0 до 3).

Вывод символов на неактивную страницу, а затем установка ее активной позволяют формировать более естественный и плавный вывод данных.

-- Прерывание BIOS 10h, функция 06 – прокрутка экрана вверх (вставка чистых строк снизу):

Ввод:

АН = 06,  
Если AL = 00 – прокрутка всего экрана с заполнением пробелами (очистка), иначе AL = число строк для прокрутки вверх,  
BH =атрибут вставляемого символа,  
CH = строка верхнего левого угла окна (считая от 0),  
CL = столбец верхнего левого угла окна (считая от 0),  
DH = строка нижнего правого угла окна (считая от 0),  
DL = столбец нижнего правого угла окна (считая от 0).

-- Прерывание BIOS 10h, функция 07 – прокрутка экрана вниз (вставка чи- стых строк сверху):  
Ввод: АН = 07,

Если AL = 00 – прокрутка всего экрана с заполнением пробелами

(очистка), иначе AL = число строк для прокрутки вниз, BH =атрибут вставляемого символа,  
CH = строка верхнего левого угла окна (считая от 0), CL = столбец верхнего левого угла окна (считая от 0), DH = строка нижнего правого угла окна (считая от 0), DL = столбец нижнего правого угла окна (считая от 0).

Прерывание 10h также обеспечивает функции вывода данных на уровне BIOS:

-- Прерывание BIOS 10h, функция 08 – считать символ и атрибут символа в текущей позиции курсора:

Ввод: АН ВН Вывод: АН AL

= 08, = номер страницы.

= атрибут символа,  
= ASCII-кодсимвола.

-- Прерывание BIOS 10h, функция 09 – вывести символ с заданным атри- бутом на экран:

Ввод:

АН = 09,  
ВН = номер страницы,  
AL = ASCII-код символа,  
BL = атрибут символа,  
СХ = число повторений символа.  
Выводит на экран любой символ, включая специальные символы (например, CR).

-- Прерывание BIOS 10h, функция 0Ah – вывести символ с текущим атрибутом на экран:

Ввод:

АН = 0Ah,  
ВН =номер страницы,  
AL =ASCII-код символа,  
СХ =число повторений символа.  
Выводит на экран любой символ, включая специальные символы (например, CR).

-- Прерывание BIOS 10h, функция 0Eh – вывести символ в режиме теле- тайпа:

Ввод:

АН = 0Eh,  
ВН =номер страницы,  
AL =ASCII-кодсимвола.  
Символы CR (0Dh), LF (0Ah), BEL (07h) интерпретируются как управляющие символы.  
Если текст при выводе выходит за пределы нижней строки, то экран

Ввод:

AH = 01,  
CH = номер верхней линии(20H –подавитькурсор),  
CL = номер нижней линии.  
Можно установить следующий размер курсора по вертикали:

-- от 0 до 13 – для монохромных и EGA мониторов;

-- от 0 до 7 для большинства цветных мониторов.  
Курсор сохраняет свой вид, пока программа не изменит его. Стандартные размеры курсора для монитора:

-- монохромный – 12/13; -- цветной – 6/7.

-- Прерывание BIOS 10h, функция 02 – установить положение курсора: Ввод: AH = 02,

BH = номер страницы,  
DH = номер строки (считая от 0, установка на строку 25 делает кур- сор невидимым), прокручивается вверх.

-- Прерывание BIOS 10h, функция 13h – вывести строку символов с задан- ными атрибутами:

Ввод:

АН = 13h  
AL = режим вывода (биты):

-- 0 – переместить курсор в конец строки после вывода,

-- 1 – строка содержит не только символы, но также и атрибуты, так что каждый символ описывается двумя байтами: ASCII-код и атрибут,

-- 2 – 7 – зарезервированы.  
СХ = длина строки (только число символов),  
BL = атрибут, если строка содержит только символы,  
DH = строка, начиная с которой будет выводиться строка символов, DL = столбец, начиная с которого будет выводиться строка симво- лов,  
ES:BP = адрес начала строки в памяти.  
Символы CR (0Dh), LF (0Ah), BEL (07h) интерпретируются как управляющие символы.

