**Лабораторная работа 8. Создание сложных программ на ассемблере для микроконтроллеров AVR**

**Задание на лабораторную работу №8**

Создать автомат «Бегущие огни» на основе процессора AVR Tiny1323 в соответствии с перечисленными ниже вариантами. Номер варианта должен соответствовать номеру в списке группы.

Схема бегущих огней со светодиодами приведена на рисунке 1.

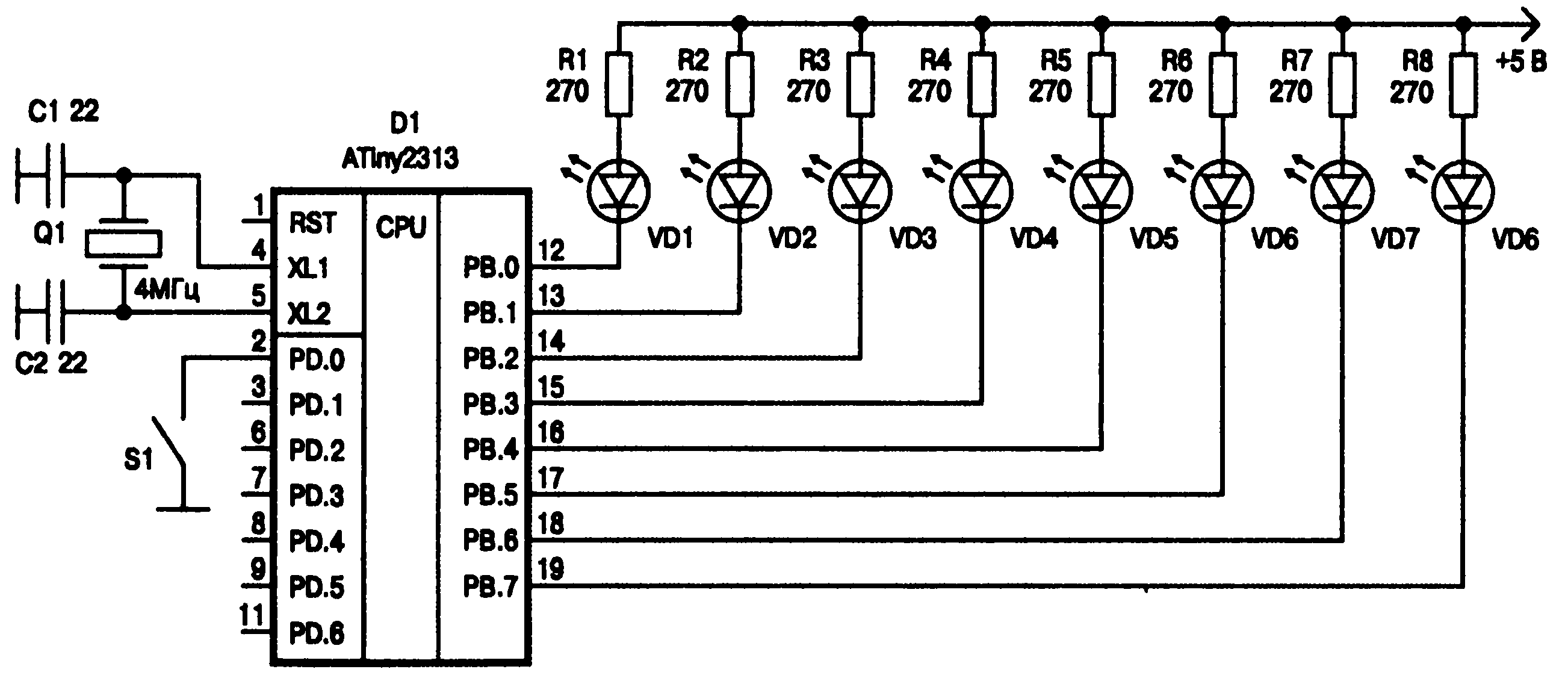


Рисунок 1

Для создания эффекта «бегущих огней» удобнее всего воспользоваться операторами сдвига, которые имеются в системе команд любого микроконтроллера. Такие операторы сдвигают содержимое одного из регистров микроконтроллера на один разряд влево или вправо. Если сдвигать содержимое регистра и после каждого сдвига выводить новое содержимое в порт РВ, подключенные к нему светодиоды будут загораться поочередно, имитируя бегущий огонь. Алгоритм работы бегущих огней может быть разный. Один из возможных алгоритмов в самых общих чертах будет выглядеть следующим образом:

1. Считать состояние переключателя управления;

2. Если контакты переключателя разомкнуты, перейти к проце-дуре сдвига вправо;

3. Если контакты замкнуты, перейти к процедуре сдвига влево;

4. После окончания полного цикла сдвига (восемь последовательных сдвигов) перейти к началу алгоритма, то есть к пункту 1.

Таким образом, все время, пока контакты переключателя разомкнуты, программа будет выполнять сдвиг вправо. Если состояние переключателя не изменилось, сдвиг в прежнем направлении продолжается. Если замкнуть контакты переключателя, то все время, пока они замкнуты, будет выполняться сдвиг влево. Как при сдвиге вправо, так и при сдвиге влево после каждого полного цикла сдвига (8 шагов) происходит проверка переключателя. Если его состояние не такое же, как и прежде, то направление сдвига не изменяется. В противном случае программа меняет направление сдвига.

**Выполнение алгоритма сдвига**

Посмотрим теперь, как выполняется сам алгоритм сдвига. Сдвиг влево и сдвиг вправо выполняются аналогично. Ниже приводится обобщенный алгоритм для сдвига влево и сдвига вправо, снабженный комментариями.

1. Записать в рабочий регистр начальное значение. В качестве начального значения используется двоичное число, у которого один из разрядов равен единице, а остальные разряды равны нулю. Для сдвига вправо нам нужно число с единицей в самом старшем разряде (0b10000000). Для сдвига влево в единицу устанавливается младший разряд (0b0000000l).

2. Вывести значение рабочего регистра в порт РВ.

3. Вызвать подпрограмму задержки. Задержка нужна для того, чтобы скорость «бега» огней была нормальная для глаз наблюдателя. Если бы не было задержки, то скорость «бега» была бы столь велика, что мы бы и не увидели движения огней.

4. Сдвинуть содержимое рабочего регистра вправо (влево) на один разряд.

5. Проверить, не окончился ли полный цикл сдвига (8 шагов).

6. Если полный цикл сдвига не закончен, перейти к пункту 2 данного алгоритма. Это приведет к тому, что пункты 2,3,4,5 и 6 повторятся 8 раз, и лишь затем завершится полный цикл сдвига.

**Программа на Ассемблере**

Возможный вариант программы приведен на листинге 1. В программе встречается несколько новых операторов. Кроме того, мы будем иметь дело с новым для нас флагом. Этот флаг также является одним из разрядов регистра SREG и называется флагом переноса.

Флаг переноса — это разряд, куда помещается бит переноса при выполнении операций сложения двух чисел или бит заема при операциях вычитания.

Листинг 1

/\*

\* Beg\_Ogni.asm

\*

\* Created: 17.11.2014 10:41:27

\* Author: student

\*/

.include "tn2313def.inc" ; Присоединение файла описаний

.list ;Включение листинга

.def temp = R16 ;Определение главного рабочего регистра

.def loop1 = R17 ;Определение первого регистра организации цикла

.def loop2 = R18 ;Определение второго регистра организации цикла

.def loop3 = R19 ;Определение третьего регистра организации цикла

.def rab = R20 ; Определение рабочего регистра для команд сдвига

;Начало программного кода

.cseg ;Выбор сегмента программного кода

.org 0 ;Установка текущего адреса на ноль

;Инициализация стека

ldi temp, RAMEND ;Выбор адреса вершины стека

out SPL, temp ;Запись его в регистр стека

;Инициализация портов ВВ

ldi temp, 0 ;Записываем ноль в регистр temp

out DDRD, temp ;Записываем этот ноль в DDRD (порт PD на ввод)

ldi temp, 0xFF ;Записываем число $FF в регистр temp

out DDRB, temp ;Записываем temp в DDRB (порт РВ на вывод)

out PORTB, temp ;Записываем temp в PORTB (потушить светодиод)

out PORTD, temp ;Записываем temp в PORTD (включаем внутр.резист.)

