

# **Как писать игры для ZX Spectrum**

**Версия 0.6**

**Джонатан Коулдвелл**

## **История документа**

Версия 0.1 - Февраль 2006

Версия 0.2 - Январь 2007

Версия 0.3 - Декабрь 2008

Версия 0.4 - Июль 2009

Версия 0.5 - Октябрь 2010

Версия 0.6 - Апрель 2014

**Перевод: helcril специально для zx.pk.ru**

*(Работоспособность всех примеров программ проверена переводчиком  
с помощью встроенного ассемблера эмулятора ZX Spin v0.7)*

## Авторские права

Все права защищены. Любое цитирование этого документа, или его части запрещено без письменного разрешения автора.

## Введение

Итак, вы прочитали документацию к ZX80, знаете как инструкции влияют на регистры и хотите применить эти знания на практике. Судя по количеству полученных мной писем, с вопросами о том, как опрашивать клавиатуру, как вычислять адреса экрана или выдать белый шум на динамик, стало совершенно ясно, что ресурсов для начинающего программиста на Спектруме явно не хватает. Я надеюсь, что этот документ будет постепенно заполнять пробелы, по мере своего развития. В нынешнем своем состоянии он конечно очень далек от завершения. Но публикуя те основные главы, что уже готовы, я надеюсь помочь другим программистам.

ZX Spectrum был выпущен в апреле 1982 г., и по нынешним меркам это примитивная машина. В Великобритании и некоторых других странах он был самой популярной игровой платформой 80-х. Но и сегодня, при помощи эмуляторов многие ностальгирующие люди любят поиграть в игры своего детства. Другие же открыли для себя эту платформу совсем недавно, а некоторые даже отваживаются начать писать игры для этого простенького компьютера. В конце концов, если вы способны писать достойный машинный код для компьютера 80-х, то вам, скорее всего, удастся написать практически что угодно.

Перфекционистам это точно не понравится, но я скажу, что написание игры не имеет ничего общего с «идеальным кодом для ZX80», если таковой вообще существует. Написание игры для Спектрума - довольно трудоемкое занятие, и вы никогда не закончите ее, если будете все время озабочены оптимизацией процедур подсчета очков и алгоритмов опроса клавиатуры. Написав процедуру, которая выполняет свою задачу и не приводит к ошибкам где-либо – двигайтесь дальше. Некоторая неэффективность и корявость гораздо менее важны, чем хороший геймплей. Никто в здравом уме не будет дизассемблировать ваш код и выискивать в нем ошибки.

Главы этого документа организованы таким образом, чтобы позволить читателю начать писать простые игры как можно быстрее. Ничто не сравнится с трепетом при написании своей первой игры в машинных кодах, поэтому в этом руководстве основной необходимый минимум дается в первых главах. Дальше мы рассмотрим более продвинутые методы, которые позволят читателю улучшить качество игр, которые он способен написать.

В этом документе я сделал несколько допущений. Для начала, предполагается, что вы уже знакомы с большинством инструкций ZX80 и знаете, что они делают. Если нет, то есть достаточное количество источников, которые расскажут об этом лучше, чем я. Выучить машинные инструкции несложно, а вот осмысленно выстраивать их в программу – другое дело. Знакомство с инструкциями загрузки (**ld**), сравнения (**cp**) и условного перехода (**jp z / jp c / jp nc**) – неплохое начало. Остальное встанет на свои места позже.

## Инструменты

Сегодня мы можем использовать более совершенное железо, и нет нужды разрабатывать программы на платформе, для которой они сделаны. Есть множество неплохих кросс-ассемблеров, которые позволяют писать программы для Спектрума на вашем ПК. Бинарные файлы затем могут быть импортированы в эмулятор. Например, Spin – один из эмуляторов, поддерживающих данную функцию.

Для графики я использую (и вам рекомендую) утилиту SevenUp. Она может конвертировать изображения в формат Спектрума, и позволяет установить порядок, в котором необходимо отсортировать спрайты и другую графику. Вывод может быть в форме бинарного изображения, или исходного кода. Другая популярная программа – TommyGun.

Для музыки я рекомендую утилиту SoundTracker, которую можно скачать с сайта World of Spectrum. Так же вам понадобится отдельная программа-компилятор. Имейте ввиду, что эти программы – для Спектрума, поэтому вам понадобится эмулятор для их запуска.

Я не могу дать рекомендаций по современным редакторам и кросс-компиляторам, т.к. сам использую древний редактор и кросс-ассемблер Z80 Macro для DOS, написанный в 1985 году. Если вам нужен совет о том, какие инструменты лучше использовать – предлагаю вам посетить форум <http://www.worldofspectrum.org/forums/>. В этом дружелюбном сообществе есть множество людей с богатым опытом, всегда готовых помочь.

## Мои личные предпочтения

За многие годы написания программ для Спектрума у меня сформировалось несколько привычек, которые могут показаться странными. Например, я определяю координатную плоскость не так, как это принято в математике. Мои программы в машинных кодах следуют логике команды **PRINT AT X,Y** Sinclair BASIC'а. Здесь **X** соответствует порядковому номеру знакоместа или пикселя от вершины экрана, а **Y** – номеру знакоместа в строке, или пикселя от левого края экрана. Если это поначалу покажется вам странным, то извините, но такой способ всегда казался мне более логичным и я к нему очень привык. Также некоторые мои методы порой могут показаться необычными, поэтому, если вам известен лучший по всем параметрам способ – используйте его.

Еще кое-что. Комментировать свой код не только важно, но просто необходимо. Найти ошибку в некомментированном коде, написанном всего лишь пару недель назад, будет чертовски сложно. Описание всех подпрограмм, которые вы пишете может показаться утомительным, но это сократит время разработки в долгосрочной перспективе. К тому же, если вы захотите использовать процедуру в другой игре когда-нибудь в будущем, вам будет очень легко взять нужный кусок кода и адаптировать его под новый проект.

Кроме этого всего этого, просто развлекайтесь. Если у вас есть какие-либо предложения, или сообщения об ошибках, пожалуйста, пишите.

Джонатан Коулдвелл, Январь 2007.  
<http://www.spanglefish.com/egghead/>



## **Содержание**

### **Глава 1 – Простая графика и текст**

**Hello World**

**Вывод простой графики**

**Вывод чисел**

**Меняем цвета**

### **Глава 2 – Управление с клавиатуры и джойстика**

**Одиночные нажатия кнопки**

**Несколько кнопок одновременно**

**Джойстики**

**Простая игра**

### **Глава 3 – Звуковые эффекты динамика**

**Динамик**

**ВЕЕР**

**Белый шум**

### **Глава 4 – Случайные числа**

### **Глава 5 – Простейшее определение столкновений**

**Определяем атрибуты**

**Вычисляем адреса атрибутов**

**Применяем полученные знания к нашей игре**

### **Глава 6 - Таблицы**

**Враги не ходят поодиночке**

**Используем индексные регистровые пары**

### **Глава 7 – Определение столкновений с пришельцами**

**Проверка координат**

**Столкновение спрайтов**

### **Глава 8 - Спрайты**

**Перевод позиций пикселей в адреса экранной области**

**Используем таблицу адресов экрана**

**Вычисляем адреса экрана**

**Сдвиг**

**Предварительный сдвиг спрайтов**

**Метод сканирования байтов**

### **Глава 9 – Background-графика**

**Вывод блоков**

## **Глава 10 - Счет и таблицы рекордов**

Процедуры подсчета очков

Таблицы рекордов

## **Глава 11 – Движения врагов**

Патрулирующие враги

Умные пришельцы

## **Глава 12 – Управление временем**

Инструкция Halt

Процедуры Спектрума Clock и Vsync

Генерируем случайные числа

## **Глава 13 – Двойная буферизация**

Создаем буфер экрана

Скроллинг буфера

## **Глава 14 – Более сложные движения**

Таблицы прыжка и инерции

Более точный расчет координат

Повороты

## **Глава 15 - Математика**

## **Глава 16 - Музыка и АУ-эффекты**

АУ-3-8912

Используем музыкальные драйверы

## **Глава 17 - Прерывания**

## **Глава 18 – Делаем загрузчик и автозапуск игры**

## Глава 1 – Простая графика и текст

### Hello World

Обычно первая программа на BASIC, которую пишут начинающие программисты, выглядит примерно так:

```
10 PRINT "hello world"
20 GOTO 10
```

Хорошо, текст в кавычках, конечно, может и отличаться. В первой программ вы могли написать «Вася - codemaster», или «Здесь был Дима». Однако признаем, что отображение текста и графики на экране – вероятно, самый важный аспект в написании компьютерной игры. К тому же, практически невозможно вообразить электронную игру без дисплея, за исключением, наверное, пинбола, или «однорукого бандита». Принимая это во внимание, давайте начнем наше обучение с нескольких важных процедур из ПЗУ Спектрума.

Итак, как же перевести приведенную выше программу из BASIC в машинные коды? Мы конечно можем выводить на экран с помощью инструкции RST 16 (аналог PRINT CHR\$). Однако она печатает только один символ, содержащийся в аккумуляторе. Чтобы вывести строку на экран, нам нужно вызвать две процедуры. Одну – чтобы установить верхнюю область экрана для печати (канал 2), вторую – чтобы напечатать саму строку. Процедура ПЗУ по адресу 5633 установит канал, номер которого мы передали в аккумулятор. А процедура 8252 напечатает строку, начинающуюся по адресу, содержащемуся в **de** и длиной, указанной в **bc**. Как только канал 2 выбран – весь вывод будет происходить в верхнюю часть экрана, пока мы снова не вызовем 5633 с другим аргументом, чтобы перенаправить вывод куда-либо еще. Вот еще парочка интересных каналов: 1 для нижней части экрана (как PRINT #1 в BASIC, служит для печати в двух нижних строках экрана) и 3 для ZX-принтера.

```
loop    ld a,2           ; верхняя часть экрана
        call 5633        ; устанавливаем канал
        ld de,string     ; адрес строки
        ld bc,eostr-string ; длина выводимой строки
        call 8252        ; печатаем нашу строку
        jp loop          ; повторяем, пока не заполним экран

string defb '(Здесь ваше имя) is cool'
eostr equ $
```

При запуске этой программы экран будет заполняться текстом до запроса scroll?. Однако, как вы уже, наверное, заметили, в отличие от программы на BASIC, где фраза печатается в отдельной строке, у нас текст выводится непрерывно. Это не совсем то, чего мы хотели. Чтобы добиться нужного результата, нам нужно разделить строки с помощью управляющего кода ASCII. Можно сделать это так: загрузить в аккумулятор код перевода строки (13), а затем вызвать RST 16 для печати этого кода. Другой, более эффективный способ - просто добавить код ASCII в конец нашей строки:

```
string defb '(Здесь ваше имя) is cool'
        defb 13
eostr equ $
```

Есть множество управляющих кодов ASCII, подобных этому, которые позволяют изменить позицию вывода, цвета и прочее. Поэкспериментируйте, и выберите для себя наиболее полезные. Вот те основные, что использую я:

13	NEWLINE	устанавливает вывод на начало новой строки.
16,c	INK	устанавливает цвет чернил, байт c – код цвета.
17,c	PAPER	устанавливает цвет бумаги, байт c – код цвета.
22,x,y	AT	устанавливает позицию вывода по координатам, указанным в байтах x,y.

Код 22 особенно полезен, он выводит строку, или символ по нужным координатам. В этом примере мы выводим восклицательный знак в нижнюю строчку верхней части экрана:

```

ld a,2          ; верхняя часть экрана
call 5633       ; устанавливаем канал
ld de,string    ; адрес строки
ld bc,eostr-string ; длина строки
call 8252       ; печатаем нашу строку
ret

string defb 22,21,31,'!'
eostr equ $

```

Эта программа идет еще дальше, и выводит анимацию передвижения символа звездочки по экрану снизу вверх:

```

loop  ld a,2          ; 2 = верхняя часть экрана.
      call 5633       ; устанавливаем канал.
      ld a,21         ; строка 21 = самая нижняя строка.
      ld (xcoord),a   ; устанавливаем начальную координату x.
      call setxy      ; устанавливаем наши x/y координаты.
      ld a,'*'        ; выбираем символ.
      rst 16          ; выводим его.
      call delay      ; задержка.
      call setxy      ; снова те же координаты.
      ld a,32         ; ASCII-код для пробела.
      rst 16          ; удаляем символ на старой позиции (печатаем пробел).
      ld hl,xcoord    ; позиция по вертикали.
      dec (hl)        ; вверх на одну строку.
      ld a,(hl)       ; где она сейчас?
      cp 255          ; вышла за пределы экрана?
      jr nz,loop      ; если нет, продолжаем.
      ret
delay ld b,10         ; время задержки.
delay0 halt           ; ждем прерывания.
      djnz delay0     ; переходим на метку loop.
      ret             ; возврат из процедуры.
setxy ld a,22         ; ASCII-код для печати по координатам.
      rst 16          ; выводим его.
      ld a,(xcoord)   ; позиция по вертикали.
      rst 16          ; выводим ее.
      ld a,(ycoord)   ; позиция по горизонтали.
      rst 16          ; выводим ее.
      ret

xcoord defb 0
ycoord defb 15

```



## Вывод простой графики

Передвигать звездочки по экрану конечно очень весело, но даже для самой простой игры нам необходимо уметь выводить графику. Более продвинутые способы мы будем изучать в последующих главах, сейчас же остановимся на простой графике в стиле Space Invaders. Любой программист на BASIC вам скажет, что для этого у Спектрума есть очень простой механизм – графика определяемая пользователем, или UGD (User Defined Graphic).