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 No2. По сравнению с функциями DOS, прерывание BIOS 16h предоставляет больше возможностей для считывания данных и управления клавиатурой и такой доступ практически эквивалентен по произво- дительности прямому доступу к буферу клавиатуры.

Каждой клавише на клавиатуре соответствует уникальный код, называе- мый скан-код. Этот код посылается клавиатурой при каждом нажатии и отпус- кании клавиши и обрабатывается BIOS – записывается в кольцевой буфер кла- виатуры.

Функции прерывания 16h:

-- Прерывание BIOS 16h, функция 00h (10h, 20h) – чтение символа с ожиданием:

Ввод:

АН = тип клавиатуры:  
-- 00h – 83/84-клавиши,  
-- 10h – 101/102-клавиши,

-- 20h – 122-клавиши.

Тип клавиатуры можно определить с помощью функции 09h прерывания 16h.

Если нажатой клавише соответствует ASCII-символ, то в АН возвра- щается код этого символа, а в AL – скан-код клавиши.  
Если нажатой клавише соответствует расширенный ASCII-код, то в AL возвращается префикс скан-кода (например, Е0 для серых клавиш) или 0, если префикса нет, а в АН – расширенный ASCII-код.

-- Прерывание BIOS 16h, функция 01h (11h, 21h) – проверка наличия символа в буфере:

АН = тип клавиатуры:  
-- 01h – 83/84-клавиши,  
-- 11h – 101/102-клавиши, -- 21h – 122-клавиши.

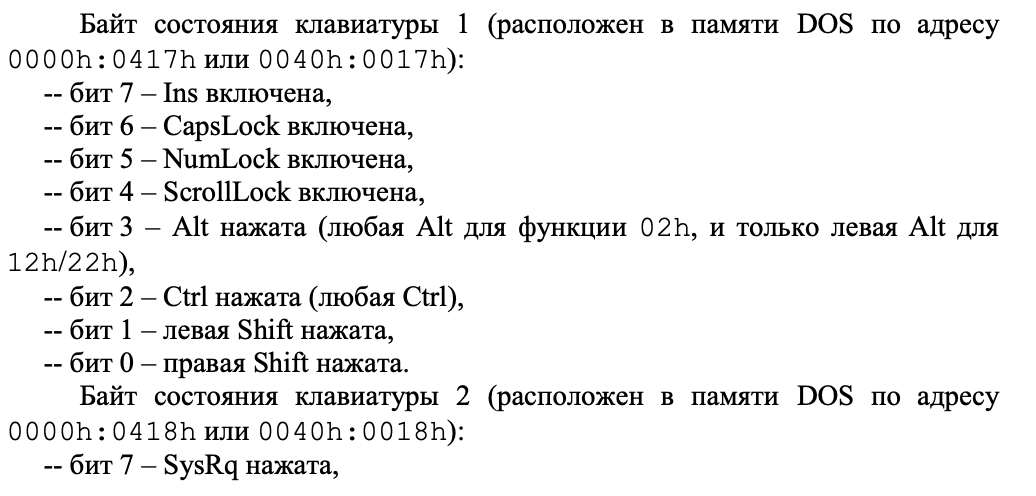
АН = тип клавиатуры:  
-- 02h – 83/84-клавиши,  
-- 12h – 101/102-клавиши, -- 22h – 122-клавиши.

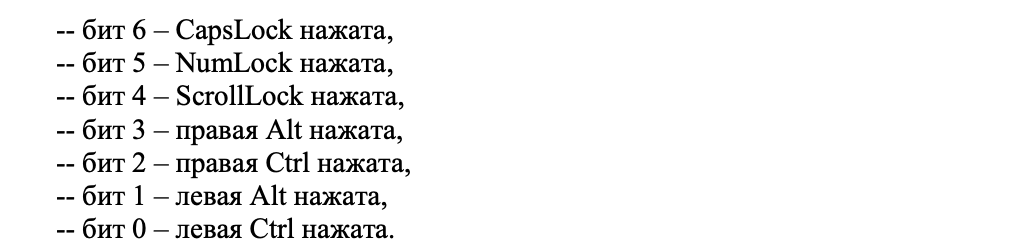
AL = байт состояния клавиатуры 1,  
АН = байт состояния клавиатуры 2 (только для функций 12h и 22h).