;Инициализация компаратора

ldi temp, 0x80 ;Выключение компаратора

out ACSR, temp

; Начало основного цикла

main: in temp, PIND ;Читаем содержимое порта PD

sbrc temp, 0 ;Проверка младшего разряда

rjmp m3 ;Если не ноль, переходим в начало

;Сдвиг вправо

m1: ldi rab, 0b10000000 ;Запись начального значения

m2: ldi temp, 0xff

eor temp, rab ;Инверсия содержимого регистра rab

out PORTB, temp ;Вывод текущего значения в порт РВ

rcall wt1 ;Задержка

lsr rab ;Сдвиг содержимого рабочего регистра

brcc m2 ;Если не дошло до конца регистра продолжить

rjmp main ;На начало

;Сдвиг влево

m3: ldi rab, 0b10000001 ;Запись начального значения

m4: ldi temp, 0xff

eor temp, rab ;Инверсия содержимого регистра rab

out PORTB, temp ;Вывод текущего значения в порт РВ

rcall wt1 ;Задержка

lsl rab ;Сдвиг содержимого рабочего регистра

brcc m4 ;Если не дошло до конца регистра продолжить

rjmp main ;На начало

;Подпрограмма задержки

push loop1 ;Сохраняем содержимое регистра loopl

push loop2 ;Сохраняем содержимое регистра lоор2

push loop3 ;Сохраняем содержимое регистра lоорЗ

ldi loop3, 15 ;Помещаем в 1оорЗ константу задержки

wt1: dec loop3 ;Уменьшаем значение регистра lоорЗ

breq wt4

ldi loop2, 100 ;Помещаем в lоор2 константу задержки

wt2: dec loop2 ;Уменьшаем значение регистра lоор2

breq wt1

ldi loop1, 255 ;Помещаем в loop1 константу задержки

wt3: dec loop1 ;Уменьшаем значение регистра loop1

brne wt3 ;Если не ноль, продолжаем цикл

rjmp wt2

wt4: pop loop3 ;Восстанавливаем значение регистра lоор3

pop loop2 ;Восстанавливаем значение регистра lоор2

pop loop1 ;Восстанавливаем значение регистра loop1

ret ;Выход из подпрограммы

Содержимое флага переноса так же, как и содержимое флага нулевого результата Z, может служить условием для оператора условного перехода. Кроме своего основного предназначения, флаг переноса иногда выполняет и вспомогательные функции. Например, он участвует во всех операциях сдвига в качестве дополнительного разряда. Теперь рассмотрим подробнее все новые операторы.

***rcall***

Переход к подпрограмме. У этого оператора всего один параметр — относительный адрес начала подпрограммы. Относительный адрес — это просто смещение относительно текущего адреса. Выполняя команду rcall 1, микроконтроллер запоминает в стеке текущий адрес программы из счетчика команд и переходит по адресу, определяемому смещением. Такой же принцип задания адреса для перехода мы уже встречали в команде rjmp. В программе (листинг 1) производится вызов подпрограммы задержки по адресу, соответствующему метке wt1.

***ret***

Команда выхода из подпрограммы. По этой команде микроконтроллер извлекает из стека адрес, записанный туда при выполнении команды rcall, и осуществляет передачу управления по этому адресу. В листинге 1 команду ret мы можем видеть в конце подпрограммы wt.

***push***

Запись содержимого регистра общего назначения в стек. У данного оператора всего один операнд — имя регистра, содержимое которого нужно поместить в стек. В программе (листинг 1) в стек помещается содержимое регистра с именем loop.

***pop***

Извлечение информации из стека. У этого оператора тоже всего один операнд — имя регистра, в который помещается информация, извлекаемая из стека. В программе (листинг 1) информация извлекается из стека и помещается в регистр loop.

***dec***

Уменьшение содержимого РОН. У команды один параметр — имя регистра. Команда dec (декремент) уменьшает на единицу содержимое регистра, имя которого является ее параметром. В программе (листинг 1) уменьшается на единицу содержимое регистра loop.