ASCII-таблица Спектрума содержит 21 символ графики, определяемой пользователем (19 в режиме 128K). Начинаются они с кода 144, и заканчиваются 164 (162 в режиме 128K). В BASIC мы задаем UDG при помощи команды POKE, записывая нужные данные в верхнюю область памяти ОЗУ. Но в машинных кодах будет более логичным изменить системную переменную, которая указывает на адрес, по которому хранится UGD. Это достигается путем изменения двухбайтового значения по адресу 23675.

Теперь мы можем изменить нашу программу передвижения, и вместо звездочки у нас будет графический символ. Строки, которые нужно изменить выделены подчеркиванием.

	<u>ld h1,udgs</u>	<u>; наш графический символ.</u>
	<u>ld (23675),h1</u>	<u>; устанавливаем системную переменную UDG.</u>
	ld a,2	; 2 = верхняя часть экрана.
	call 5633	; устанавливаем канал.
	ld a,21	; строка 21 = самая нижняя строка.
	ld (xcoord),a	; устанавливаем начальную координату x.
loop	call setxy	; устанавливаем наши x/y координаты.
	<u>ld a,144</u>	<u>; выбираем наш UDG, вместо звездочки.</u>
	rst 16	; выводим его.
	call delay	; задержка.
	call setxy	; снова те же координаты.
	ld a,32	; ASCII-код для пробела.
	rst 16	; удаляем символ на старой позиции (печатаем пробел).
	call setxy	; снова те же координаты. (прим. перев. – Не знаю
		; <u>зачем здесь эта строчка, код работает и без нее</u> )
	ld h1,xcoord	; позиция по вертикали.
	dec (h1)	; вверх на одну строку.
	ld a,(xcoord)	; где она сейчас?
	cp 255	; вышла за пределы экрана?
	jr nz,loop	; если нет, продолжаем.
	ret	
delay	ld b,10	; время задержки.
delay0	halt	; ждем прерывания.
	djnz delay0	; переходим на метку loop.
	ret	; возврат из процедуры.
setxy	ld a,22	; ASCII-код для печати по координатам.
	rst 16	; выводим его.
	ld a,(xcoord)	; позиция по вертикали.
	rst 16	; выводим ее.
	ld a,(ycoord)	; позиция по горизонтали.
	rst 16	; выводим ее.
	ret	
xcoord	defb 0	
ycoord	defb 15	
udgs	<u>defb 60,126,219,153</u>	
	<u>defb 255,255,219,219</u>	

Как говорил Рольф Харрис "Можете сказать, что из этого получится?" (прим. перев. – Рольф Харрис – австралийский музыкант, комик, художник, актер и телеведущий)

Конечно же, вы можете использовать более 21 символа пользовательской графики, если вам захочется. Просто выделите для них место в памяти и указывайте на них, когда это нужно.

Альтернативный вариант – вы можете переопределить таблицу символов. Это дает больший набор ASCII символов – от 32 (пробел) до 127 (знак копирайта). Вы даже можете перемешать графику и текст, изменив буквы и цифры шрифта по своему вкусу. Например, используя символы и строчные буквы шрифта в качестве зомби, пришельцев и т.д. в вашей игре. Чтобы указать на другой набор символов, мы вычитаем 256 из адреса, по которому расположен шрифт и помещаем это значение в двухбайтную системную переменную, которая находится по адресу 23606. Например, стандартный шрифт Sinclair, находится в ПЗУ по адресу 15616, поэтому системная переменная по адресу 23606 указывает на адрес 15360 при первом включении Спектрума.

Этот код копирует Sinclair-шрифт из ПЗУ в ОЗУ, делает его жирным, а затем меняет значение системной переменной, чтобы она указывала на него:

```

font1  ld hl,15616      ; загружаем адрес ROM-шрифта.
        ld de,60000     ; адрес нашего нового шрифта.
        ld bc,768       ; 96 символов * 8 строк для изменения.
        ld a,(hl)       ; получаем изображение символа.
        rlca            ; сдвиг влево.
        or (hl)         ; совмещаем два изображения.
        ld (de),a       ; записываем в новый шрифт.
        inc hl          ; следующий байт старого шрифта.
        inc de          ; следующий байт нового шрифта.
        dec bc          ; уменьшаем счетчик.
        ld a,b         ; проверяем регистровую
        or c            ; пару bc на ноль.
        jr nz,font1     ; повторяем пока bc не будет=0.
        ld hl,60000-256 ; адрес шрифта минус 32*8.
        ld (23606),hl   ; указываем на новый шрифт.
        ret

```

## Вывод чисел

Для большинства игр счет игрока лучше определять как строку ASCII-символов. Это, конечно, потребует от нас написания процедур подсчета очков, а составление таблиц рекордов станет настоящей занозой в заднице для неопытного программиста на Ассемблере. Но мы рассмотрим эти вопросы позже, а сейчас используем для печати чисел очень удобные процедуры из ПЗУ.

Есть два способа вывести число на экране. Первый из них – использовать ту же процедуру, которую ПЗУ использует для нумерации строк в Sinclair BASIC. Для этого мы просто должны загрузить в регистровую пару **bc** нужное нам число, а затем вызвать 6683:

```

ld bc,(score)
call 6683

```

Однако, максимальный номер строки в BASIC всего лишь 9999, что дает нам возможность выводить лишь четырехзначные числа. Как только игрок достигнет 10000, вместо цифр будут отображаться другие символы ASCII. К счастью есть другой способ. Вместо вызова процедуры нумерации строк, мы можем вызвать процедуру, которая поместит содержимое регистровой пары **bc** в стек калькулятора. Затем, вызовем другую процедуру, которая отобразит число, находящееся на вершине стека калькулятора. Не забивайте себе голову вопросами о том, что такое стек калькулятора и зачем он нужен, т.к. он практически бесполезен для разработчика аркадных игр. Тем не менее, мы будем использовать его там, где

сможем. Просто помните, что следующие строки отображают число от 0 до 65535 включительно:

```
ld bc,(score)
call 11563      ; помещаем число из bc в стек калькулятора.
call 11747      ; выводим вершину стека калькулятора.
```

## Меняем цвета

Чтобы установить цвет чернил, бумаги, эффекты яркости и мерцания мы можем записать напрямую в переменную 23693, а затем очистить экран вызвав процедуру ПЗУ:

; Нам нужен желтый экран.

```
ld a,49          ; синие чернила (1) на желтой бумаге (6*8).
ld (23693),a      ; устанавливаем наши цвета.
call 3503         ; очищаем экран.
```

Самый быстрый и простой способ изменить цвет бордюра – это записать код цвета в порт 254. Три младшие бита посланного нами байта определяют цвет, в данном случае красный:

```
ld a,2           ; 2 код для красного.
out (254),a       ; записываем в порт 254.
```

Однако порт 254 так же управляет динамиком и разъемом микрофона с помощью битов 3 и 4. Соответственно, наш эффект бордюра будет длиться только пока мы не вызовем процедуру работы с динамиком (об этом позже). Поэтому нам необходимо более надежное решение. Для этого нужно просто загрузить в аккумулятор код цвета и вызвать процедуру ПЗУ 8859. Так мы изменим цвет и соответствующую системную переменную BORDCR (которая находится по адресу 23624). Чтобы установить постоянный красный бордюр можно написать следующее:

```
ld a,2           ; 2 код для красного.
call 8859        ; устанавливаем цвет бордюра.
```

## Глава 2 – Клавиатура и джойстик

### Одиночные нажатия

Если вы не отключали или, либо как-либо не изменяли стандартный режим прерываний Спектрума, то ПЗУ автоматически опрашивает клавиатуру и обновляет несколько системных переменных, расположенных по адресу 23552, со скоростью пятьдесят раз в секунду. Простейший способ проверить кнопку на нажатие – сначала загрузить ноль по адресу 23560 и проверять, меняется ли значение по этому адресу. Результатом будет ASCII-код нажатой клавиши. Это больше всего подходит для ситуаций вроде «нажмите любую клавишу для продолжения», выбора пунктов меню, и ввода имени с клавиатуры в таблицу рекордов. Подобная процедура может выглядеть так:

```
loop  ld hl,23560      ; системная переменная LAST К.
      ld (hl),0        ; помещаем туда ноль.
      ld a,(hl)        ; новое значение LAST К.
      cp 0             ; все еще ноль?
      jr z,loop        ; если да, кнопка не была нажата.
      ret              ; кнопка была нажата, возврат.
```

### Нескольких кнопок одновременно

Однако, проверки на одиночные нажатия очень редко нужны в динамичных аркадных играх. В них мы чаще отслеживаем одновременные нажатия нескольких кнопок. И вот здесь все становится немного сложнее. Вместо считывания адресов памяти, нам придется считывать данные с одного или нескольких из восьми портов, каждый из которых отвечает за группу из пяти клавиш. Конечно, большинство моделей Спектрума имеют гораздо больше сорока кнопок. Так куда же подключены остальные? Ну, на самом деле никуда. Оригинальная раскладка клавиатуры Спектрума состоит из сорока клавиш, разделенных на восемь групп, в каждой из которых пять кнопок. Чтобы использовать некоторые функции, необходимо нажимать определенные комбинации. Например, для удаления – CAPS SHIFT+0. Синклер добавил дополнительные клавиши в 1985 году для модели Spectrum+, но они просто имитируют необходимые комбинации оригинальной клавиатуры Спектрума.

Оригинальная раскладка была разделена на следующие группы:

#### Порт    Клавиши

```
32766 В, N, M, Symbol Shift, Space
49150 H, J, K, L, Enter
57342 Y, U, I, O, P
61438 6, 7, 8, 9, 0
63486 5, 4, 3, 2, 1
64510 T, R, E, W, Q
65022 G, F, D, S, A
65278 V, C, X, Z, Caps shift
```

Чтобы определить, какие клавиши были нажаты, мы считываем данные из нужного порта. Младшие пять бит (d0-d4, значения 1,2,4,8 и 16) будут соответствовать пяти клавишам. При этом бит d0 отвечает за самую крайнюю в ряду, а d4 – за ту, что в середине ряда. Учтите, что

если клавиша нажата, то бит сбрасывается в ноль, если не нажата – устанавливается в единицу. Наверное, это не совсем интуитивно, но так уж есть.

Чтобы считать значения группы из пяти клавиш нужно просто загрузить номер порта в регистровую пару **bc**, а затем выполнить инструкцию **in a,(c)**. Так как нам необходимы только младшие пять бит, мы можем отбросить ненужные биты инструкцией **and 31**, либо циклическим сдвигом аккумулятора вправо через флаг переноса, пять раз используя конструкцию вида **rra:call c, (address)**.

Если вам не очень понятно, то посмотрите на следующий пример:

```
ld bc,63486      ; группа клавиш 1-5/порт джойстика 2.
in a,(c)          ; смотрим какие клавиши нажаты.
rra              ; крайний бит = клавиша 1.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, двигаемся влево.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 2) = клавиша 2.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, двигаемся вправо.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 4) = клавиша 3.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, двигаемся вниз.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 8) считываем клавишу 4.
call nc,mp1      ; была нажата, двигаемся вверх.
```

## Джойстики

Порты Sinclair-джойстика 1 и 2 были просто назначены на две группы цифровых клавиш. Вы легко можете проверить это, набирая цифры при помощи джойстика в режиме BASIC. Порт 1 (Интерфейс 2) назначен на клавиши 6, 7, 8, 9 и 0, а Порт 2 (Интерфейс 1) на клавиши 1, 2, 3, 4 и 5. Чтобы опросить джойстик, мы, так же как и с клавиатурой, просто считываем данные из порта. Sinclair-джойстики используют порты 63486 (Интерфейс 1/ Порт2) и 61438 (Интерфейс 2/ Порт 1). Биты d0-d4 выдают 0 при нажатии, 1 при отсутствии сигнала.

Популярный Kempston-джойстик не привязан к клавиатуре и может быть считан из порта 31. Это означает, что мы можем просто использовать инструкцию **in a, (31)**. Здесь так же используются биты d0-d4, однако они устанавливаются по-другому: высокий уровень при срабатывании, низкий – при отсутствии сигнала. В результате бит будет единицей при нажатии, и нулем в обычном состоянии.

; Пример процедуры управления при помощи джойстика.

```
joycon ld bc,31      ; порт kempston-джойстика.
in a,(c)           ; считываем значение.
and 2              ; проверяем бит направления влево.
call nz,joyl       ; двигаемся влево.
in a,(c)           ; считываем вход.
and 1              ; проверяем бит направления вправо.
call nz,joyr       ; двигаемся вправо.
in a,(c)           ; считываем вход.
and 8              ; проверяем бит направления вверх.
call nz,joyu       ; двигаемся вверх.
in a,(c)           ; считываем вход.
and 4              ; проверяем бит направления вниз.
call nz,joyd       ; двигаемся вниз.
in a,(c)           ; считываем вход.
and 16             ; проверяем бит клавиши «огонь».
call nz,fire       ; «огонь» нажат.
```

## Простая игра

Теперь мы можем пойти дальше и применить на практике то, что уже изучили. Напишем основной участок кода, отвечающий за игровое управление. Здесь мы сформируем основу для клона игры «Centipede» (*прим. перев. – аркадная игра в жанре shoot 'em up, Atari Inc. 1981*), которую мы будем разрабатывать в течение нескольких следующих глав. Мы еще не изучили всего, что нужно для такой игры, но уже можем начать с небольшого цикла управления, который позволит игроку передвигать небольшую пушку по экрану. Учтите, что эта программа не выходит в BASIC, поэтому перед ее запуском убедитесь, что сохранили копию исходного кода.