Если буфер пуст, то флаг ZF  
Если в буфере присутствует символ, то флаг ZF = 0, а дополнитель- ная информация содержится в регистре AX:  
AL = ASCII-код символа, 0 или префикс скан-кода;  
АН =скан-коднажатойклавишиилирасширенныйASCII-код. Проверяемый символ остается в буфере клавиатуры.

= 1;

-- Прерывание BIOS 16h, функция 02h (12h, 22h) – получить состояние клавиатуры:





Т.к. эти байты расположены в памяти по фиксированному адресу, то вме- сто вызова прерывания удобнее просто считывать и даже перезаписывать зна- чения этих байт напрямую, что изменит состояние клавиатуры.

; программа для выключения NumLock, CapsLock и ScrollLock

.model tiny

.code

org 100h

xor ax,ax

mov ds,ax

mov byte ptr ds:0417h,al ; байт состояния клавиатуры 1 = 0

ret ; выход из программы

end start

-- Прерывание BIOS 16h, функция 05 – поместить символ в буфер клавиа- туры:

start:

Ввод:

Вывод:

АН = 05h,  
СН = скан-код (можно поместить 0 вместо скан-кода, если функция, которая будет выполнять чтение из буфера, будет использовать толь- ко ASCII-код),  
CL =ASCII-код.

Если операция выполнена успешно, то AL = 00, Иначе, если буфер клавиатуры переполнен, то AL = 01h.

; СОМ-файл

; AX = 0 ; DS = 0

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 процессами:  
-- часы реального времени (RTC) – имеют автономное питание, используются для чтения/установки текущих даты и времени, установки будильника и для вызовапрерыванияIRQ8(INT 4Ah)каждуюмиллисекунду;

-- системный таймер – используется одновременно для управления контрол- лером прямого доступа к памяти, для управления динамиком и как генератор импульсов,вызывающийпрерываниеIRQ0(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: СХ = год (1980 – 2099), DH =месяц,

DL = день,  
AL = деньнедели(0–воскресенье,1–понедельникит.п.).

-- Функция DOS 2Bh (INT 21h) – установить дату:

Ввод:

АН = 2Bh,  
СХ = год (1980–2099),

DH = месяц,  
DL = день.

Вывод:  
-- Функция DOS 2Ch (INT 21h) – считать время:

Если введена несуществующая дата, то АН Если дата установлена, то АН = 00h.

= FFh,

Ввод: AH = 2Ch

Вывод: СН = час,  
CL = минута,

DH = секунда,  
DL = сотая доля секунды.

-- Функция DOS 2Dh (INT 21h) – установить время:

Ввод:

Вывод:

АН = 2Dh,  
СН =час,  
CL =минута,  
DH =секунда,  
DL = сотая доля секунды.

Если введено несуществующее время, то AL

= FFh,

Если время установлено, то AL = 00h.  
BIOS позволяет управлять часами PTC напрямую:

-- Прерывание BIOS 1Ah, функция 02h – считать время RTC:

Ввод:

АН = 02h

Вывод:

CF = 1 – если часы не работают или попытка чтения пришлась на момент обновления,

CF = 0 – если время успешно считано, то:   
СН = час (в формате BCD),  
CL = минута (в формате BCD),  
DH = секунда (в формате BCD  
DL = 01h – если действует летнее время,  
DL = 00h - если не действует летнее время.

-- Прерывание BIOS 1Ah, функция 03h – установить время RTC:

Ввод:

АН = 03h,  
СН = 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.

