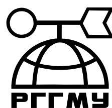
Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

А.В. Переспелов

МИКРОПРОЦЕССОРЫ

Лабораторный практикум



Санкт-Петербург
2013

УДК 621.38(075.8)

Переспелов А.В. Микропроцессоры. Лабораторный практикум. – СПб.: РГГМУ, 2013. – 72 с.

ISBN 978-5-86813-360-2

Рецензент: Леонтьев В.В., д-р техн. наук, проф. ГЭТУ (ЛЭТИ).

В лабораторный практикум включены 8 лабораторных работ, в которых моделируются с помощью САПР ATMEL AVR Studio и исследуются архитектура, язык ассемблер микропроцессоров. Каждая работа содержит теоретическую часть со сведениями, необходимыми для понимания изучаемого материала, подробную инструкцию по выполнению заданий. Подробное описание работ позволяет использовать практикум для самостоятельного изучения основ программирования микропроцессорных устройств.

Практикум предназначен для студентов, обучающихся по специальности «Информационная безопасность телекоммуникационных систем».

Perespelov A.V. Microprocessors. The laboratory course. – St. Petersburg: RSHU Publishers, 2013. – 72 pp.

The laboratory course includes 8 laboratory-based works, where the Assembler programming language is emulated in the ATMEL AVR Studio CAD exploring the architecture. Each lab-based work includes theoretical part, with information, necessary for understanding learned material and precise instructions to accomplishing tasks. Detailed description of laboratory-based works permits to use of practical course as a self-instructional tutorial to basics of programming microprocessor based devices.

This practicum is intended for students being trained by occupation in «Information security in telecommunications systems».

ISBN 978-5-86813-360-2

© Переспелов А.В., 2013
© Российский государственный гидрометеорологический университет, (РГГМУ), 2013

ПРЕДИСЛОВИЕ

Современную микроэлектронику трудно представить без такой важной составляющей, как микроконтроллеры. Устройства, которое раньше собирались на традиционных элементах, будучи собраны с применением микроконтроллеров, становится проще. Они не требуют регулировки и меньше по размерам. Применение микроконтроллеров дает практически безграничные возможности при разработке новых цифровых устройств и систем, а также доработке уже существующих устройств. Достаточно поменять программу. Микроконтроллеры применяются и в бытовых приборах, и в сложных промышленных установках. Задача разработки радиоэлектронных устройств с применением микроконтроллеров требует знания и понимания принципов их работы, а также – умение составлять управляющие программы. Особенно важно знать этапы разработки микропроцессоров специалистам в области информационной техники. Лабораторный практикум предназначен в первую очередь для закрепления знаний, полученных в теоретической части дисциплины, и приобретения навыков исследования микропроцессорных устройств на основе компьютерного моделирования.В качестве базовых, для всех приведенных примеров, использованы микросхемы микропроцессорной серии AVR фирмы Atmel [1]. Практикум построен таким образом, что представленные задания, позволяют изучить язык программирования от практически нулевого уровня, до уровня, позволяющего писать программы средней сложности. Каждая работа содержит набор заданий, которые подобраны так, чтобы за период похождения практикума были получены практические навыки работы с микропроцессорами имеющими RISC архитектуру. В начале каждого работы содержатся теоретические сведения и пояснения по ее выполнению.

ПОРЯДОК ВЫПОЛНЕНИЯ И ОФОРМЛЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Отчет по лабораторной работе должен быть выполнен с соблюдением ГОСТ, иметь титульный лист. В верхней части титульного листа указывается название университета и кафедры. Затем название и номер лабораторной работы, номер группы и фамилия выполнившего ее студента.

В каждой лабораторной работе необходимо выполнить следующие этапы:

- сформулировать цель работы;
- построить алгоритм процесса;
- спроектировать электрическую схему;
- разработать управляющую программу для этой схемы с комментариями;
- сделать выводы по лабораторной работе.

Лабораторная работа 1.
РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ

Цель работы: создать файлы в среде AVR Studio определять сегменты программы и проверять ее работу в симуляторе. Освоить: форму записи программ на Ассемблере, команды и директивы ассемблера, виды памяти, текущий сегмент, инициализацию портов, настройку портов ввода-вывода, включение внутренних нагрузочных резисторов, регистр общего назначения, регистр ввода-вывода.