***brne***

Оператор условного перехода (переход по условию). У этого оператора всего один параметр — относительный адрес перехода. Условие перехода звучит как «не равно». Попробуем разобраться, как проверяется это условие. И тут нам придется вспомнить о регистре состояния микроконтроллера (SREG).

Каждый бит этого регистра представляет собой флаг. Все флаги регистра предназначены для управления работой микроконтроллера. Кроме уже известного нам флага I (глобальное разрешение прерываний), этот регистр имеет ряд флагов, отражающих результаты работы различных операций.

В данном случае нас интересует лишь один из таких флагов — флаг нулевого результата (флаг Z). Этот флаг устанавливается в том случае, когда при выполнения очередной команды результат окажется равным нулю. Например, при вычитании двух чисел, сдвиге разрядов числа или в результате операции сравнения. В нашем случае на значение флага влияет команда dec.

Если в результате действия этого оператора содержимое регистра окажется равным нулю, то флаг Z устанавливается в единицу. В противном случае он сбрасывается в ноль.

Флаг Z будет хранить записанное в него значение до тех пор, пока какая-нибудь другая команда его не изменит. Какие из команд оказывают влияние на флаг Z, а какие нет, можно узнать из описания данного регистра.

Команда brne использует флаг Z в качестве условия. Команда выполняет переход только в том случае, если флаг Z сброшен. То есть когда результат предыдущей команды неравен нулю.

***breq***

Оператор условного перехода по условию «равно». Этот оператор — полная противоположность оператору brne. Отличие этих двух операторов друг от друга в том, что brne вызывает переход в том случае, если флаг Z установлен, а оператор breq, напротив, вызовет переход, если Z сброшен.

***lsr***

Логический сдвиг вправо. Этот оператор имеет всего один параметр — имя регистра, содержимое которого сдвигается. Схематически данная операция выглядит следующим образом:

0 🡪d7 🡪 d6 🡪 d5 🡪 d4 🡪 d3 🡪 d2 🡪 d1 🡪 d0 🡪 C.

Содержимое младшего разряда переносится в флаг переноса С, на его место поступает содержимое разряда 1, в разряд 1 попадает содержимое разряда 2, и так далее. В самый старший разряд записывается ноль.

***lsl***

Логический сдвиг влево. Действие этого оператора обратно действию предыдущего. Схема такого сдвига выглядит следующим образом:

C 🡨 d7 🡨 d6 🡨 d5 🡨 d4 🡨 d3 🡨 d2 🡨 d1 🡨 d0 🡨 0.

В данном случае в С попадает содержимое старшего разряда. Содержимое всех остальных разрядов сдвигается на один шаг влево. В самый младший разряд записывается ноль.

***brcc***

Переход по условию «нет переноса». Данный оператор проверяет содержимое флага переноса С и осуществляет переход по относительному адресу в том случае, если флаг С не установлен (равен нулю).

***eor***

Оператор «Исключающее ИЛИ». Этот оператор имеет два параметра. В качестве параметров выступают имена двух регистров, с содержимым которых производится данная операция. Оператор производит поразрядную операцию «Исключающее ИЛИ» между содержимым обоих регистров. Результат помещается в тот регистр, имя которого указано в качестве первого параметра.

**Описание программы**

Так как циклы нужны в разных местах программы, они оформлены в виде подпрограммы. Для их формирования используются дополнительные регистры общего назначения.

Регистрам присваивается имя loop. По-английски слово loop означает цикл. Таким именем принято называть переменные, применяемые для организации циклов.

Запись значения в этот регистр эквивалентна присвоению значения переменной. Также можно интерпретировать и другие операции с регистром. Сложение содержимого двух регистров можно считать сложением переменных, вычитание — вычитанием, и так далее.

Первой строке подпрограмм циклов присвоена метка wt. Именно по этой метке и будет вызываться подпрограмма. Сначала в регистр loop записывается некое начальное значение. Затем начинается цикл, который постепенно уменьшает значение регистра loop до нуля. Когда же содержимое loop кажется равным нулю, очередного перехода не произойдет, и цикл закончится.