; Устанавливаем черный экран.

```
ld a,71          ; белые чернила (7) на черной бумаге (0),
                  ; яркость (64).
ld (23693),a      ; устанавливаем наши цвета.
xor a             ; быстрый способ загрузить 0 в аккумулятор.
call 8859         ; устанавливаем постоянный цвет бордюра.
```

; Настраиваем нашу графику.

```
ld hl,blocks      ; адрес данных UDГ.
ld (23675),hl     ; переменная теперь указывает на него.
```

; ОК, начнем игру.

```
call 3503         ; Процедура ПЗУ – очищает экран, открывает канал 2.
```

; Инициализация координат.

```
ld hl,21+15*256   ; загружаем в hl начальные координаты.
ld (plx),hl       ; устанавливаем координаты игрока.

call basexy       ; устанавливаем координаты x,y игрока.
call splyar       ; выводим изображение игрока.
```

; Здесь основной цикл.

mloop equ \$

; Стираем изображение игрока.

```
call basexy       ; устанавливаем координаты x,y игрока.
call wspace       ; выводим пробел поверх изображения игрока.
```

; Теперь после стирания мы можем передвигать игрока, перед тем как вывести его по новым координатам.

```
ld bc,63486       ; группа клавиш 1-5/порт джойстика 2.
in a,(c)          ; смотрим какие клавиши нажаты.
rra               ; крайний бит = клавиша 1.
push af           ; запоминаем значение.
call nc,mpl       ; если была нажата, двигаем влево.
pop af            ; восстанавливаем аккумулятор.
rra               ; следующий бит (значение 2) = клавиша 2.
push af           ; запоминаем значение.
call nc,mpr       ; если была нажата, двигаем вправо.
pop af            ; восстанавливаем аккумулятор.
rra               ; следующий бит (значение 4) = клавиша 3.
push af           ; запоминаем значение.
call nc,mpd       ; если была нажата, двигаем вниз.
pop af            ; восстанавливаем аккумулятор.
rra               ; следующий бит (значение 8) считывает клавишу 4.
call nc,mpr       ; если была нажата, двигаем вверх.
```

; После передвижения игрока можем снова отобразить его.

```

        call basexy          ; устанавливаем координаты x,y игрока.
        call splayr         ; выводим игрока.

        halt                ; задержка.

; Переходим на начало основного цикла.

        jp mloop

; Движение игрока влево.

mpl      ld hl,ply          ; запомните, y координата по горизонтали!
        ld a,(hl)          ; каково текущее значение?
        and a              ; оно равно нулю?
        ret z              ; если да – мы не можем больше идти влево.
        dec (hl)           ; вычитаем 1 из координаты y.
        ret

; Движение игрока вправо.

mpr      ld hl,ply          ; запомните, y координата по горизонтали!
        ld a,(hl)          ; каково текущее значение?
        cp 31              ; это правый край экрана (31)?
        ret z              ; если да – мы не можем больше идти вправо.
        inc (hl)           ; прибавляем 1 к координате y.
        ret

; Движение игрока вверх.

mru      ld hl,plx          ; запомните, x координата по вертикали!
        ld a,(hl)          ; каково текущее значение?
        cp 4               ; это верхний край экрана (4)?
        ret z              ; если да – не можем двигаться выше.
        dec (hl)           ; вычитаем 1 из координаты x.
        ret

; Движение игрока вниз.

mpd      ld hl,plx          ; запомните, x координата по вертикали!
        ld a,(hl)          ; каково текущее значение?
        cp 21              ; это нижний край экрана (21)?
        ret z              ; если да – не можем двигаться ниже.
        inc (hl)           ; прибавляем 1 к координате x.
        ret

; Установка координат x, y для позиции игрока.
; Эта процедура вызывается перед выводом и стиранием изображения игрока.

basexy  ld a,22             ; управляющий ASCII-код AT.
        rst 16
        ld a,(plx)         ; x координата игрока.
        rst 16             ; устанавливаем ее.
        ld a,(ply)         ; y координата игрока.
        rst 16             ; устанавливаем ее.
        ret

; Вывод игрока в текущей позиции печати.

splayr  ld a,69             ; голубые чернила (5) на черной бумаге (0),
                             ; яркость (64).
        ld (23695),a       ; устанавливаем наши временные цвета.
        ld a,144           ; ASCII-код для символа UGD 'A'.
        rst 16             ; выводим изображение игрока.
        ret

wspace  ld a,71             ; белые чернила (7) на черной бумаге (0),
                             ; яркость (64).
        ld (23695),a       ; устанавливаем наши временные цвета.
        ld a,32            ; символ пробела.
        rst 16             ; выводим пробел.
        ret

plx      defb 0             ; x координата игрока.
ply      defb 0             ; y координата игрока.

```

; UDG графика.

blocks defb 16,16,56,56,124,124,254,254 ; изображение игрока.

Слишком быстро, не правда ли? Мы, конечно, замедлили цикл с помощью инструкции **halt**, но этого оказалось недостаточно. Программа выдает 50 кадров в секунду, что довольно много. Но не переживайте, по мере добавления дополнительных возможностей, она будет замедляться. Если вы чувствуете себя достаточно уверенно, то можете попытаться адаптировать ее для работы с Kempston-джойстиком. Это нетрудно, и требует всего лишь изменения порта 63486 на порт 31 и замены четырех последовательных **call nc, (address)** на **call c,(address)**. (Вы ведь помните, что биты в этом случае устанавливаются по-другому?)

А вот сделать переназначаемые клавиши немного сложнее. Как вы уже знаете, клавиатура оригинального Спектрума разделена на восемь групп, по пять клавиш в каждой. Считывая значения битов d0-d4 из соответствующего этой группе порта, мы можем сказать, какие клавиши были нажаты. Если в нашем коде заменить **ld bc, 31** на **ld bc, 49150**, то можно проверять нажатия клавиш от H до Enter. Однако так нам не сделать удобную процедуру переназначения клавиш. К счастью, есть другой выход.

Мы можем установить необходимый адрес порта для каждой группы клавиш, используя формулу из руководства пользователя Спектрума. Адрес порта будет  $254 + 256 * (255 - 2^n)$ , где n – это номер группы, принимающий значения от 0 до 7. Так же в ПЗУ, по адресу 654, есть очень полезная подпрограмма, которая возвращает код нажатой клавиши (от 0 до 39) в регистр **e**. Коды от 0 до 7 – это клавиши, которые в каждой группе ближе к середине (т.е. В, Н, Y, 6, 5, Т, G и V), 8-15 – следующие за ними, а 39 – крайняя клавиша последней группы – CAPS SHIFT. Кстати, статус клавиши SHIFT так же возвращается в регистр **d**. Значение 255 говорит о том, что не нажата ни одна клавиша.

Данная подпрограмма ПЗУ может возвращать код только одной клавиши, и не подходит для определения нескольких одновременных нажатий. Тогда для определения нажатия какой-либо клавиши в любой момент времени, нам необходимо конвертировать полученный код в адрес порта и номер бита, а затем считать оттуда значение. Для этого я использую одну очень удобную процедуру. Это единственная процедура в моих играх, написанная не мной. Выражаю за нее свою признательность Стивену Джонсу, программисту, который много лет назад писал отличные статьи для «Spectrum Discovery Club». Для работы с этой подпрограммой, загрузите в аккумулятор код клавиши, нажатие которой вы хотите проверить, вызовите **ktest**, и проверьте флаг переноса. Если он установлен, то клавиша не была нажата, если сброшен – то была. Если вам этот способ кажется запутанным и не слишком правильным, поставьте инструкцию **ccf** перед **ret**.

; Процедура проверки клавиатуры мистера Джонса.

```
ktest  ld c,a          ; код клавиши помещаем в c.
      and 7            ; оставляем только биты d0-d2,
                      ; для определения номера группы.
      inc a            ; прибавляем 1 чтобы получить значение от 1 до 8.
      ld b,a          ; загружаем номер порта в b.
      srl c            ; делим c на 8,
                      ; чтобы найти позицию клавиши внутри группы.
      srl c            ; значение позиции будет от 0 до 4.
      srl c            ; в группе только пять клавиш.
      ld a,5          ; вычитаем из аккумулятора значение позиции,
      sub c            ; и помещаем его в регистр c
      ld c,a
```



	ld a,254	; (прим. перев. - результатом будет число от 1 до 5).
ktest0	rrca	; старший байт порта, из которого будем читать.
		; с помощью сдвига вычисляем
	djnz ktest0	; старший байт адреса порта.
		; повторяем, пока не найдем соответствующую группу
	in a,(254)	; (прим. перев. - номер группы у нас хранится в b).
ktest1	rra	; читаем порт (a=старший байт, 254=младший).
	dec c	; сдвигаем биты в полученном результате.
		; счетчик цикла.
	jp nz,ktest1	; (прим. перев. - в регистре c число от 1 до 5).
		; повторяем, пока нужный бит не окажется
	ret	; во флаге переноса.

## Глава 3 – Звуковые эффекты

### Динамик

На Спектруме есть два решения для воспроизведения звуков. Более качественный и сложный – использование звукового чипа AY38912 на моделях 128К, этот способ мы обсудим в последующих главах. Сейчас же остановимся на динамике Спектрума 48К. Может такой способ и слишком прост, но все-таки позволяет добиться неплохих звуков в игре.

### ВЕЕР

Прежде всего нам нужно знать, как воспроизвести звук определенной частоты и длительности. В ПЗУ Спектрума для выполнения этой работы есть подпрограмма. Находится она по адресу 949, и все что нужно, так это передать параметры для частоты в регистровую пару **HL** и длительности в **DE**. После вызова **call 949** мы услышим соответствующий звук.

К сожалению, передача необходимых параметров звука усложняется тем, что приходится произвести некоторые вычисления. Нужно знать значение частоты воспроизводимого звука в герцах, попросту говоря – количество колебаний динамика в секунду, для того, чтобы получить желаемый звук. Вот небольшая таблица соответствия:

До 1 окт.	261.63
До-диез	277.18
Ре	293.66
Ре-диез	311.13
Ми	329.63
Фа	349.23
Фа-диез	369.99
Соль	392.00
Соль-диез	415.30
Ля	440.00
Ля-диез	466.16
Си	493.88

Для получения нот из октавы выше – просто удвойте частоту, для октавы ниже – поделите пополам. Например, чтобы воспроизвести ноту До второй октавы, мы удваиваем 261.63, и получаем 523.26.

После того, как мы определились с частотой – нужно умножить ее на длительность в секундах и загрузить результат в регистровую пару **DE**. Так, для ноты До первой октавы длительностью в одну десятую секунды, необходимое значение будет равно  $261.63 * 0.1 = 26.163$ . Высота звука высчитывается путем деления 437500 на частоту, вычитания 30.125 и передачи результата в **HL**. Для До первой октавы это будет означать  $437500 / 261.63 - 30.125 = 1642$ .

Другими словами:

$DE = \text{Длительность} = \text{Частота} * \text{Секунды}$

$HL = \text{Высота ноты} = 437500 / \text{Частота} - 30.125$

Воспроизводим ноту Соль-диез второй октавы в течение одной четвертой секунды:

```
; частота Соль-диез первой октавы = 415.30
; частота Соль-диез второй октавы = 830.60
; значение параметра длительности = 830.6 / 4 = 207.65
; высота = 437500 / 830.6 - 30.125 = 496.6
```

```
ld hl,497          ; высота.
ld de,208          ; длительность.
call 949           ; вызов подпрограммы ПЗУ.
ret
```

Конечно же, подпрограмма ПЗУ служит не только для воспроизведения музыкальных нот, мы можем использовать ее для ряда других эффектов. Вот, например, простой эффект «подтяжки» (бенда):

```
loop    ld hl,500      ; начальная высота.
        ld b,250       ; длительность бенда.
        push bc
        push hl        ; сохраняем высоту.
        ld de,1        ; очень короткая длительность.
        call 949       ; процедура ПЗУ.
        pop hl         ; восстанавливаем высоту.
        inc hl          ; увеличиваем ее.
        pop bc
        djnz loop      ; повторяем.
        ret
```

Поэкспериментируйте с этой программой. Изменяя начальную частоту, шаг и длительность можно получить некоторые интересные эффекты. Однако, должен предупредить вас – не перестарайтесь со значениями высоты и длительности, иначе процедура ПЗУ зависнет, и вам придется нажимать «сброс».

## Белый шум

Работая с динамиком, не обязательно использовать только процедуры из ПЗУ. Довольно легко написать собственные процедуры эффектов, особенно если мы хотим симитировать звуки ударов или взрывов. Белый шум – довольно интересная и подходящая штука для таких экспериментов.

Чтобы сгенерировать белый шум, нам понадобится простой генератор случайных чисел. Подойдет последовательность Фибоначчи, но я бы порекомендовал пройтись указателем по начальным 8Кб ПЗУ и брать значения байт оттуда, чтобы получить псевдослучайные 8-битные числа. После этого следует записать их значения в порт 254. Помните, что этот порт отвечает так же за цвет бордюра. Поэтому, если вам не нужны цветные полосы на бордюре – маскируем биты бордюра командой **AND 248** и прибавляем соответствующий код цвета (1 – синий, 2 – красный и т.д.) перед выполнением инструкции **OUT (254)**. Затем запустим небольшой цикл задержки (для высоких тонов задержка поменьше, для низких – побольше), и повторим процесс пару сотен раз. И у нас получится неплохой эффект удара.