Теоретическая часть

Программа для микроконтроллера – это набор кодов, который записывается в его специальную программную память. Программу должен написать программист, который разрабатывает ту или иную конкретную микропроцессорную систему.Однако программист никогда не имеет дело с кодами, т.к. для человека программирование в кодах очень неудобно. Для человека удобнее оперировать с командами, каждой из которых имеет свое осмысленное название. Поэтому для написания программ человек использует языки программирования.

Язык программирования – *это специально разработанный язык, служащий посредником между машиной и человеком. Как и обычный человеческий язык, любой язык программирования имеет свой словарь (набор слов) и правила их написания.*

В качестве слов в языке программирования выступают:

- команды (операторы);
- специальные управляющие слова;
- названия регистров;
- числовые выражения.

Задача языка – однозначно описать последовательность действий, которую должен выполнить микроконтроллер. В то же время язык должен быть удобен и понятен человеку.

В процессе создания программы программист просто пишет ее текст на компьютере точно так же, как он пишет любой другой текст. Затем программист запускает специальную программу – транслятор.

Транслятор – *это специальная программа, которая переводит текст, написанный программистом, в машинные коды, то есть в форму, понятную для микроконтроллера.*

Написанный программистом текст программы называется исходным или объектным кодом. Код, полученный в результате трансляции, называется результирующим или машинным кодом. Именно этот код записывается в программную память микроконтроллера. Для записи

результирующего кода в программную память применяются специальные устройства – программаторы.

Все языки программирования делятся на две группы:

- языки низкого уровня (машиноориентированные);
- языки высокого уровня.

Типичным примером машиноориентированного языка программирования является язык Ассемблер. Этот язык максимально приближен к системе команд микроконтроллера. Каждый оператор этого языка – это, по сути, словесное название какой-либо конкретной команды.

В процессе трансляции такая команда просто заменяется кодом операции. Составляя программу на языке Ассемблер, программист должен оперировать теми же видами данных, что и сам процессор, то есть байтами и битами. Специфика языка Ассемблер состоит в том, что набор операторов для этого языка напрямую зависит от системы команд конкретного микроконтроллера. Поэтому, если два микроконтроллера имеют разную систему команд, то и язык Ассемблер для каждого такого микроконтроллера будет свой.

Постановка задачи

«Разработать устройство управления одним светодиодным индикатором при помощи одной кнопки. При нажатии кнопки светодиод должен зажечься, при отпускании – погаснуть».

Принципиальная электрическая схема

Разработаем принципиальную электрическую схему, способную выполнять описанную выше задачу.

1. К микроконтроллеру нужно подключить светодиод и кнопку управления.
2. Для подключения к микроконтроллеру AVR любых внешних устройств используются порты ввода – вывода.

Каждый такой порт способен работать либо на ввод, либо и на вывод. Удобно светодиод подключить к одному из портов, а кнопку – к другому. В этом случае управляющая программа должна будет настроить порт, к которому подключен светодиод, на вывод, а порт, к которому подключена кнопка, на ввод. Нужен микроконтроллер, который имеет не менее двух портов. Данным условиям удовлетворяют многие микроконтроллеры AVR, например Ating2313. Эта микросхема содержит два основных и один дополнительный порт ввода-вывода. Если не считать порта А, который включается только в особом режиме, который мы пока рассматривать не будем, микроконтроллер имеет два основных порта ввода-вывода (порт В и порт D). Пример схемы устройства, позволяющего решить поставленную задачу, приведен на рис. 1 [2].