Команды работы со стеком push и pop предназначены для сохранения в стеке и последующего восстановления содержимого регистра loop. В начале подпрограммы значение loop сохраняется, а перед выходом их подпрограммы — восстанавливается.

Подобный прием придает программе одно полезное свойство. После окончания работы подпрограммы значения всех регистров микроконтроллера остается без изменений. В данном конкретном случае такое свойство ничего не дает, кроме, разве что, дополнительной задержки. Однако в сложных программах, имеющих не одну, а несколько подпрограмм, одни и те же регистры удобно использовать в разных подпрограммах.

Те же самые регистры может использовать и основная программа. В этом случае описанное выше полезное свойство просто необходимо для правильной работы всей программы. Зная эту особенность, программисты стараются применять подобный прием в каждой подпрограмме, независимо от того, полезен он в данном конкретном случае или нет.

Не исключена ситуация, когда в процессе доработки программы вам все же придется повторно использовать какие-либо регистры. Заранее обеспечить корректную работу вашей подпрограммы — это хороший стиль программирования.

Модуль инициализации новой программы остался таким же, как и в предыдущей лабораторной работе. Дополнен лишь блок описания переменных.

Кроме регистров loop1, 1оор2 и 1оорЗ, нам понадобится еще один дополнительный регистр. Этот регистр мы будем использовать как рабочий в операциях сдвига. В качестве такого регистра выбран регистр r20, которому присваивается имя rab.

Основной цикл программы начинается с чтения содержимого порта PD. Результат помещается в регистр temp. Далее происходит оценка младшего разряда прочитанного числа. Если этот разряд равен единице, то оператор безусловного перехода пропускается, и программа переходит к выполнению процедуры сдвига вправо. Если младший разряд считанного из PD числа равен нулю, то оператор rjmp передает управление по метке m3, и программа переходит к процедуре сдвига влево.

Процедура «сдвиг вправо» работает следующим образом. Рабочему регистру rab присваивается начальное значение. Для наглядности это число записано в двоичном формате. Затем начинается цикл сдвига. Первой операцией цикла сдвига, в соответствии с алгоритмом, должна быть операция вывода содержимого регистра rab в порт РВ. Однако существует одно небольшое препятствие.

Если просто вывести содержимое rab в порт РВ, то мы получим картину, обратную той, которая нам необходима. Все светодиоды, кроме одного, будут светиться. Это произойдет потому, что ноль на выходе порта зажигает светодиод, а единица тушит. Если мы хотим получить бегущий огонь, а не бегущую тень, нам нужно проинвертировать содержимое регистра rab перед тем, как вывести в порт РВ.

Для инвертирования содержимого регистра rab воспользуемся командой eor («Исключающее ИЛИ»). Операция «Исключающее ИЛИ» обладает способностью инвертирования чисел. Если один из операндов равен единице, то результат операции равен инверсному значению второго операнда.

Поэтому, если произвести операцию «Исключающее ИЛИ» между двумя байтами, значение одного из которых будет равно 0xFF, то в результате этой операции мы получим инверсное значение второго байта. Для выполнения такой операции используется вспомогательный регистр temp. В регистр temp записывается число 0xFF, а затем производится операция «Исключающее ИЛИ» между содержимым регистров temp и rab.

Результат этой операции помещается в temp, так как именно он является первым параметром данной команды. Содержимое регистра rab при этом не изменяется. Затем содержимое регистра temp выводится в порт РВ.

Следующий этап процедуры сдвига — вызов подпрограммы задержки. Вызов этой подпрограммы происходит по метке wt1. Далее производится сдвиг содержимого регистра rab на один бит вправо. Оператор brсc проверяет состояние признака переноса. Эта проверка позволяет обнаружить момент, когда закончится полный цикл сдвига. Как это происходит, иллюстрирует таблица 1.

Таблица 1 - Сдвиг информации в рабочем регистре

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | B7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | C |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

В таблице показаны значения всех разрядов вспомогательного регистра rab для каждого из восьми шагов, составляющих полный цикл сдвига. Разряды сдвигаемого регистра обозначены как b7-b0. Последний столбец показывает содержимое признака переноса. Как видно из таблицы, единица, которая вначале находится в самом старшем разряде, при каждом очередном шаге сдвигается в соседнюю позицию.