Вот пример процедуры звукового эффекта из игры Egghead 3:

```
noise  ld e,250          ; повторяем 250 раз.
      ld hl,0            ; указываем на начало ПЗУ.
noise2  push de
      ld b,32            ; длина шага.
noise0  push bc
      ld a,(hl)          ; следующее "случайное" число.
      inc hl             ; указатель на следующий адрес.
      and 248            ; нам нужен черный бородюр.
      out (254),a        ; выводим на динамик.
      ld a,e             ; по мере уменьшения е...
      cpl               ; ...мы увеличиваем задержку.
noise1  dec a            ; уменьшаем счетчик цикла.
      jr nz,noise1       ; цикл задержки.
      pop bc
      djnz noise0        ; следующий шаг.
      pop de
      ld a,e
      sub 24             ; размер шага.
      cp 30              ; конец диапазона.
      ret z
      ret c
      ld e,a
      cpl
noise3  ld b,40          ; делаем паузу.
noise4  djnz noise4
      dec a
      jr nz,noise3
      jr noise2
```

## Глава 4 – Случайные числа

Написание программы генерации случайных чисел в машинных кодах – может быть довольно сложной задачей для начинающего программиста.

Но для начала проясним кое-что. Нет такой вещи как генератор случайных чисел. Центральный процессор четко следует инструкциям, и не имеет собственного разума, чтобы по желанию взять из ниоткуда число. Вместо этого мы дадим ему некую формулу, выдающую непредсказуемую последовательность чисел, которые только *кажутся* случайными. Таким образом, мы получим всего лишь подделку, т.е. *псевдослучайное* число.

Один из способов получения псевдослучайных чисел – использование чисел Фибоначчи. Однако наиболее легкий способ получить псевдослучайное восьмибитное число на Спектруме – последовательный перебор содержимого байтов участка ПЗУ. У этого метода, однако, есть свой недостаток. К концу своего адресного пространства ПЗУ содержит очень однородные и не кажущиеся случайными данные, чего лучше избегать. Однако, ограничив используемый участок ПЗУ, скажем, начальными 8Кб мы все еще имеем в распоряжении последовательность из 8192 «случайных» чисел, чего вполне достаточно для большинства игр. Фактически я использовал этот, или очень похожий метод во всех играх, где требовался генератор случайных чисел.

```
; Простой генератор псевдослучайных чисел.  
; Проходит указателем по адресам ПЗУ (хранится в seed), возвращает  
; их содержимое.
```

```
random 1d h1,(seed)      ; Указатель  
        1d a,h           ;  
        and 31           ; используем только первые 8Кб ПЗУ.  
        1d h,a           ;  
        1d a,(h1)        ; Получаем «случайное» число по этому адресу.  
        inc h1           ; Увеличиваем указатель.  
        1d (seed),h1  
        ret  
seed    defw 0
```

Давайте используем этот генератор в нашей игре. В каждом клоне «Centipede» должны быть грибы, очень много грибов, рассыпанных по игровому полю в случайном порядке. Теперь мы можем вызывать процедуру генератора псевдослучайных чисел для определения координат каждого такого гриба. Подчеркнутые строки – это то, что необходимо добавить к уже готовым частям кода.

```
; Устанавливаем черный экран.
```

```
        1d a,71          ; белые чернила (7) на черной бумаге (0),  
                        ; яркость (64).  
        1d (23693),a     ; устанавливаем наши цвета.  
        xor a            ; быстрый способ загрузить ноль в аккумулятор.  
        call 8859        ; устанавливаем постоянный цвет бордюра.
```

```
; Настраиваем графику.
```

```
        1d h1,blocks     ; Адрес где хранится пользовательская графика.  
        1d (23675),h1    ; теперь переменная будет указывать на него.
```

```
; ОК, начнем игру.
```

```
        call 3503        ; Процедура ПЗУ, очищаем экран, открываем канал 2.
```

; Инициализируем координаты.

```
ld hl,21+15*256 ; загружаем в пару hl начальные координаты.
ld (plx),hl      ; устанавливаем координаты игрока .

call basexy      ; устанавливаем координаты x,y игрока.
call splayr      ; выводим изображение игрока.
```

; Теперь заполняем игровое поле грибами.

```
ld a,68          ; зеленые чернила (4) на черной бумаге (0).
                 ; яркость (64).
ld (23695),a     ; устанавливаем наши временные цвета.
ld b,50          ; начнем с небольшого количества.
mushlp ld a,22    ; управляющий ASCII код для AT.
rst 16
call random      ; получаем «случайное» число.
and 15           ; по вертикали от 0 до 15.
rst 16
call random      ; еще одно псевдослучайное число.
and 31           ; по горизонтали от 0 до 31.
rst 16
ld a,145         ; Символ UDG 'В' - изображение гриба.
rst 16           ; выводим гриб на экран.
djnz mushlp      ; повторяем, пока не выведем все грибы.
```

; Это основной цикл.

mloop equ \$

; Удаляем изображение игрока.

```
call basexy      ; устанавливаем координаты x,y игрока.
call wspace      ; Выводим пробел поверх изображения игрока.
```

; Теперь, когда удалили игрока – можно переместить его и вывести  
; его изображение по новым координатам.

```
ld bc,63486      ; группа клавиш 1-5/порт джойстика 2.
in a,(c)         ; смотрим какие клавиши нажаты.
rra              ; крайний бит = клавиша 1.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, двигаемся влево.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 2) = клавиша 2.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, двигаемся вправо.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 4) = клавиша 3.
push af          ; запоминаем значение.
call nc,mpd      ; была нажата, двигаемся вниз.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 8) считываем клавишу 4.
call nc,mpu      ; была нажата, двигаемся вверх.
```

; После передвижения, можно снова вывести изображение на экран.

```
call basexy      ; устанавливаем координаты x,y игрока.
call splayr      ; выводим изображение.

halt             ; задержка.
```

; Переход на начало основного цикла.

jr mloop

; Двигаем игрока влево.

```
mp1 ld hl,ply     ; помните, у это координата по горизонтали!
ld a,(hl)        ; каково текущее значение?
and a            ; это ноль?
ret z            ; если да – мы больше не можем двигаться влево.
```

```

        dec (hl)          ; вычитаем 1 из координаты у.
        ret

; Двигаем игрока вправо.

mpr      ld hl,ply        ; помните, у это координата по горизонтали!
        ld a,(hl)        ; каково текущее значение?
        cp 31            ; это правый край экрана (31)?
        ret z            ; если да – мы больше не можем двигаться вправо.
        inc (hl)         ; прибавляем 1 к координате у.
        ret

; Двигаем игрока вверх.

mpr      ld hl,plx        ; помните, х это координата по вертикали!
        ld a,(hl)        ; каково текущее значение?
        cp 4             ; это верхний край экрана (4)?
        ret z            ; если да – нельзя двигаться выше.
        dec (hl)         ; вычитаем 1 из координаты х.
        ret

; Двигаем игрока вниз.

mpd      ld hl,plx        ; помните, х это координата по вертикали!
        ld a,(hl)        ; каково текущее значение?
        cp 21            ; это нижний край экрана (21)?
        ret z            ; если да – нельзя больше двигаться вниз.
        inc (hl)         ; прибавляем 1 к координате х.
        ret

; Устанавливает координаты х,у игрока на экране, эту процедуру необходимо
; вызывать перед выводом изображения игрока, или его удалением.

basexy  ld a,22           ; управляющий ASCII-код АТ.
        rst 16
        ld a,(plx)       ; координата игрока по вертикали.
        rst 16
        ld a,(ply)       ; координата игрока по горизонтали.
        rst 16
        ret              ; устанавливаем позицию по горизонтали.

; Вывод изображения игрока по текущим координатам.

splayr  ld a,69           ; голубые чернила (5) на черной бумаге (0),
                        ; яркость (64).
        ld (23695),a     ; устанавливаем временные цвета.
        ld a,144         ; ASCII-код для символа 'A' UDG.
        rst 16           ; выводим изображение игрока.
        ret

wspace  ld a,71           ; белые чернила (7) на черной бумаге (0),
                        ; яркость (64).
        ld (23695),a     ; устанавливаем наши временные цвета.
        ld a,32          ; символ пробела.
        rst 16           ; выводим пробел.
        ret

; Простой генератор псевдослучайных чисел.
; Проходит указателем по адресам ПЗУ (хранится в seed), возвращает
; их содержимое.

random  ld hl,(seed)      ; Указатель
        ld a,h
        and 31           ; Используем только первые 8Кб ПЗУ.
        ld h,a
        ld a,(hl)        ; Получаем «случайное» число по этому адресу.
        inc hl           ; Увеличиваем указатель.
        ld (seed),hl
        ret
seed    defw 0

plx     defb 0            ; координата х игрока.
ply     defb 0            ; координата у игрока

```

; UDG-графика.

```
blocks defb 16,16,56,56,124,124,254,254    ; игрок.  
       defb 24,126,255,255,60,60,60,60      ; гриб.
```

---

После запуска этого кода мы увидим, что теперь наша игра выглядит гораздо более похожей на «Centipede». Но есть одна большая проблема: хоть грибы и разбросаны по полю в случайном порядке – игрок может проходить сквозь них. Чтобы это предотвратить, нам понадобится процедура проверки столкновений. О ней мы и поговорим в следующей главе.



## Глава 5 - Простейшее определение столкновений

### Определяем атрибуты

Каждый, кто некоторое время программировал на Sinclair BASIC, должен хорошо помнить функцию ATTR. Она позволяет определить цветовые атрибуты любого знакоместа на экране. И, хотя начинающему программисту на BASIC трудновато понять ее работу, она может быть очень полезна для простого определения столкновений. Фактически, этот метод оказался настолько хорош, что его вариацию в машинных кодах использовали в ряде коммерческих игр. Естественно, он очень пригодится и начинающему программисту на Спектрум.

Определить атрибуты конкретного знакоместа можно двумя способами. Если взглянуть на дизассемблированный код ПЗУ Спектрума, то можно найти процедуру по адресу 9603, которая сделает всю работу за нас, либо можно заняться вычислениями самостоятельно.

Простейший способ определения атрибутов – использование парочки процедур ПЗУ:

```
ld bc,(ballx)      ; помещаем координаты x,y в регистровую пару bc.
call 9603           ; эта процедура ПЗУ помещает атрибуты (c,b) на
                   ; вершину стека.
call 11733          ; помещаем атрибуты в аккумулятор.
```

Однако, гораздо быстрее произвести вычисления самому. Так же, полезно найти не только значение, но и *адрес*, в случае, если мы вдруг захотим что-то туда записать.

### Вычисление адресов атрибутов

Цветовые атрибуты знакомест, занимают адреса с 22528 по 23295 включительно и, в отличие от пикселей, довольно логично расположены в оперативной памяти Спектрума. Т.е. с адреса 22528 по 22559 расположены 32 атрибута знакомест верхней строки экрана, с 22560 по 22591 – атрибуты второй строки, и так далее. Чтобы вычислить адрес ячейки атрибутов знакоместа находящегося по координатам **x**, **y**, нам необходимо всего лишь умножить **x** на 32, прибавить **y**, и добавить 22528 к результату. Проверив содержимое по этому адресу, мы узнаем цветовые атрибуты в заданной позиции. Следующий пример вычисляет адрес ячейки атрибутов символа, координаты которого находятся в **bc**, и помещает результат в регистровую пару **hl**.

; Вычисляем адрес атрибутов символа (координаты в bc).

```
atadd  ld a,b          ; координата x.
      rrca             ; умножаем на 32.
      rrca
      rrca
      ld l,a           ; сохраняем в l.
      and 3            ; битовая маска для старшей части.
      add a,88          ; 88*256=22528, начальный адрес атрибутов.
      ld h,a           ; со старшей частью закончили, помещаем в h.
      ld a,l           ; нам снова нужен x*32.
      and 224          ; маскируем младшую часть.
      ld l,a           ; помещаем в l.
      ld a,c           ; координата y.
      add a,l          ; прибавляем к младшей части.
      ld l,a           ; теперь в hl адрес атрибутов.
      ld a,(hl)        ; возвращаем атрибут в a.
      ret
```

Проверка содержимого байта по адресу в **hl** даст нам значение атрибутов, тогда как запись по этому адресу поменяет цвета знакоместа.

Каждый атрибут состоит из 8 бит, которые назначены следующим образом:

d0-d2	цвет чернил 0-7,	0=черный, 1=синий, 2=красный, 3=пурпурный, 4=зеленый, 5=голубой, 6=желтый, 7=белый
d3-d5	цвет бумаги 0-7,	0=черный, 1=синий, 2=красный, 3=пурпурный, 4=зеленый, 5=голубой, 6=желтый, 7=белый
d6	яркость,	0=обычный, 1=яркий
d7	мигание,	0=обычный, 1=мигающий

Например, проверка на зеленый цвет бумаги может выглядеть так:

```
and 56          ; маскируем все, кроме битов бумаги.  
cp 32           ; это зеленый (4) * 8?  
jr z,green      ; если да, идем на обработку зеленого.
```

Тогда как проверка на желтый цвет чернил будет такой:

```
and 7           ; маскируем все, кроме битов чернил.  
cp 6            ; желтый цвет (6)?  
jr z,yellow     ; да, уходим на обработку желтого.
```

## Применяем изученное к нашей игре

Теперь мы можем добавить проверку на столкновения по атрибутам к нашей игре. Как и ранее, новые строки выделены подчеркиванием.