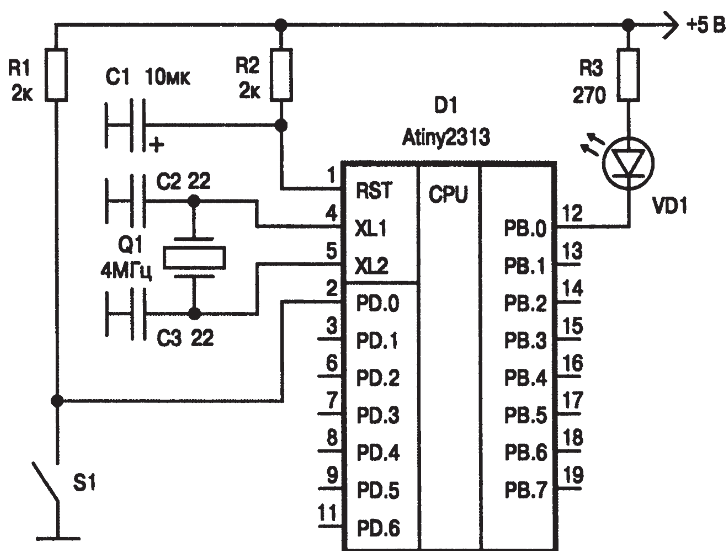


Рис. 1. Схема управления светодиодом

Для управления светодиодом использован младший разряд порта В (линия PB.O), а для считывания информации с кнопки управления использован младший разряд порта D (линия PD.O). Для подключения кнопки S1 использована классическая схема. В исходном состоянии контакты кнопки разомкнуты. Через резистор R1 на вход PD.O микроконтроллера подается «плюс» напряжения питания, что соответствует сигналу логической единицы. При замыкании кнопки напряжение падает до нуля, что соответствует логическому нулю. Считывая значение сигнала на соответствующем выводе порта, программа может определять момент нажатия кнопки. Несмотря на простоту данной схемы, микроконтроллер AVR позволяет ее упростить. А именно, исключить резистор R1, заменив его внутренним нагрузочным резистором микроконтроллера. Микроконтроллеры серии AVR имеют встроенные нагрузочные резисторы для каждого разряда порта. Подключение светодиода также выполнено по классической схеме. Это непосредственное подключение к выходу порта. Каждый выход микроконтроллера рассчитан на непосредственное управление светодиодом среднего размера с током потребления до 20 мА. В цепь светодиода включен токоограничивающий резистор R3. Для того, чтобы зажечь светодиод, микроконтроллер должен подать на вывод PB.O сигнал логического нуля. В этом случае напряжение, приложенное к цепочке R3, VD1, окажется равным напряжению питания, что вызовет ток через светодиод, и он

загорится. Если же на вывод PD.O подать сигнал логической единицы, падение напряжения на светодиоде и резисторе окажется равным нулю, и светодиод погаснет. Кроме цепи подключения кнопки и цепи управления светодиодом, на схеме можно видеть еще несколько цепей. Это стандартные цепи, обеспечивающие нормальную работу микроконтроллера. Кварцевый резонатор Q1 обеспечивает работу встроенного тактового генератора. Конденсаторы C2 и C3 – это цепи согласования кварцевого резонатора. Элементы C1, R2 – это стандартная цепь начального сброса. Такая цепь обеспечивает сброс микроконтроллера в момент включения питания.

Алгоритм

Разработка любой программы начинается с разработки алгоритма. Алгоритм – это последовательность действий, которую должен произвести микроконтроллер, чтобы достичь требуемого результата. Для простых задач алгоритм можно просто описать словами. Для более сложных задач алгоритм рисуется в графическом виде. В нашем случае алгоритм таков: После операций начальной настройки портов микроконтроллер должен войти в непрерывный цикл, в процессе которого он должен опрашивать вход, подключенный к нашей кнопке, и в зависимости от ее состояния управлять светодиодом. Операции начальной настройки:

- установить начальное значение для вершины стека микроконтроллера;
- настроить порт B на вывод информации;
- подать на выход PB.O сигнал логической единицы (потушить светодиод);
- сконфигурировать порт D на ввод;
- включить внутренние нагрузочные резисторы порта D.

Операции, составляющее тело цикла:

- прочитать состояние младшего разряда порта PD (PD.O);
- если значение этого разряда равно единице, выключить светодиод;
- если значение разряда PD.O равно нулю, включить светодиод; перейти на начало цикла.

Программа на Ассемблере

Для создания программ мы используем версию Ассемблера, предложенную разработчиком микроконтроллеров AVR – фирмой Atmel. А также воспользуемся программным комплексом «AVR Studio», разработанным той же фирмой и предназначенным для создания, редактирования, трансляции и отладки программ для AVR на Ассемблере. Текст возможного варианта, реализующий поставленную выше задачу, приведен в примере 1 [3].

```
;#################################################
;##              Пример 1                        ##
;#################################################
1               .include        “2313def.inc”   ;
2               .list                           ;
3               .cseg                           ;
4               .org            0               ;
5               .def            temp = r16      ;
6               ldi             temp, 0×7F      ;
7               out             SPL, temp       ;
8               ldi             temp, 0         ;
9               out             DDRD, temp      ;
10              ldi             temp, 0×FF      ;
11              out             DDRB, temp      ;
12              out             PORTB, temp     ;
13              out             PORTD, temp     ;
14              ldi             temp, 0×80      ;
15              out             ACSR, temp
16 main:
17              in              temp, PORTD     ;
18              out             PORTB, temp     ;
19              rjmp            main            ;
```