-- Прерывание BIOS 1Ah, функция 04h – считать дату RTC:

Ввод:

АН = 04h

Вывод: CF = 1 – если часы не работают или попытка чтения пришлась на момент обновления,  
CF = 0 – если дата успешно считана, то:  
СХ = 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: АН = 05h,

СХ = 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 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:

-- Прерывание BIOS 1Ah, функция 00h – прочитать значение счетчика вре- мени:

Ввод: АН = 00h.  
Вывод: CX:DX = значение счетчика,

AL = байт переполнения счетчика.

--Прерывание BIOS 1Ah, функция 01h– установить значение счетчика времени:

Ввод: АН = 01h,  
CX:DX =значениесчетчика.

Для изменения частоты работы таймера, BIOS имеет специальные функ- ции:

-- Прерывание BIOS 15h, функция 86h – формирование задержки таймера:

АН = 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:

АН = 83h,  
AL = 1–прерватьсчетчик,  
AL = 0–запуститьсчетчик:  
CX:DX =длительностьзадержкивмикросекундах,  
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 преры- ваниями.

Пример одного из вариантов организации задержки в видеоигре:

; начальная инициализация данных

...

main\_cycle:

mov cx,0

mov dx,20000

mov ah,86h

; пауза = 20000 микросекунд

; функция задержки

int 15h ; задержка

mov ah,1 ; функция проверки клавиатуры

int 16h

jz short no\_key\_pressed ; клавиша не нажата

; клавиша нажата – проверка нажата ли нужная клавиша:

xor ah,ah  
int 16h  
cmp ah,K1  
jne short not\_key\_K1  
; действия по нажатию клавиши K1

not\_key\_K1:

; проверка нажатия клавиши K2

...

no\_key\_pressed:

; действия, если не было нажатых клавиш

...

; проверка завершения программы

...

jne short exit

jmp short main\_cycle

; функция чтения кода клавиши  
; АН = код клавиши  
; проверка нажатия на клавишу K1

; продолжить основной цикл

exit:  
...

Код программы:

.model small

.stack 100h

.data

instruction db "Tetris",0Dh ,0Ah

db "Controls:",0Dh ,0Ah

db "Left/Right arrow - move",0Dh ,0Ah

db "Up arrow - rotate:",0Dh ,0Ah

db "Esc - exit:",0Dh ,0Ah

db "Enter - start:",0Dh ,0Ah, '$'

play\_field db 276 dup(00h)

figure\_shape db 00h, 00h, 00h, 00h ;L 10

db 00h, 00h, 10h, 00h

db 10h, 10h, 10h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h ;T 20

db 00h, 20h, 00h, 00h

db 20h, 20h, 20h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 30h, 00h ;Z 30

db 00h, 30h, 30h, 00h

db 00h, 30h, 00h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h ;S 40

db 00h, 00h, 40h, 40h

db 00h, 40h, 40h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h ;O 50

db 00h, 50h, 50h, 00h

db 00h, 50h, 50h, 00h

db 00h, 00h, 00h, 00h

db 00h, 0E0h, 0E0h, 00h ;J E0

db 00h, 0E0h, 00h, 00h

db 00h, 0E0h, 00h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 60h, 00h ;I 60

db 00h, 00h, 60h, 00h

db 00h, 00h, 60h, 00h

db 00h, 00h, 60h, 00h

cur\_shape db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h

db 00h, 00h, 00h, 00h

cur\_figure dw 0

cur\_figureX dw 0

cur\_figureY dw 0

prev\_time dw 0

score dw 0

.code

startPage proc near

push ax

push bx

push dx

push ds

mov ax, 0B800h ;segment of videodisplay

mov ds, ax

xor bx, bx

mov cx, 1000

loopScreenStart:

mov [bx], ' '

inc bx

mov [bx], 07h ;symbol beeps

inc bx

loop loopScreenStart

pop ds

mov ah, 9h

mov dx, offset instruction

int 21h

waitForEnter:

mov ah, 1 ;checks if there is a symbol in the buffer (a key is pressed)