; Устанавливаем черный экран.

```
ld a,71          ; белые чернила (7) на черной бумаге (0),  
                  ; яркость (64).  
ld (23693),a     ; устанавливаем наши цвета.  
xor a            ; быстрый способ загрузить ноль в аккумулятор.  
call 8859        ; устанавливаем постоянный цвет бордюра.
```

; Настраиваем графику.

```
ld hl,blocks     ; Адрес где хранится пользовательская графика.  
ld (23675),hl    ; теперь переменная будет указывать на него.
```

; ОК, начнем игру.

```
call 3503        ; Процедура ПЗУ, очищаем экран, открываем канал 2.
```

; Инициализируем координаты.

```
ld hl,21+15*256  ; загружаем в пару hl начальные координаты.  
ld (plx),hl      ; устанавливаем координаты игрока .  
  
call basexy      ; устанавливаем координаты x,y игрока.  
call spayr       ; выводим изображение игрока.
```

; Теперь заполняем игровое поле грибами.

```
ld a,68          ; зеленые чернила (4) на черной бумаге (0),  
                  ; яркость (64).  
ld (23695),a     ; устанавливаем наши временные цвета.  
ld b,50          ; начнем с небольшого количества.
```

```

mushlp ld a,22          ; управляющий ASCII код для AT.
      rst 16
      call random       ; получаем «случайное» число.
      and 15            ; по вертикали от 0 до 15.
      rst 16
      call random       ; еще одно псевдослучайное число.
      and 31           ; по горизонтали от 0 до 31.
      rst 16
      ld a,145          ; Символ UDG 'В' - изображение гриба.
      rst 16           ; выводим гриб на экран.
      djnz mushlp      ; повторяем, пока не выведем все грибы.

; Это основной цикл.

mloop equ $

; Удаляем изображение игрока.

      call basexy       ; устанавливаем координаты x,y игрока.
      call wspace      ; Выводим пробел поверх изображения игрока.

; Теперь, когда удалили игрока – можно переместить его и вывести
; его изображение по новым координатам.

      ld bc,63486       ; группа клавиш 1-5/порт джойстика 2.
      in a,(c)          ; смотрим какие клавиши нажаты.
      rra               ; крайний бит = клавиша 1.
      push af           ; запоминаем значение.
      call nc,mp1       ; была нажата, двигаемся влево.
      pop af            ; восстанавливаем аккумулятор.
      rra               ; следующий бит (значение 2) = клавиша 2.
      push af           ; запоминаем значение.
      call nc,mp1       ; была нажата, двигаемся вправо.
      pop af            ; восстанавливаем аккумулятор.
      rra               ; следующий бит (значение 4) = клавиша 3.
      push af           ; запоминаем значение.
      call nc,mp1       ; была нажата, двигаемся вниз.
      pop af            ; восстанавливаем аккумулятор.
      rra               ; следующий бит (значение 8) считываем клавишу 4.
      call nc,mp1       ; была нажата, двигаемся вверх.

; После передвижения, можно снова вывести изображение на экран.

      call basexy       ; устанавливаем координаты x,y игрока.
      call splayr       ; выводим изображение.

      halt              ; задержка.

; Переход на начало основного цикла.

      jp mloop

; Двигаем игрока влево.

mp1   ld hl,ply         ; помните, у это координата по горизонтали!
      ld a,(hl)         ; каково текущее значение?
      and a              ; это ноль?
      ret z             ; если да – мы больше не можем двигаться влево.

;Проверяем, нет ли на пути гриба.

      ld bc,(plx)       ; текущие координаты.
      dec b              ; проверяем знакоместо слева.
      call atadd         ; получаем адрес атрибутов в этой позиции.
      cp 68              ; грибы яркие (64) + зеленые (4).
      ret z             ; если гриб есть – не можем туда двигаться.

      dec (hl)           ; вычитаем 1 из координаты y.
      ret

; Двигаем игрока вправо.

```

```

mpr    ld hl,ply          ; помните, у это координата по горизонтали!
        ld a,(hl)         ; каково текущее значение?
        cp 31             ; это правый край экрана (31)?
        ret z             ; если да – мы больше не можем двигаться вправо.

```

; Проверяем, нет ли на пути гриба..

```

        ld bc,(plx)       ; текущие координаты.
        inc b             ; проверяем знакоместо справа.
        call atadd        ; получаем адрес атрибутов в этой позиции.
        cp 68             ; грибы яркие (64) + зеленые (4).
        ret z             ; если гриб есть – не можем туда двигаться.

```

```

        inc (hl)          ; прибавляем 1 к координате у.
        ret

```

; Двигаем игрока вверх.

```

mpr    ld hl,plx          ; помните, х это координата по вертикали!
        ld a,(hl)         ; каково текущее значение?
        cp 4              ; это верхний край экрана (4)?
        ret z             ; если да – нельзя двигаться выше.

```

; Проверяем, нет ли на пути гриба.

```

        ld bc,(plx)       ; текущие координаты.
        dec c             ; проверяем знакоместо сверху.
        call atadd        ; получаем адрес атрибутов в этой позиции.
        cp 68             ; грибы яркие (64) + зеленые (4).
        ret z             ; если гриб есть – не можем туда двигаться.

```

```

        dec (hl)          ; вычитаем 1 из координаты х.
        ret

```

; Двигаем игрока вниз.

```

mpd    ld hl,plx          ; помните, х это координата по вертикали!
        ld a,(hl)         ; каково текущее значение?
        cp 21             ; это нижний край экрана (21)?
        ret z             ; если да – нельзя больше двигаться вниз.

```

; Проверяем, нет ли на пути гриба.

```

        ld bc,(plx)       ; текущие координаты.
        inc c             ; проверяем знакоместо снизу.
        call atadd        ; получаем адрес атрибутов в этой позиции.
        cp 68             ; грибы яркие (64) + зеленые (4).
        ret z             ; если гриб есть – не можем туда двигаться.

```

```

        inc (hl)          ; прибавляем 1 к координате х.
        ret

```

; Устанавливает координаты х,у игрока на экране, эту процедуру необходимо  
; вызывать перед выводом изображения игрока, или его удалением.

```

basexy ld a,22            ; управляющий ASCII-код АТ.
        rst 16
        ld a,(plx)        ; координата игрока по вертикали.
        rst 16            ; устанавливаем координату игрока по вертикали.
        ld a,(ply)        ; координата игрока по горизонтали.
        rst 16            ; устанавливаем позицию по горизонтали.
        ret

```

; Вывод изображения игрока по текущим координатам.

```

splayr ld a,69            ; голубые чернила (5) на черной бумаге (0),
                          ; яркость (64).
        ld (23695),a      ; устанавливаем временные цвета.
        ld a,144          ; ASCII-код для символа 'A' UDГ.
        rst 16            ; выводим изображение игрока.
        ret

```

```

wspace ld a,71            ; белые чернила (7) на черной бумаге (0),
                          ; яркость (64).

```

```

ld (23695),a      ; устанавливаем наши временные цвета.
ld a,32           ; символ пробела.
rst 16            ; выводим пробел.
ret

```

```

; Простой генератор псевдослучайных чисел.
; Проходит указателем по адресам ПЗУ (хранится в seed), возвращает
; их содержимое.

```

```

random ld hl,(seed)      ; Указатель
      ld a,h
      and 31             ; Используем только первые 8Кб ПЗУ.
      ld h,a
      ld a,(hl)          ; Получаем «случайное» число по этому адресу.
      inc hl             ; Увеличиваем указатель.
      ld (seed),hl
      ret
seed   defw 0

```

; Вычисление адреса атрибутов символа с координатами (dispx, dispy).

```

atadd  ld a,c             ; координата по вертикали.
      rrca               ; умножаем на 32.
      rrca               ; Тройной сдвиг вправо с переносом
      rrca               ; быстрее пяти сдвигов влево.
      ld e,a
      and 3
      add a,88            ; 88x256=адрес атрибутов.
      ld d,a
      ld a,e
      and 224
      ld e,a
      ld a,b             ; позиция по горизонтатали.
      add a,e
      ld e,a             ; de=адрес атрибутов.
      ld a,(de)          ; возврат, адрес теперь в аккумуляторе.
      ret

```

```

plx    defb 0             ; координата x игрока.
ply    defb 0             ; координата y игрока

```

; UDG-графика.

```

blocks defb 16,16,56,56,124,124,254,254 ; игрок.
      defb 24,126,255,255,60,60,60,60   ; гриб.

```

## Глава 6 - Таблицы

### Враги не приходят поодиночке

Допустим, нам захотелось написать игру Space Invaders, в которой флот пришельцев выстраивался бы в пять рядов по одиннадцать кораблей в каждом. Было бы непрактично писать код для каждого из 55 пришельцев поочередно. Поэтому нам нужно организовать таблицу. В Sinclair BASIC это можно сделать объявив три массива из 55 элементов. Один – для координат **x**, другой – **y**, а третий – массив байтов статуса. Примерно то же самое мы можем сделать и на ассемблере, организовав в памяти три таблицы по 55 байтов, и используя определенное смещение относительно начального адреса таблицы для каждого пришельца, чтобы получить доступ к конкретному элементу. Однако это очень неуклюжий и медленный способ.

Лучшим решением будет объединить все три элемента данных каждого пришельца в структуру, а затем из них организовать таблицу. Так, например, мы сможем поместить в **hl** адрес байта статуса, **hl+1** будет указывать на координату **x**, **hl+2** – на координату **y**. Код вывода на экран в таком случае может выглядеть примерно так:

```
loop0    ld hl,aliens      ; таблица структур данных по пришельцам.
         ld b,55          ; их количество.
         call show        ; показать текущего.
         djnz loop0       ; повторить для остальных.
         ret
show     ld a,(hl)         ; проверяем байт статуса.
         cp 255           ; пришелец отключен?
         jr z,next        ; если да, то его не выводим.
         push hl          ; сохраняем начальный адрес текущего пришельца.
         inc hl           ; указываем на координату x.
         ld d,(hl)        ; загружаем ее в d.
         inc hl           ; указываем на координату y.
         ld e,(hl)        ; загружаем ее в e.
         call displ       ; выводим пришельца по координатам (d,e).
         pop hl           ; восстанавливаем из стека начальный адрес.
next     ld de,3           ; размер структуры для одного пришельца.
         add hl,de        ; указываем на следующего.
         ret              ; выходим из процедуры, теперь в hl адрес следующего.
```

### Использование индексных регистров

Недостаток этой процедуры заключается в том, что нам нужно постоянно следить за тем, что находится в регистровой паре **hl**. Поэтому, возможно будет неплохой идеей сохранять содержимое **hl** в памяти перед вызовом процедуры **show**, затем его восстанавливать, а в конце главного цикла увеличивать значение на три и выполнять условный переход **djnz**. Если бы мы писали игру для Nintendo GameBoy с его урезанным Z80, тогда это, вероятно, было бы лучшим решением. Но на машинах с более продвинутыми процессорами, таких как Спектрум или CPC464, мы можем использовать индексные регистры для упрощения кода. Регистровая пара **ix** позволяет использовать косвенную адресацию, поэтому мы можем поместить в нее начальный адрес текущей структуры данных и получать доступ ко всем ее элементам, не меняя **ix**. В этом случае наш код будет выглядеть примерно так:

```
loop0    ld ix,aliens      ; таблица структур данных по пришельцам.
         ld b,55          ; их количество.
         call show        ; показать текущего.
         ld de,3          ; размер одной структуры данных.
         add ix,de        ; указываем на следующего пришельца.
         djnz loop0       ; повторить для остальных.
```

```

ret
show  ld a,(ix)      ; проверяем байт статуса.
      cp 255         ; пришелец отключен?
      ret z          ; если да, то его не выводим.
      ld d,(ix+1)     ; загружаем координату.
      ld e,(ix+2)     ; загружаем координату.
      jp displ        ; выводим пришельца по координатам (d,e).

```

С помощью **ix** мы теперь можем указывать лишь начальный адрес структуры данных. В этой регистровой паре всегда будет содержаться адрес текущего пришельца, в **ix+1** координата **x** и т.д. Этот метод позволяет программисту создавать сложные структуры данных размером до 128 байт, не заботясь каждый раз о том на какую часть структуры указывают наши регистры (как в примере с регистром **hl** выше). К сожалению, инструкции для работы с регистровой парой **ix** выполняются немного медленней, чем с **hl**, поэтому, при более ресурсоемких задачах, вроде вывода графики лучше их не использовать.

Давайте используем этот метод в нашей игре. Сначала нам нужно решить, сколько всего у нас будет сегментов, и какие данные будут характеризовать каждый сегмент. Сегменты будут двигаться вправо или влево до столкновения с грибом, затем смещаться вниз, и продолжать движение в обратном направлении. Таким образом, нам понадобятся: флаг для определения направления движения, а так же координаты **x** и **y**. Наш флаг так же можно использовать для определения того уничтожен сегмент, или нет. Принимая это во внимание, создадим структуру данных из трех байт:

```

centf defb 0          ; флаг, 0=влево, 1=вправо, 255=уничтожен.
centx defb 0          ; координата x сегмента.
centy defb 0          ; координата y сегмента.