Программа на Ассемблере представляет собой набор команд и комментариев (иногда команды называют инструкциями). Каждая команда занимает одну отдельную строку. Их допускается перемежать пустыми строками. Команда обязательно содержит оператор, который выглядит как имя выполняемой операции. Некоторые команды состоят только из одного оператора. Другие же команды имеют один или два операнда (параметра). Операнды записываются в той же строке сразу после оператора, через пробел. Если операндов два, их записывают через запятую. Так, в строке 6 программы записана команда загрузки константы в регистр общего назначения (РОН). Она состоит из оператора idi и двух операндов temp и RAMED. В случае необходимости перед командой допускается ставить так называемую метку. Она состоит из имени метки, заканчивающимся двоеточием. Метка служит для именования данной строки программы. Затем это имя используется в различных командах для обращения к помеченной строке.

При выборе имени метки необходимо соблюдать следующие правила:

- имя должно состоять из одного слова, содержащего только латинские буквы и цифры;
- допускается также применять символ подчеркивания;

- первым символом метки обязательно должна быть буква или символ подчеркивания.

Строка 16 программы содержит метку с именем main. Метка не обязательно должна стоять в строке с оператором. Допускается ставить метку в любой строке программы. Кроме команд и меток, программа содержит комментарии.

Комментарий – это специальная запись в теле программы, предназначенная для человека. Компьютер в процессе трансляции программы игнорирует все комментарии. Комментарий может занимать отдельную строку, а может стоять в той же строке, что и команда. Начинается комментарий с символа «точка с запятой». Все, что находится после точки с запятой до конца текущей строки программы, считается комментарием. Кроме операторов, в языке Ассемблер применяются псевдооператоры или директивы. Если оператор – это некий эквивалент реальной команды микроконтроллера и в процессе трансляции заменяется соответствующим машинным кодом, который помещается в файл результата трансляции, то директива, хотя по форме и напоминает оператор, но не является командой процессора.

Директивы (псевдооператоры) – это специальные вспомогательные команды для транслятора, определяющие режимы трансляции и реализующие различные вспомогательные функции. В данной конкретной версии Ассемблера директивы выделяются особым образом. Имя каждой директивы начинается с точки. Смотри пример 1, строки с 1 по 5.

При написании программ на Ассемблере принято соблюдать особую форму записи:
- программа записывается в несколько колонок;
- аналогичные элементы разных команд принято размещать друг под другом;
- самая первая (левая) колонка зарезервирована для меток;
- если метка отсутствует, место в колонке пустует;
- следующая колонка предназначена для записи операторов;
- затем идет колонка для операндов;
- оставшееся пространство (крайняя колонка справа) предназначено для комментариев.

Подробное описание конкретной программы, приведенной в примере 1.

Директивы

.include

Присоединение к текущему тексту программы другого программного текста. Подобный прием используется практически во всех существующих языках программирования. При составлении программ часто бывает как,

что в совершенно разных программах приходится применять абсолютно одинаковые программные фрагменты. Для того, чтобы не переписывать эти фрагменты из программы в программу, их принято оформлять в виде отдельного файла с таким расчетом, чтобы этот файл могли использовать все программы, где этот фрагмент потребуется. В языке Ассемблер для присоединения фрагмента к программе используется директива include. В качестве параметра для этой директивы должно быть указано имя присоединяемого файла. Если такой оператор поставить в любом месте программы, то содержащийся в присоединяемом файле фрагмент в процессе трансляции как бы вставляется в то самое место, где находится оператор. В программе на листинге 1 в строке 1 в основной текст программы вставляется текст из файла tn2313def.inc. Файл tn2313def.inc – это файл описаний. Он содержит описание всех регистров и некоторых других параметров микроконтроллера ATiny2313. Это описание понадобится для того, чтобы в программе была возможность обращаться к каждому регистру по его имени.

.list
Директива включение генерации листинга. В данном случае листинг-это специальный файл, в котором отражается весь ход трансляции программы. Такой листинг повторяет весь текст программы, включая все присоединенные фрагменты. Против каждой строки программы, содержащей реальную команду, помещаются соответствующие ей машинные коды. Там же показываются все найденные в процессе трансляции ошибки.