В результате, после восьмого шага она оказывается в ячейке признака переноса. Пока С равно нулю, оператор brcc передает управление по метке m2, и цикл сдвига продолжается. После восьмого шага признак переноса С станет равен единице. Поэтому перехода на начало цикла не произойдет, и управление перейдет к следующей строке. В результате очередного девятого цикла сдвига не произойдет. Оператор безусловного перехода передаст управление на начало основного цикла, и программа снова приступит к проверке состояния кнопки.

Процедура сдвига влево работает точно так же, как и процедура сдвига вправо.

Отличия:

- начальное значение, записываемое в регистр rab, равно 0b00000001;

- вместо оператора lsr (сдвиг вправо) использован оператор lsl (сдвиг влево).

В качестве подпрограммы задержки применена подпрограмма с тремя вложенными циклами.

**Индивидуальное задание:**

Используя пособие **Белов А.В. Разработка устройств на микроконтроллерах AVR: шагаем от «чайника» до профи. — СПб.: Наука и Техника, 2013. с.161-170.** студенты создают программу «Бегущие огни» на ассемблере и СИ для МК AVR.

Техническое задание:

Разработать автомат «Бегущие огни» для управления несколькими светодиодами. Устройство должно обеспечивать движение включения светодиодов в двух разных направлениях. Переключение направления «движения» должно осуществляться при помощи двух кнопок (нормальное положение – «разомкнуто»), подключенных к портам ввода/вывода.

В отличие от программ по созданию эффекта бегущих огней, рассмотренных в предлагаемом выше для изучения пособии, следует, во-первых, использовать две раздельные кнопки, подключенные к портам ввода/вывода. Во-вторых, гирлянды могут быть подключены, как к портам PD, так и к портам PB в соответствии с предложенным вариантом. Таким образом, в соответствии с поставленной задачей устройство должно управлять семью или восемью светодиодами, в зависимости от порта, к которому они подключены.

При этом, студенты с нечетным номером по списку группы создают программу «Бегущие огни» на языке программирования СИ, а студенты с четными номерами - такую же программу, но написанную на ассемблере для МК AVR.

|  |  |  |  |
| --- | --- | --- | --- |
| №  варианта | Порт подключения кнопки «бежать вверх» (от младшего к старшему биту) | Порт подключения кнопки «бежать вниз» (от старшего к младшему биту) | Порты  подключения  светодиодов |
| 1 | PB0 | PВ1 | PD |
| 2 | PD0 | PD1 | PB |
| 3 | PВ1 | PB2 | PD |
| 4 | PD1 | PD2 | PB |
| 5 | PB1 | PB3 | PD |
| 6 | PD1 | PD3 | PB |
| 7 | PB2 | PB3 | PD |
| 8 | PD2 | PD3 | PB |
| 9 | PB3 | PD4 | PD |
| 10 | PD3 | PD4 | PB |
| 11 | PB4 | PB5 | PD |
| 12 | PD4 | PD5 | PB |
| 13 | PB4 | PB6 | PD |
| 14 | PD4 | PD6 | PB |
| 15 | PВ5 | PВ6 | PD |
| 16 | PD5 | PD6 | PB |
| 17 | PB0 | PB3 | PD |
| 18 | PD5 | PD2 | PB |
| 19 | PB5 | PB6 | PD |
| 20 | PD6 | PD0 | PB |
| 21 | PB5 | PB7 | PD |
| 22 | PD0 | PD4 | PB |
| 23 | PB7 | PB6 | PD |
| 24 | PD1 | PD6 | PB |
| 25 | PB7 | PB0 | PD |
| 26 | PD2 | PD6 | PB |
| 27 | PB7 | PB4 | PD |
| 28 | PD3 | PD0 | PB |
| 29 | PB0 | PB5 | PD |
| 30 | PD0 | PD5 | PB |

При защите лабораторной работы студент должен продемонстрировать с помощью отладчика «AtmelStudio» (правила работы с ним освоить самостоятельно) работоспособность своей программы и уметь ответить на вопросы о назначении в программе тех или иных директив или команд.