```

Если мы решим задействовать десять сегментов в нашей игре, то необходимо будет выделить память под таблицу размером в тридцать байт. Каждый сегмент, в процессе игры, нужно будет инициализировать, затем удалять, перемещать, и выводить заново.

Сделать инициализацию сегментов, пожалуй, проще всего. Поэтому мы задействуем простейший цикл, увеличивая содержимое регистровой пары **hl** для доступа к каждому элементу таблицы. Выглядеть это будет примерно так:

```

      ld b,10          ; количество инициализируемых сегментов.
      ld hl,segmnt     ; начальный адрес таблицы сегментов.
segint ld (hl),1        ; начинаем двигаться вправо.
      inc hl
      ld (hl),0         ; начинаем с верхнего края экрана.
      inc hl
      ld (hl),b         ; используем регистр B как координату y.
      inc hl
      djnz segint      ; повторяем для оставшихся сегментов.

```

Обработка и вывод каждого сегмента немного сложнее, поэтому здесь мы используем регистровую пару **ix**. Напишем простой алгоритм, который двигает сегмент влево или вправо до столкновения с грибом, затем перемещает его вниз, и переключает флаг направления движения. Назовем эту процедуру **proseg** ("process segment"), и организуем цикл, который будет указывать на каждый элемент поочередно и вызывать **proseg**. Если мы верно составили алгоритм, то увидим, как наша «сороконожка» ползет вниз, лавируя между грибами. Применительно к коду нашей игры довольно просто – мы проверяем байт флага для каждого сегмента (**ix**) для определения направления движения, в зависимости от его значения увеличиваем либо уменьшаем горизонтальную координату (**ix+2**), проверяем атрибуты

знакоместа. Если это зеленый и черный, то увеличиваем вертикальную координату (**ix**+1), и переключаем флаг направления (**ix**).

Есть еще несколько важных моментов, таких как столкновения с краями экрана, но тут мы просто проверяем координаты сегмента и, переключая флаг направления, смещаемся вниз, либо перемещаемся в верхнюю часть экрана, если достигли нижней. Так же, сегменты необходимо удалять с их старой позиции перед перемещением, а затем выводить снова, но эти вопросы мы уже рассматривали ранее.

Новый код будет таким:

; Устанавливаем черный экран.

```
ld a,71          ; белые чернила (7) на черной бумаге (0),
                  ; яркость (64).
ld (23693),a      ; устанавливаем наши цвета.
xor a             ; быстрый способ загрузить ноль в аккумулятор.
call 8859         ; устанавливаем постоянный цвет бордюра.
```

; Настраиваем графику.

```
ld hl,blocks      ; Адрес где хранится пользовательская графика.
ld (23675),hl     ; теперь переменная будет указывать на него.
```

; ОК, начнем игру.

```
call 3503         ; Процедура ПЗУ, очищаем экран, открываем канал 2.
```

; Инициализируем координаты.

```
ld hl,21+15*256   ; загружаем в пару hl начальные координаты.
ld (plx),hl       ; устанавливаем координаты игрока .

ld b,10           ; количество инициализируемых сегментов.
ld hl,segmnt      ; начальный адрес таблицы сегментов.
segint ld (hl),1   ; начинаем двигаться вправо.
inc hl
ld (hl),0         ; начинаем с верхнего края экрана.
inc hl
ld (hl),b         ; используем регистр B как координату y.
inc hl
djnz segint       ; повторяем для оставшихся сегментов.

call basexy       ; устанавливаем координаты x,y игрока.
call splayr       ; выводим изображение игрока.
```

; Теперь заполняем игровое поле грибами.

```
ld a,68          ; зеленые чернила (4) на черной бумаге (0),
                  ; яркость (64).
ld (23695),a      ; устанавливаем наши временные цвета.
ld b,50           ; начнем с небольшого количества.
mushlp ld a,22     ; управляющий ASCII код для AT.
rst 16
call random       ; получаем «случайное» число.
and 15           ; по вертикали от 0 до 15.
rst 16
call random       ; еще одно псевдослучайное число.
and 31           ; по горизонтали от 0 до 31.
rst 16
ld a,145         ; Символ UDG 'В' – изображение гриба.
rst 16           ; выводим гриб на экран.
djnz mushlp      ; повторяем, пока не выведем все грибы.
```

; Это основной цикл.

```
mloop equ $
```



; Удаляем изображение игрока.

```
call basexy      ; устанавливаем координаты x,y игрока.
call wspace      ; выводим пробел поверх изображения игрока.
```

; Теперь, когда удалили игрока – можно переместить его и вывести  
; его изображение по новым координатам.

```
ld bc,63486      ; группа клавиш 1-5/порт джойстика 2.
in a,(c)         ; смотрим какие клавиши нажаты.
rra              ; крайний бит = клавиша 1.
push af          ; запоминаем значение.
call nc,mp1      ; была нажата, движемся влево.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 2) = клавиша 2.
push af          ; запоминаем значение.
call nc,mpr      ; была нажата, движемся вправо.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 4) = клавиша 3.
push af          ; запоминаем значение.
call nc,mpd      ; была нажата, движемся вниз.
pop af           ; восстанавливаем аккумулятор.
rra              ; следующий бит (значение 8) считываем клавишу 4.
call nc,mpu      ; была нажата, движемся вверх.
```

; После передвижения, можно снова вывести изображение на экран.

```
call basexy      ; устанавливаем координаты x,y игрока.
call splayr      ; выводим изображение.
```

; Теперь сегменты сороконожки.

```
ld ix,segmnt      ; таблица данных сегментов.
ld b,10           ; количество сегментов в таблице.
censeg push bc
ld a,(ix)         ; сегмент включен?
inc a             ; выключен это 255, inc 255=0.
call nz,proseg    ; включен (в а не ноль) – обрабатываем сегмент.
pop bc
ld de,3           ; 3 байта на сегмент.
add ix,de         ; в ix адрес следующего сегмента.
djnz censeg       ; повторить для всех сегментов.

halt              ; задержка.
```

; Переход на начало основного цикла.

```
jp mloop
```

; Двигаем игрока влево.

```
mp1 ld hl,ply      ; помните, у это координата по горизонтали!
ld a,(hl)         ; каково текущее значение?
and a             ; это ноль?
ret z             ; если да – мы больше не можем двигаться влево.
```

;Проверяем, нет ли на пути гриба.

```
ld bc,(plx)       ; текущие координаты.
dec b             ; проверяем знакоместо слева.
call atadd        ; получаем адрес атрибутов в этой позиции.
cp 68             ; грибы яркие (64) + зеленые (4).
ret z             ; если гриб есть – не можем туда двигаться.

dec (hl)          ; вычитаем 1 из координаты y.
ret
```

; Двигаем игрока вправо.

```
mpr ld hl,ply      ; помните, у это координата по горизонтали!
ld a,(hl)         ; каково текущее значение?
cp 31             ; это правый край экрана (31)?
```

```

    ret z                ; если да – мы больше не можем двигаться вправо.
; Проверяем, нет ли на пути гриба..

    ld bc,(plx)          ; текущие координаты.
    inc b                ; проверяем знакоместо справа.
    call atadd           ; получаем адрес атрибутов в этой позиции.
    cp 68                ; грибы яркие (64) + зеленые (4).
    ret z                ; если гриб есть – не можем туда двигаться.

    inc (hl)             ; прибавляем 1 к координате y.
    ret

; Двигаем игрока вверх.

mpi    ld hl,plx          ; помните, x это координата по вертикали!
        ld a,(hl)         ; каково текущее значение?
        cp 4              ; это верхний край экрана (4)?
        ret z             ; если да – нельзя двигаться выше.

; Проверяем, нет ли на пути гриба.

    ld bc,(plx)          ; текущие координаты.
    dec c                ; проверяем знакоместо сверху.
    call atadd           ; получаем адрес атрибутов в этой позиции.
    cp 68                ; грибы яркие (64) + зеленые (4).
    ret z                ; если гриб есть – не можем туда двигаться.

    dec (hl)             ; вычитаем 1 из координаты x.
    ret

; Двигаем игрока вниз.

mpd    ld hl,plx          ; помните, x это координата по вертикали!
        ld a,(hl)         ; каково текущее значение?
        cp 21             ; это нижний край экрана (21)?
        ret z             ; если да – нельзя больше двигаться вниз.

; Проверяем, нет ли на пути гриба.

    ld bc,(plx)          ; текущие координаты.
    inc c                ; проверяем знакоместо снизу.
    call atadd           ; получаем адрес атрибутов в этой позиции.
    cp 68                ; грибы яркие (64) + зеленые (4).
    ret z                ; если гриб есть – не можем туда двигаться.

    inc (hl)             ; прибавляем 1 к координате x.
    ret

; Устанавливает координаты x,y игрока на экране, эту процедуру необходимо
; вызывать перед выводом изображения игрока, или его удалением.

basexy ld a,22            ; управляющий ASCII-код AT.
        rst 16
        ld a,(plx)        ; координата игрока по вертикали.
        rst 16            ; устанавливаем координату игрока по вертикали.
        ld a,(ply)        ; координата игрока по горизонтали.
        rst 16            ; устанавливаем позицию по горизонтали.
        ret

; Вывод изображения игрока по текущим координатам.

splayr ld a,69            ; голубые чернила (5) на черной бумаге (0),
                           ; яркость (64).
        ld (23695),a      ; устанавливаем временные цвета.
        ld a,144          ; ASCII-код для символа 'A' UDG.
        rst 16            ; выводим изображение игрока.
        ret

wspace ld a,71            ; белые чернила (7) на черной бумаге (0),
                           ; яркость (64).
        ld (23695),a      ; устанавливаем наши временные цвета.
        ld a,32           ; символ пробела.
        rst 16            ; выводим пробел.

```

```

ret

segxy  ld a,22          ; Управляющей ASCII-код AT.
      rst 16           ; вывести код AT.
      ld a,(ix+1)       ; координата x сегмента.
      rst 16           ; вывести код координаты.
      ld a,(ix+2)       ; координата y сегмента.
      rst 16           ; вывести код координаты.
      ret

proseg ld a,(ix)        ; проверяем был ли сегмент выключен
      inc a            ; процедурой проверки столкновений.
      ret z            ; был, значит этот сегмент теперь уничтожен.
      call segxy       ; устанавливаем координаты сегмента.
      call wspace      ; выводим пробел, белые чернила на черном.
      call segmov      ; перемещаем сегмент.
      ld a,(ix)        ; проверяем был ли сегмент выключен
      inc a            ; процедурой проверки столкновений.
      ret z            ; был, значит этот сегмент теперь уничтожен.
      call segxy       ; устанавливаем координаты сегмента.
      ld a,2           ; код атрибута 2 = сегмент красный.
      ld (23695),a     ; устанавливаем временные атрибуты.
      ld a,146         ; ASCII-код для символа 'C' UDG.
      rst 16
      ret
segmov ld a,(ix+1)      ; координата x.
      ld c,a           ; GP x area.
      ld a,(ix+2)      ; координата y.
      ld b,a           ; GP y area.
      ld a,(ix)        ; флаг статуса.
      and a            ; сегмент движется влево?
      jr z,segm1       ; да – переходим к нужному участку кода.

; сегмент движется вправо!

segmr  ld a,(ix+2)      ; координата y.
      cp 31           ; уже край экрана?
      jr z,segmd       ; да – двигаем сегмент вниз.
      inc a           ; проверяем знакоместо справа.
      ld b,a           ; set up GP y coord.
      call atadd       ; находим адрес атрибута.
      cp 68           ; грибы ярко (64) + зеленые (4).
      jr z,segmd       ; гриб справа, двигаемся вниз.
      inc (ix+2)       ; нет препятствий, идем вправо.
      ret

; сегмент движется вправо!

segm1  ld a,(ix+2)      ; координата y.
      and a           ; уже левый край экрана?
      jr z,segmd       ; да – двигаем сегмент вниз.
      dec a           ; смотрим вправо.
      ld b,a           ; set up GP y coord.
      call atadd       ; получаем адрес атрибутов в позиции (dispx,dispy).
      cp 68           ; грибы ярко (64) + зеленые (4).
      jr z,segmd       ; гриб слева, двигаемся вниз.
      dec (ix+2)       ; нет препятствий, идем влево.
      ret

; сегмент движется вниз!

segmd  ld a,(ix)        ; направление движения сегмента.
      xor 1           ; меняем на обратное.
      ld (ix),a        ; и помещаем по адресу (ix).
      ld a,(ix+1)      ; координата y.
      cp 21           ; уже низ экрана?
      jr z,segmt       ; да – перемещаем на самый верх.

; На данный момент мы продолжаем движение вниз, игнорируя возможные ;препятствия
; в виде грибов. Все, что находится на пути сегмента будет ;уничтожено.

      inc (ix+1)       ; не достигли низа экрана, двигаем вниз.
      ret

```

; перемещаем сегмент на вершину экрана.

```
segmt xor a ; тоже, что и ld a,0 но экономит 1 байт.  
ld (ix+1),a ; новая координата x = вершина экрана.  
ret
```

; Простой генератор псевдослучайных чисел.  
; Проходит указателем по адресам ПЗУ (хранится в seed), возвращает  
; их содержимое.

```
random ld hl,(seed) ; Указатель  
ld a,h  
and 31 ; Используем только первые 8кб ПЗУ.  
ld h,a  
ld a,(hl) ; Получаем «случайное» число по этому адресу.  
inc hl ; Увеличиваем указатель.  
ld (seed),hl  
ret  
seed defw 0
```