.def
Директива макроопределение. Эта команда позволяет присваивать различным регистрам микроконтроллера любые осмысленные имена, упрощающие чтение и понимание текста программы. В нашем случае нам понадобится один регистр для временного хранения различных величин. Выберем для этой цели регистр r16 и присвоим ему наименование temp от английского слова temporary – временный. Данная команда выполняется в строке 3. Теперь в любом месте программы вместо имени r16 можно применять имя temp. В данной программе мы будем использовать лишь один регистр, и преимущества такого переименования здесь не очень видны.

.cseg
Директива выбора программного сегмента памяти. Микроконтроллер для хранения данных имеет три вида памяти: память программ (Flash), оперативную память (SRAM) и энергонезависимую память данных (EEPROM). Программа на Ассемблере должна работать с любым из этих трех видов памяти. Для этого в Ассемблере существует понятие «сегмент памяти». Существуют директивы, объявляющие каждый такой сегмент:

- сегмент кода (памяти программ) cseg;
- сегмент данных (ОЗУ) dseg;
- сегмент EEPROM eseg.

После объявления каждого такого сегмента он становится текущим. Это значит, что все последующие операторы относятся исключительно к объявленному сегменту. Только в сегменте кода Ассемблер описывает команды, которые затем в виде кодов будут записаны в память программ. Так как команды в программной памяти должны располагаться по порядку, одна за другой, то их размещение удобно автоматизировать. Программист не указывает, по какому адресу в памяти должна быть расположена та либо иная команда. Программист просто последовательно пишет команды. А уже транслятор автоматически размещает их в памяти. Для этого используется понятие «указатель текущего адреса». Указатель помогает транслятору разместить все команды программы по ячейкам памяти. По умолчанию считается, что в начале программы значение текущего указателя равно нулю. Поэтому первая же команда программы будет размещена по нулевому адресу. По мере трансляции программы указатель смещается в сторону увеличения адреса. Если команда имеет длину в один байт, то после ее трансляции указатель смещается на одну ячейку. Если команда состоит из двух байтов, на две. Таким образом, размещаются все команды программы.

.org

Директива принудительного указателя текущего адреса. Иногда необходимо разместить какой-либо фрагмент программы в программной памяти не сразу после предыдущего фрагмента, а в конкретном месте программной памяти. Например, начиная с какого-нибудь заранее определенного адреса. Для этого используют директиву org. Она позволяет изменить значение указателя текущего адреса. Директива org имеет всего один параметр – новое значение указателя адреса. К примеру, команда .org 0×10 установит указатель на адрес 0×10. Транслятор автоматически следит, чтобы при перемещении указателя фрагменты программы не налезали друг на друга. В случае несоблюдения этого условия транслятор выдает сообщение об ошибке. В примере 1 команда позиционирования указателя применяется всего один раз. В строке 5 указатель устанавливается на нулевой адрес. В данном случае директива org имеет чисто декларативное значение, так как в начале программы значение указателя и так равно нулю.

Операторы

Ldi

Загрузка в регистр общего назначения (РОН) числовой константы. В строке 6 программы (Листинг 1) при помощи этой команды в регистр temp

(r16) записывается числовая константа, равная максимальному адресу ОЗУ. Эта константа имеет имя RAMEND. Ее значение описано в файле tn2313def.inc. В нашем случае (для микроконтроллера ATiny2313) значение RAMEND равно $7F.
Как можно видеть, оператор ldi имеет два параметра:

- первый параметр – это имя РОН, куда помещается константа;
- второй параметр – значение этой константы.

Out
Вывод содержимого РОН в регистр ввода–вывода (РВВ). Команда также имеет два параметра:

- первый параметр – имя РВВ, являющегося приемником информации;
- второй параметр – имя РОН, являющегося источником.

В строке 7 программы содержимое регистра temp выводится в РВВ с именем SPL. I.

In
Ввод информации из регистра ввода-вывода. Имеет два параметра. Параметры те же, что и в предыдущем случае, но источник и приемник меняются местами. В строке 16 программы содержимое регистра помещается в регистр temp.