int 16h

jz waitForEnter ;if not

mov ah, 0 ;read the symbol into AH

int 16h

cmp ah, 1Ch ;if Enter is pressed

je Enter

cmp ah, 01h ;if ESC is pressed

jne waitForEnter

mov ah, 00 ;sets videomode

mov al, 03 ;standard 16 colours (80\*25)

int 10h

mov ah, 4Ch

int 21h

Enter:

pop dx

pop bx

pop ax

ret

endp

initScreen proc near

push cx

push ax

push si

push ds

mov ax, 0b800h

mov ds, ax

xor bx, bx

mov cx, 1000

loopScreen:

mov [bx], ' '

inc bx

mov [bx], 07h ;symbol beeps

inc bx

loop loopScreen

xor si, si

mov ax, 40

field:

mov al, 80 ;starting point of the field

mul cl ;provides moving down

add ax, 4 ;offset from the left border of the screen

mov si, ax

;mov [si], ' '

inc si

mov [si], 70h ;left side of the field

add si, 21 ;shifting to the right side of the field

;mov [si], ' '

inc si

mov [si], 70h ;right side of the field

inc cx

cmp cx, 23 ;th length of the field

je field\_end

jmp field

field\_end:

mov cx, 2

field\_bottom:

mov al, 2

mul cl

add ax, 1760 ;starting point of the bottom of the field

mov si, ax

;mov [si], ' '

inc si

mov [si], 70h

inc cx

cmp cx, 14

je field\_bottom\_end

jmp field\_bottom

field\_bottom\_end:

mov [190], 'S'

mov [192], 'c'

mov [194], 'o'

mov [196], 'r'

mov [198], 'e'

mov [200], ':'

pop ds

pop si

pop ax

pop cx

ret

endp

initPlayfield proc near

push cx

push bx

push ax

xor ax, ax

mov cx, 276 ;sizes of the field

mov bx, offset play\_field

loopInit:

mov [bx], ah

inc bx

loop loopInit

mov cx, 0

borders:

mov al, 12

mul cl

mov bx, offset play\_field

add bx, ax

mov [bx], 60

add bx, 11

mov [bx], 60

inc cx

cmp cx, 23

je bordersEnd

jmp borders

bordersEnd:

mov cx, 0

bottom:

mov bx, offset play\_field

add bx, cx

add bx, 264

mov [bx], 10

inc cx

cmp cx, 12

je bottomEnd

jmp bottom

bottomEnd:

pop ax

pop bx

pop cx

ret

endp

newFigure proc near

push ax

push bx

push cx

push es

push si

push di

mov ah, 2Ch ;read time

int 21h

mov bh, 6

xor ax, ax

mov al, dl

div bh

mov bx, offset cur\_figure

mov [bx], ah

mov cur\_figureX, 5

mov cur\_figureY, 0

mov ax, ds

mov es, ax

mov bx, offset cur\_figure

mov cx, [bx]

mov al, 16

mul cl

mov bx, offset figure\_shape

mov si, bx

add si, ax

mov di, offset cur\_shape

mov cx, 16

loop11:

movsb

loop loop11

pop di

pop si

pop es

pop cx

pop bx

pop ax

ret

endp

printScore proc near

pusha ;pushes general-purpose registers

xor cx, cx

mov ax, score

xor dx, dx

mov si, 10

load\_stack:

div si ;ax/si

add dl, '0'

push dx

xor dx, dx

inc cx

cmp ax, 0

jne load\_stack

mov bx, 202

print\_stack:

pop dx

push ds

mov ax, 0b800h

mov ds, ax

mov [bx], dl

inc bx

mov [bx], 07h

inc bx

pop ds

loop print\_stack

popa

ret

endp

displayPlayField proc near

push ax

push es

push cx

push di

push si

mov ax, 0B800h

mov es, ax

mov cx, 20

mov di, 167

mov si, offset play\_field

add si, 25

loop1:

push cx

mov cx, 10

loop2:

movsb

inc di

loop loop2

add di, 60

add si, 2

pop cx

loop loop1

pop si

pop di

pop cx

pop es

pop ax

ret

endp

displayCurrentFigure proc near

push ax

push es

push cx

push di

push si

mov ax, 0B800h

mov es, ax

mov bx, offset cur\_figureY

mov cx, [bx]

mov al, 80

mul cl

mov di, ax

mov bx, offset cur\_figureX

mov ax, [bx]

add ax, ax

add ax, 5

add di, ax

mov si, offset cur\_shape

mov cx, 4

loop21:

push cx

mov cx, 4

loop22:

cmp di, 160

jl transparent

cmp [si], 00h

je transparent

movsb

jmp not\_transparent

transparent:

inc si

inc di

not\_transparent:

inc di

loop loop22

add di, 72

pop cx

loop loop21

pop si

pop di

pop cx

pop es

pop ax

ret

endp

checkCollision proc near

push di

push si

push bx

push cx

mov di, offset cur\_shape

mov si, offset play\_field

mov bx, offset cur\_figureY

mov ax, [bx]

mov bl, 12

mul bl

add si, ax

mov bx, offset cur\_figureX

mov ax, [bx]

cmp ax, 0FFh

jne check\_for\_collision

mov ax, 01h

pop cx

pop bx

pop si

pop di

ret

check\_for\_collision:

add si, ax

xor bx, bx

mov cx, 4

loop31:

push cx

mov cx, 4

loop32:

cmp [bx + di], 00h

je not\_collis

cmp [bx + si], 00h

je not\_collis

mov ax, 01h

pop cx

pop cx

pop bx

pop si

pop di

ret

not\_collis:

inc di

inc si

loop loop32

pop cx

add si, 8

loop loop31

xor ax, ax

pop cx

pop bx

pop si

pop di

ret

endp

rotateFigure proc near

push ax

push bx

push cx

mov bx, offset cur\_shape

mov ah, [bx]

mov ch, ah

mov ah, [bx + 3]

mov [bx], ah

mov ah, [bx + 15]

mov [bx + 3], ah

mov ah, [bx + 12]

mov [bx + 15], ah

mov [bx + 12], ch

mov ah, [bx + 1]

mov ch, ah

mov ah, [bx + 7]

mov [bx + 1], ah

mov ah, [bx + 14]

mov [bx + 7], ah

mov ah, [bx + 8]

mov [bx + 14], ah

mov [bx + 8], ch

mov ah, [bx + 2]

mov ch, ah

mov ah, [bx + 11]

mov [bx + 2], ah

mov ah, [bx + 13]

mov [bx + 11], ah

mov ah, [bx + 4]

mov [bx + 13], ah

mov [bx + 4], ch

mov ah, [bx + 5]

mov ch, ah

mov ah, [bx + 6]

mov [bx + 5], ah

mov ah, [bx + 10]

mov [bx + 6], ah

mov ah, [bx + 9]

mov [bx + 10], ah

mov [bx + 9], ch

pop cx

pop bx

pop ax

ret

endp

checkLines proc near

push ax

push bx

push cx

push es

push si

push di

mov bx, offset play\_field

add bx, 25

mov cx, 20

loop51:

push cx

xor dx, dx

mov cx, 10

loop52:

cmp [bx], 00h

je if\_transp

inc dx

if\_transp:

inc bx

loop loop52

add bx, 2

cmp dx, 10

jl not\_full

add score, 10

call printScore

pop cx

push cx

push bx

mov ax, 22

sub ax, cx

push ax

mov cl, 12

mul cl

mov bx, offset play\_field

add bx, ax

inc bx

pop ax

loop53:

mov cx, 10

loop54:

push ax

mov ah, [bx-12]

mov [bx], ah

pop ax

inc bx

loop loop54

sub bx, 22

dec ax

cmp ax, 1

jg loop53

pop bx

not\_full:

pop cx

loop loop51

pop di

pop si

pop es

pop cx

pop bx

pop ax

ret

endp

printLose proc near

push ds

mov ax, 0b800h

mov ds, ax

mov bx, 808

mov [bx], 'Y'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'o'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'u'