; Вычисление адреса атрибутов символа с координатами (dispx, dispy).

```
atadd ld a,c ; координата по вертикали.  
rrca ; умножаем на 32.  
rrca ; Тройной сдвиг вправо с переносом  
rrca ; быстрее пяти сдвигов влево.  
ld e,a  
and 3  
add a,88 ; 88x256=адрес атрибутов.  
ld d,a  
ld a,e  
and 224  
ld e,a  
ld a,b ; позиция по горизонтатали.  
add a,e  
ld e,a ; de=адрес атрибутов.  
ld a,(de) ; возврат, адрес теперь в аккумуляторе.  
ret
```

```
plx defb 0 ; координата x игрока.  
ply defb 0 ; координата y игрока
```

; UDG-графика.

```
blocks defb 16,16,56,56,124,124,254,254 ; игрок.  
defb 24,126,255,255,60,60,60,60 ; гриб.  
defb 24,126,126,255,255,126,126,24 ; сегмент сороконожки.
```

; Таблица сегментов.  
; Структура: 3 байта для каждого, всего 10 сегментов.  
; байт 1: 255=сегмент отключен, 0=влево, 1=вправо.  
; байт 2 = x (горизонтальная) координата.  
; байт 3 = y (вертикальная) координата.

```
segmnt defb 0,0,0 ; сегмент 1.  
defb 0,0,0 ; сегмент 2.  
defb 0,0,0 ; сегмент 3.  
defb 0,0,0 ; сегмент 4.  
defb 0,0,0 ; сегмент 5.  
defb 0,0,0 ; сегмент 6.  
defb 0,0,0 ; сегмент 7.  
defb 0,0,0 ; сегмент 8.  
defb 0,0,0 ; сегмент 9.  
defb 0,0,0 ; сегмент 10.
```

## Глава 7 – Определение столкновений с пришельцами

### Проверка координат

Алгоритм проверки координат должен быть знаком большинству программистов. Тем не менее, для полноты картины мы рассмотрим его в этом руководстве. К тому же, это будет следующим этапом в развитии нашей игры.

Простейший случай проверки столкновения двух символов UDG, будет выглядеть примерно так:

```
ld a,(playx)      ; координата x игрока.
cp (ix+1)          ; сравниваем с координатой x пришельца.
ret nz             ; если не равны, то нет столкновения.
ld a,(playy)      ; координата y игрока.
cp (ix+2)          ; сравниваем с координатой y пришельца.
ret nz             ; если не равны, то нет столкновения.
jp collis          ; столкновение произошло.
```

Здесь все довольно просто. Однако большинство игр использует графические элементы произвольных размеров. Что если изображения пришельцев будут размером четыре знакоместа в ширину и два в высоту, а игрока – три на три знакоместа? Нам необходимо проверить столкнулись ли какие-либо части изображений, поэтому сравнивать придется некоторый диапазон координат. Если пришелец будет менее чем одним знакоместом над игроком, или менее двух под ним, то вертикальные координаты совпадают. Если он менее чем в трех знакоместах левее игрока, либо менее двух правее, то горизонтальные координаты так же совпадают, и имеет место столкновение.

Давайте напишем код. Начнем с загрузки вертикальной координаты игрока:

```
ld a,(playx)      ; координата x игрока.
```

Вычитаем вертикальную координату пришельца:

```
sub (ix+1)         ; вычитаем координату x пришельца.
```

Теперь вычитаем один из высоты изображения игрока в знакоместах, и прибавляем это значение.

```
add a,2            ; высота игрока 3 знакоместа, прибавляем 3 - 1 = 2.
```

Если пришелец находится в этом диапазоне, то результат будет меньше суммы высот игрока и пришельца, поэтому делаем проверку:

```
cp 5               ; сумма высот 3 + 2 = 5.
ret nc             ; вне диапазона по вертикали.
```

Аналогично, проверим горизонтальные координаты:

```
ld a,(playy)      ; координата y игрока.
sub (ix+2)         ; вычитаем координату y пришельца.
add a,2            ; ширина игрока 3 знакоместа, прибавляем 3 - 1 = 2.
cp 7               ; суммарная ширина 3 + 4 = 7.
ret nc             ; вне диапазона по горизонтали.
jp collis          ; есть столкновение.
```

Естественно, данный метод работает не только для графики, основанной на знакоместах. Он прекрасно подходит и для спрайтов, но об этом позднее. Пришло время закончить нашу игру, добавив к ней определение столкновений. Так как вся наша графика основана на символах UDG – нам не потребуется никаких сложных вычислений. Простейших проверок  $x=x$  и  $y=y$  будет достаточно.

```

numseg equ 8                ; количество сегментов сороконожки.

; Устанавливаем черный экран.

    ld a,71                  ; белые чернила (7) на черной бумаге (0),
                                ; яркость (64).
    ld (23693),a             ; устанавливаем наши цвета.
    xor a                    ; быстрый способ загрузить ноль в аккумулятор.
    call 8859                 ; устанавливаем постоянный цвет бордюра.

; Настраиваем графику.

    ld hl,blocks              ; Адрес где хранится пользовательская графика.
    ld (23675),hl            ; теперь переменная будет указывать на него.

; ОК, начнем игру.

    call 3503                 ; Процедура ПЗУ, очищаем экран, открываем канал 2.

    xor a                     ; обнуляем аккумулятор.
    ld (dead),a              ; сбрасываем флаг «игрок погиб».

; Инициализируем координаты.

    ld hl,21+15*256           ; загружаем в пару hl начальные координаты.
    ld (plx),hl              ; устанавливаем координаты игрока .
    ld hl,255+255*256         ; начальные координаты пули.
    ld (pbx),hl              ; устанавливаем координату пули.

    ld b,10                   ; количество инициализируемых сегментов.
    ld hl,segmnt              ; начальный адрес таблицы сегментов.
segint ld (hl),1               ; начинаем двигаться вправо.
        inc hl
    ld (hl),0                 ; начинаем с верхнего края экрана.
        inc hl
    ld (hl),b                 ; используем регистр В как координату у.
        inc hl
    djnz segint               ; повторяем для оставшихся сегментов.

    call basexy               ; устанавливаем координаты x,y игрока.
    call splayr               ; выводим изображение игрока.

; Теперь заполняем игровое поле грибами.

    ld a,68                   ; зеленые чернила (4) на черной бумаге (0),
                                ; яркость (64).
    ld (23695),a             ; устанавливаем наши временные цвета.
    ld b,50                   ; начнем с небольшого количества.
mushlp ld a,22                ; управляющий ASCII код для АТ.
        rst 16
    call random                ; получаем «случайное» число.
    and 15                     ; по вертикали от 0 до 15.
    rst 16
    call random                ; еще одно псевдослучайное число.
    and 31                     ; по горизонтали от 0 до 31.
    rst 16
    ld a,145                  ; Символ UDG 'В' – изображение гриба.
    rst 16                    ; выводим гриб на экран.
    djnz mushlp               ; повторяем, пока не выведем все грибы.

```

; Это основной цикл.

mloop equ \$

; Удаляем изображение игрока.

call basexy ; устанавливаем координаты x,y игрока.  
call wspace ; выводим пробел поверх изображения игрока.

; Теперь, когда удалили игрока – можно переместить его и вывести  
; его изображение по новым координатам.

ld bc,63486 ; группа клавиш 1-5/порт джойстика 2.  
in a,(c) ; смотрим какие клавиши нажаты.  
rra ; крайний бит = клавиша 1.  
push af ; запоминаем значение.  
call nc,mp1 ; была нажата, движемся влево.  
pop af ; восстанавливаем аккумулятор.  
rra ; следующий бит (значение 2) = клавиша 2.  
push af ; запоминаем значение.  
call nc,mpr ; была нажата, движемся вправо.  
pop af ; восстанавливаем аккумулятор.  
rra ; следующий бит (значение 4) = клавиша 3.  
push af ; запоминаем значение.  
call nc,mpd ; была нажата, движемся вниз.  
pop af ; восстанавливаем аккумулятор.  
rra ; следующий бит (значение 8) считываем клавишу 4.  
push af ; запоминаем значение.  
call nc,mpu ; была нажата, движемся вверх.  
pop af ; восстанавливаем аккумулятор.  
rra ; последний бит (значение 16) считываем клавишу 5.  
call nc,fire ; была нажата, огонь.

; После передвижения, можно снова вывести изображение на экран.

call basexy ; устанавливаем координаты x,y игрока.  
call splayr ; выводим изображение.

; Теперь пуля. Сначала проверим попала ли она куда-нибудь.

call bchk ; проверяем позицию пули.  
call dbull ; удаляем пули.  
call moveb ; движем пули.  
call bchk ; проверяем новую позицию пули.  
call pbull ; выводим пули по новым координатам.

; Теперь сегменты сороконожки.

ld ix,segmnt ; таблица данных сегментов.  
ld b,10 ; количество сегментов в таблице.  
censeg push bc  
ld a,(ix) ; сегмент включен?  
inc a ; выключен это 255, inc 255=0.  
call nz,proseg ; включен (в а не ноль) – обрабатываем сегмент.  
pop bc  
ld de,3 ; 3 байта на сегмент.  
add ix,de ; в ix адрес следующего сегмента.  
djnz censeg ; повторить для всех сегментов.  
  
halt ; задержка.  
  
ld a,(dead) ; был ли игрок убит сегментом?  
and a  
ret nz ; игрок убит – теряет жизнь.

; Переход на начало основного цикла.

jp mloop

; Двигаем игрока влево.

mp1 ld hl,ply ; помните, у это координата по горизонтали!  
ld a,(hl) ; каково текущее значение?

```

and a          ; это ноль?
ret z          ; если да – мы больше не можем двигаться влево.

```

;Проверяем, нет ли на пути гриба.

```

ld bc,(plx)    ; текущие координаты.
dec b          ; проверяем знакоместо слева.
call atadd     ; получаем адрес атрибутов в этой позиции.
cp 68          ; грибы яркие (64) + зеленые (4).
ret z          ; если гриб есть – не можем туда двигаться.

dec (hl)       ; вычитаем 1 из координаты y.
ret

```

; Двигаем игрока вправо.

```

mpr    ld hl,ply    ; помните, y это координата по горизонтали!
        ld a,(hl)    ; каково текущее значение?
        cp 31        ; это правый край экрана (31)?
        ret z        ; если да – мы больше не можем двигаться вправо.

```

; Проверяем, нет ли на пути гриба..

```

ld bc,(plx)    ; текущие координаты.
inc b          ; проверяем знакоместо справа.
call atadd     ; получаем адрес атрибутов в этой позиции.
cp 68          ; грибы яркие (64) + зеленые (4).
ret z          ; если гриб есть – не можем туда двигаться.

inc (hl)       ; прибавляем 1 к координате y.
ret

```

; Двигаем игрока вверх.

```

mru    ld hl,plx    ; помните, x это координата по вертикали!
        ld a,(hl)    ; каково текущее значение?
        cp 4         ; это верхний край экрана (4)?
        ret z        ; если да – нельзя двигаться выше.

```

; Проверяем, нет ли на пути гриба.

```

ld bc,(plx)    ; текущие координаты.
dec c          ; проверяем знакоместо сверху.
call atadd     ; получаем адрес атрибутов в этой позиции.
cp 68          ; грибы яркие (64) + зеленые (4).
ret z          ; если гриб есть – не можем туда двигаться.

dec (hl)       ; вычитаем 1 из координаты x.
ret

```

; Двигаем игрока вниз.

```

mpd    ld hl,plx    ; помните, x это координата по вертикали!
        ld a,(hl)    ; каково текущее значение?
        cp 21        ; это нижний край экрана (21)?
        ret z        ; если да – нельзя больше двигаться вниз.

```

; Проверяем, нет ли на пути гриба.

```

ld bc,(plx)    ; текущие координаты.
inc c          ; проверяем знакоместо снизу.
call atadd     ; получаем адрес атрибутов в этой позиции.
cp 68          ; грибы яркие (64) + зеленые (4).
ret z          ; если гриб есть – не можем туда двигаться.

inc (hl)       ; прибавляем 1 к координате x.
ret

```

; Стреляем.

```

fire    ld a,(pbx)    ; вертикальная координата пули.
        inc a          ; 255 начальное значение, инкрементируем до нуля.
        ret nz        ; пуля на экране, не можем выстрелить снова.
        ld hl,(plx)    ; координата игрока.

```



```

dec 1 ; на одно знакоместо выше.
ld (pbx),h1 ; устанавливаем координату пули.
ret

bchk ld a,(pbx) ; вертикальная координата пули.
inc a ; равна 255 (начальное значение)?
ret z ; если равна, пули нет на экране.
ld bc,(pbx) ; загружаем координаты.
call atadd ; получаем адрес атрибутов в этой позиции.
cp 68 ; грибы яркие (64) + зеленые (4).
jr z,hmush ; попала в гриб!
ret

hmush ld a,22 ; управляющий ASCII код для AT.
rst 16
ld a,(pbx) ; вертикальная координата пули.
rst 16
ld a,(pby) ; горизонтальная координата пули.
rst 16
call wspace ; устанавливаем белый цвет чернил.
kilbul ld a,255 ; если координата x пули 255 = пуля выключена.
ld (pbx),a ; уничтожаем пулю.
ret

```

; Двигаем пулю вверх по экрану на одно знакоместо вверх.

```

moveb ld a,(pbx) ; вертикальная координата пули.
inc a ; равна (начальное значение)?
ret z ; если равна, пули нет на экране.
sub 2 ; вверх на один ряд.
ld (pbx),a
ret

```

; Устанавливает координаты x,y игрока на экране, эту процедуру необходимо  
; вызывать перед выводом изображения игрока, или его удалением.

```

basexy ld a,22 ; управляющий ASCII-код AT.
rst 16
ld a,(plx) ; координата игрока по вертикали.
rst 16 ; устанавливаем координату игрока по вертикали.
ld a,(ply) ; координата игрока по горизонтали.
rst 16 ; устанавливаем позицию по горизонтали.
ret

```

; Вывод изображения игрока по текущим координатам.

```

splayr ld a,69 ; голубые чернила (5) на черной бумаге (0),
; яркость (64).
ld (23695),a ; устанавливаем временные цвета.
ld a,144 ; ASCII-код для символа 'A' UDГ.
rst 16 ; выводим изображение игрока.
ret

```

```

pbull ld a,(pbx) ; вертикальная координата пули.
inc a ; равна 255 (начальное значение)?
ret z ; если равна, пули нет на экране.
call bullxy
ld a,16 ; управляющий ASCII-код INK.
rst 16
ld a,6 ; 6 = желтый.
rst 16
ld a,147 ; Символ UDГ 'D' - пули игрока.
rst 16
ret

```

```

dbull ld a,(pbx) ; вертикальная координата пули.
inc a ; равна 255 (начальное значение)?
ret z ; если равна, пули нет на экране.
call bullxy ; устанавливаем координаты пули.

```

```

wspace ld a,71 ; белые чернила (7) на черной бумаге (0),
; яркость (64).
ld (23695),a ; устанавливаем наши временные цвета.
ld a,32 ; символ пробела.
rst 16 ; выводим пробел.

```

ret

; Устанавливаем координаты x, y для пули игрока.  
; эта процедура вызывается перед выводом и удалением пуль.

bullyx ld a,22 ; управляющий ASCII-код АТ.  
rst 16  
ld a,(pbx) ; вертикальная координата пули.  
rst 16 ; устанавливаем вертикальную позицию игрока.  
ld a,(pby) ; горизонтальная координата пули.  
rst 16 ; устанавливаем горизонтальную координату.  
ret

segxy ld a,22 ; Управляющей ASCII-код АТ.  
rst 16 ; вывести код АТ.  
ld a,(ix+1) ; координата x сегмента.  
rst 16 ; вывести код координаты.  
ld a,(ix+2) ; координата y сегмента.  
rst 16 ; вывести код координаты.  
ret

proseg call segcol ; проверка столкновений сегмента.  
ld a,(ix) ; проверяем был ли сегмент выключен  
inc a ; процедурой проверки столкновений.  
ret z ; был, значит этот сегмент теперь уничтожен.  
call segxy ; устанавливаем координаты сегмента.  
call wspace ; выводим пробел, белые чернила на черном.  
call segmov ; перемещаем сегмент.  
call segcol ; проверка столкновений сегмента на новой позиции.

ld a,(ix) ; проверяем был ли сегмент выключен  
inc a ; процедурой проверки столкновений.  
ret z ; был, значит этот сегмент теперь уничтожен.  
call segxy ; устанавливаем координаты сегмента.  
ld a,2 ; код атрибута 2 = сегмент красный.  
ld (23695),a ; устанавливаем временные атрибуты.  
ld a,146 ; ASCII-код для символа 'C' UDG.  
rst 16  
ret

segmov ld a,(ix+1) ; координата x.  
ld c,a ; GP x area.  
ld a,(ix+2) ; координата y.  
ld b,a ; GP y area.  
ld a,(ix) ; флаг статуса.  
and a ; сегмент движется влево?  
jr z,segml ; да – переходим к нужному участку кода.

; сегмент движется вправо!

segmr ld a,(ix+2) ; координата y.  
cp 31 ; уже край экрана?  
jr z,segmd ; да – двигаем сегмент вниз.  
inc a ; проверяем знакоместо справа.  
ld b,a ; set up GP y coord.  
call atadd ; находим адрес атрибута.  
cp 68 ; грибы ярко (64) + зеленые (4).  
jr z,segmd ; гриб справа, двигаемся вниз.  
inc (ix+2) ; нет препятствий, идем вправо.  
ret

; сегмент движется вправо!

segml ld a,(ix+2) ; координата y.  
and a ; уже левый край экрана?  
jr z,segmd ; да – двигаем сегмент вниз.  
dec a ; смотрим вправо.  
ld b,a ; set up GP y coord.  
call atadd ; получаем адрес атрибутов в позиции (dispx,dispy).  
cp 68 ; грибы ярко (64) + зеленые (4).  
jr z,segmd ; гриб слева, двигаемся вниз.  
dec (ix+2) ; нет препятствий, идем влево.  
ret

; сегмент движется вниз!

```

segmd  ld a,(ix)          ; направление движения сегмента.
        xor 1             ; меняем на обратное.
        ld (ix),a         ; и помещаем по адресу (ix).
        ld a,(ix+1)       ; координата y.
        cp 21             ; уже низ экрана?
        jr z,segmt        ; да – перемещаем на самый верх.

```

; На данный момент мы продолжаем движение вниз, игнорируя возможные ;препятствия в виде грибов. Все, что находится на пути сегмента будет ;уничтожено.

```

        inc (ix+1)        ; не достигли низа экрана, двигаем вниз.
        ret

```

; перемещаем сегмент на вершину экрана.

```

segmt  xor a              ; тоже, что и ld a,0 но экономит 1 байт.
        ld (ix+1),a       ; новая координата x = вершина экрана.
        ret

```

; Проверка столкновений сегмента.

; Проверяем столкновения с игроком и пулями.

```

segcol ld a,(ply)         ; координата y игрока.
        cp (ix+2)         ; она равна координате y сегмента?
        jr nz,bulcol      ; координаты y отличаются, переходим к пуле.
        ld a,(plx)        ; координата x игрока.
        cp (ix+1)         ; равна координате сегмента?
        jr nz,bulcol      ; координаты x отличаются, переходим к пуле.

```

; Если произошло столкновение с игроком.

```

killpl ld (dead),a        ; устанавливаем флаг, что игрок погиб.
        ret

```

; Проверим столкновение с пулей игрока.

```

bulcol ld a,(pbx)         ; координата x пули.
        inc a             ; текущее значение?
        ret z             ; если да, то не нужно проверять.
        cp (ix+1)         ; координаты x пули и сегмента равны?
        ret nz            ; если нет, то столкновение отсутствует.
        ld a,(pby)        ; координата y пули.
        cp (ix+2)         ; равны ли координаты y пули и сегмента?
        ret nz            ; нет – не столкнулись.

```

; Если произошло столкновение с пулей игрока.

```

        call dbul1        ; удаляем пулю.
        ld a,22           ; Управляющей ASCII-код AT.
        rst 16
        ld a,(pbx)        ; вертикальная координата пули игрока.
        inc a             ; на одну линию вниз.
        rst 16            ; устанавливаем вертикальную координату гриба.
        ld a,(pby)        ; горизонтальная координата пули игрока.
        rst 16            ; устанавливаем горизонтальную координату.
        ld a,16           ; Управляющей ASCII-код INK.
        rst 16
        ld a,4            ; 4 = зеленый цвет.
        rst 16            ; все грибы у нас этого цвета!
        ld a,145          ; символ UDG 'B' это изображение гриба.
        rst 16            ; выводим гриб на экран.
        call kilbul       ; уничтожаем пулю.
        ld (ix),a         ; уничтожаем сегмент.
        ld hl,numseg      ; количество сегментов.
        dec (hl)          ; уменьшаем его.
        ret

```

; Простой генератор псевдослучайных чисел.

; Проходит указателем по адресам ПЗУ (хранится в seed), возвращает ; их содержимое.

```

random ld hl,(seed)       ; указатель
        ld a,h
        and 31            ; используем только первые 8кб ПЗУ.

```

```

        ld h,a
        ld a,(h1)          ; Получаем «случайное» число по этому адресу.
        inc h1             ; Увеличиваем указатель.
        ld (seed),h1
        ret
seed    defw 0

```

; Вычисление адреса атрибутов символа с координатами (dispx, dispy).

```

atadd   ld a,c              ; координата по вертикали.
        rrca                ; умножаем на 32.
        rrca                ; Тройной сдвиг вправо с переносом
        rrca                ; быстрее пяти сдвигов влево.
        ld e,a
        and 3
        add a,88            ; 88x256=адрес атрибутов.
        ld d,a
        ld a,e
        and 224
        ld e,a
        ld a,b              ; позиция по горизонтатали.
        add a,e
        ld e,a              ; de=адрес атрибутов.
        ld a,(de)           ; возврат, адрес теперь в аккумуляторе.
        ret

plx     defb 0              ; координата x игрока.
ply     defb 0              ; координата y игрока
pbx     defb 255            ; координаты пули игрока.
pby     defb 255
dead    defb 0              ; флаг – игрок погиб, если не ноль.

```

; UDГ-графика.

```

blocks  defb 16,16,56,56,124,124,254,254 ; игрок.
        defb 24,126,255,255,60,60,60,60  ; гриб.
        defb 24,126,126,255,255,126,126,24 ; сегмент сороконожки.
        defb 0,102,102,102,102,102,102,0 ; пуля.

```

; Таблица сегментов.

; Структура: 3 байта для каждого, всего 10 сегментов.

; байт 1: 255=сегмент отключен, 0=влево, 1=вправо.

; байт 2 = x (горизонтальная) координата.

; байт 3 = y (вертикальная) координата.

```

segmnt  defb 0,0,0          ; сегмент 1.
        defb 0,0,0          ; сегмент 2.
        defb 0,0,0          ; сегмент 3.
        defb 0,0,0          ; сегмент 4.
        defb 0,0,0          ; сегмент 5.
        defb 0,0,0          ; сегмент 6.
        defb 0,0,0          ; сегмент 7.
        defb 0,0,0          ; сегмент 8.
        defb 0,0,0          ; сегмент 9.
        defb 0,0,0          ; сегмент 10.

```

Подождите, но почему здесь две проверки вместо одной? Давайте разберемся. Представьте, что корабль игрока и сегмент сороконожки находятся на соседних знакоместах. Игрок слева, а сегмент – справа. Игрок движется вправо, а сегмент – влево. В следующем кадре сегмент займет место корабля, а корабль – место сегмента. Игрок и сороконожка пройдут сквозь друг друга, и однократная проверка этого не определит. Во избежание этой проблемы мы делаем дополнительные проверки после перемещения игрока и сороконожки.

## Столкновения спрайтов

Откровенно говоря, в большинстве игр для Спектрума используются спрайты, а не символы UDG. Поэтому в следующей главе мы рассмотрим методы вывода их на экран. Для определения столкновений между спрайтами используется тот же самый принцип сравнения координат. Вычитаем координаты первого спрайта из координат второго, прибавляем размер спрайта и сравниваем результат. Если он в диапазоне суммарного размера двух спрайтов, значит есть столкновение по этой оси.

Простейшая проверка столкновения спрайтов размером 16x16 пикселей может выглядеть примерно так:

; проверяем (l, h) на столкновение с (c, b), точное совпадение.

```
colx16 ld a,l          ; координата x.
      sub c            ; вычитаем x.
      add a,15         ; прибавляем максимальное расстояние.
      cp 31            ; в диапазоне x?
      ret nc           ; нет – не столкнулись.
      ld a,h          ; координата y.
      sub b            ; вычитаем y.
      add a,15         ; прибавляем максимальное расстояние.
      cp 31            ; в диапазоне y?
      ret              ; устанавливаем флаг переноса при столкновении.
```

У этого способа есть недостаток. Если спрайты полностью не заполняют пространство квадрата 16x16 пикселей, то данный метод вас может не устроить. Спрайты будут сталкиваться, визуально находясь очень близко друг к другу, но не соприкасаясь. Менее чувствительной проверку можно сделать, если «отсечь» углы спрайтов до формы восьмиугольника. Рассмотрим работу процедуры, которая это реализует. При столкновении спрайтов 16x16 максимальная разница их координат – 15 пикселей для каждой оси.

Складывая разницу значений координат по x и по y и сравнивая это значение с определенным числом (в нашем случае это 25), мы фактически «отсекаем» углы спрайтов (треугольники 5x5x5 пикселей), и они не учитываются при определении столкновений.

; проверяем (l, h) на столкновение с (c, b), с «отсечением углов».

```
colc16 ld a,l          ; координата x.
      sub c            ; вычитаем x.
      jr nc,colc1a     ; если результат положительное число – переход.
      neg              ; делаем отрицательное положительным.
colc1a cp 16            ; в пределах столкновения по x?
      ret nc           ; нет – не столкнулись.
      ld e,a           ; запоминаем разницу.

      ld a,h          ; координата y.
      sub b            ; вычитаем y.
      jr nc,colc1b     ; если результат положительное число – переход.
      neg              ; делаем отрицательное положительным.
colc1b cp 16            ; в пределах столкновения по y?
      ret nc           ; нет – не столкнулись.

      add a,e          ; прибавляем разницу по x.
      cp 26            ; соприкасаются только углы 5x5x5 пикселей?
      ret              ; устанавливаем флаг переноса в случае столкновения.
```