inc bx

mov [bx], 07h

inc bx

mov [bx], ' '

inc bx

mov [bx], 07h

inc bx

mov [bx], 'l'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'o'

inc bx

mov [bx], 07h

inc bx

mov [bx], 's'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'e'

inc bx

mov [bx], 07h

inc bx

mov bx, 884

mov [bx], 'P'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'r'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'e'

inc bx

mov [bx], 07h

inc bx

mov [bx], 's'

inc bx

mov [bx], 07h

inc bx

mov [bx], 's'

inc bx

mov [bx], 07h

inc bx

mov [bx], ' '

inc bx

mov [bx], 07h

inc bx

mov [bx], 'E'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'n'

inc bx

mov [bx], 07h

inc bx

mov [bx], 't'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'e'

inc bx

mov [bx], 07h

inc bx

mov [bx], 'r'

inc bx

mov [bx], 07h

inc bx

mov [bx], ' '

inc bx

mov [bx], 07h

inc bx

pop ds

ret

endp

main:

mov ax, @data

mov ds, ax

mov ah, 00 ;sets videomode

mov al, 01 ;standard 16 colours

int 10h

call startPage

restart:

mov score, 0

mov prev\_time, 0

call initScreen

call initPlayField

call newFigure

call displayPlayField

call displayCurrentFigure

call printScore

mov ah, 01h

xor cx, cx

xor dx, dx

int 1ah

start:

mov ah, 1

int 16h

jz noKeyPressed

xor ah, ah

int 16h

cmp ah, 4Dh ;if right arrow is pressed

jne not\_right

mov bx, offset cur\_figureX

inc [bx]

push ax

call checkCollision

cmp ax, 00h

je no\_col

mov bx, offset cur\_figureX

dec [bx]

no\_col:

pop ax

call displayPlayField

call displayCurrentFigure

not\_right:

cmp ah, 4Bh ;if left arrow is pressed

jne not\_left

mov bx, offset cur\_figureX

dec [bx]

push ax

call checkCollision

cmp ax, 00h

je not\_colR

mov bx, offset cur\_figureX

inc [bx]

not\_colR:

pop ax

call displayPlayField

call displayCurrentFigure

not\_left:

cmp ah, 48h ;if up arrow is pressed

jne not\_up

call rotateFigure

push ax

call checkCollision

cmp ax, 00h

je not\_colU

call rotateFigure

call rotateFigure

call rotateFigure

not\_colU:

pop ax

call displayPlayField

call displayCurrentFigure

not\_up:

cmp ah, 01h ;if ESC is pressed

jne notEscape

jmp exit

notEscape:

noKeyPressed:

mov ah, 00h ;read time

int 1ah

push dx

mov ax, prev\_time

sub dx, ax

mov ax, dx

pop dx

cmp ax, 9

jl no\_drop

mov prev\_time, dx

mov bx, offset cur\_figureY

inc [bx]

push ax

call checkCollision

cmp ax, 00h

je not\_colDr

mov bx, offset cur\_figureY

dec [bx]

call checkLines

call newFigure

call checkCollision

cmp ax, 00h

jne wasted

not\_colDr:

pop ax

call displayPlayField

call displayCurrentFigure

no\_drop:

no\_update:

jmp start

wasted:

call printLose

wait\_for\_ent:

mov ah, 1

int 16h

jz wait\_for\_ent

xor ah, ah

int 16h

cmp ah, 1Ch

jne not\_ent

jmp restart

not\_ent:

cmp ah, 01

jne wait\_for\_ent

exit:

mov ah, 00 ;set videomode

mov al, 03

int 10h

mov ah, 4Ch

int 21h

end main

Примеры работы программы:





Вывод:

В ходе лабораторной работы было написано приложение, реализующее игру «Тетрис» посредством работы с видеопамятью и прерываниями DOS, BIOS. Я ознакомилась в рамках создания видеоигры с обработкой нажатий кнопок клавиатуры, рассмотрела прямой доступ к видеопамяти с целью формирования игрового поля и информации для пользователя.