Rjmp
Команда безусловного перехода. Команда имеет всего один параметр – адрес перехода. В строке 18 программы оператор безусловного перехода передает управление на строку, помеченную меткой main. То есть на строку 16. Данная строка демонстрирует использование метки.
На самом деле в качестве параметра оператора rjmp должен выступать так называемый относительный адрес перехода. То есть, число байт, на которое нужно сместиться вверх или вниз от текущего адреса. Направление смещения (вверх или вниз) – это знак числа. Он определяется старшим битом. Язык Ассемблера избавляет программиста от необходимости подсчета величины смещения. Достаточно в нужной строке программы поставить метку, а в качестве адреса перехода указать ее имя, и транслятор сам вычислит значение этого параметра.
При использовании команды rjmp существует одно ограничение. Соответствующая команда микроконтроллера кодируется при помощи одного шестнадцатиразрядного слова. Для указания величины смещения она использует всего двенадцать разрядов. Поэтому такая команда может вызвать переход в пределах ±2 Кбайт. Если расположить метку слишком далеко от оператора rjmp, то при трансляции программы это вызовет сообщение об ошибке [4].

Описание программы

Текст программы начинается шапкой с названием программы. Шапка представляет собой несколько строк комментариев. Шапка в начале программы помогает отличать программы друг от друга. Кроме названия программы, в шапку можно поместить ее версию, а также дату написания. Самая первая команда программы – это псевдокоманда include, которая присоединяет к основному тексту программы файл описаний строка 1. В стандартном пакете AVR-Studio имеется целый набор подобных файлов описаний. Для каждого микроконтроллера серии AVR свой отдельный файл. Все стандартные файлы описаний находятся в директории «C:\ Program Files\Atmel\AVR Tools\ AvrAssembler\Appnotes\». Программисту нужно лишь выбрать нужный файл и включить подобную строку в свою программу. Без присоединения файла описаний дальнейшая программа работать не будет. Для микроконтроллера ATiny2313 файл описаний имеет название tn2313def.inc. Если файл описаний находится в указанной выше директории, то в команде include достаточно лишь указать его полное имя (с расширением). Указывать полный путь необязательно. Назначение команды .list (строка 2) определено выше (строка 3). Эта команда, как уже говорилось, присваивает регистру r16 имя temp. Дальше в программе регистр temp используется для временного хранения промежуточных величин. Уместно задаться вопросом: почему выбран именно r16, а, к примеру, не r0. Это становится понятно, если вспомнить, что регистры, начиная с r0 и заканчивая r15, имеют меньше возможностей. Например, в строке 14 программы регистр temp используется в команде Idi. Однако команда Idi не работает с регистрами r0–r15. Именно по этой причине мы и выбрали r16. Следующие две команды (строки 4, 5) подробно описаны в начале этого раздела. Они служат для выбора программного сегмента памяти и установки начального значения указателя. В строках 6 и 7 производится инициализация стека. В регистр стека SPL записывается адрес его вершыны. В качестве адреса выбран самый верхний адрес ОЗУ. Для обозначения этого адреса в данной версии Ассемблера существует специальная константа с именем RAMEND. Одной строкой записать константу в регистр стека невозможно, так как в системе команд микроконтроллеров AVR отсутствует подобная команда. Отсутствующую команду заменяем двумя другими с помощью регистра temp. Он послужит в данном случае передаточным звеном. Сначала константа RAMEND помещается в регистр temp (строка 6), а затем уже содержимое temp помещается в регистр SPL (строка 7). В строках 8–13 производится настройка портов ввода-вывода. Порт PD будет работать на ввод, а порт PB – на вывод. Для выбора нужного направления передачи информации запишем управляющие коды в соответствующие регистры DDRX. Во все разряды регистра DDRD запишем нули (настройка порта PD на ввод), а во все разряды регистра DDRB запишем единицы (настройка

порта PB на вывод). Кроме того, включим внутренние нагрузочные резисторы порта PD. Для этого запишем единицы (то есть число 0×FF) во все разряды регистра PORTD. И, наконец, в момент старта программы желательно погасить светодиод. Для этого мы запишем единицы в разряды порта PB. Все описанные выше действия по настройке порта также выполняются с использованием промежуточного регистра temp. Сначала в него помещается ноль (строка 8). Ноль записывается только в регистр DDRD (строка 9). Затем в регистр temp помещается число 0×FF (строка 10). Это число по очереди записывается в регистры DDRB, PORTB, PORTD (строки 11, 12, 13). Строки 14 и 15 включены в программу для перестраховки. Дело в том, что встроенный компаратор микроконтроллера после системного сброса остается включен. И хотя прерывания при этом отключены и срабатывание компаратора не может повлиять на работу программы, все же отключим компаратор. Именно это и делается в строках 14 и 15. Используя регистр temp производится запись константы 0×80 в регистр ACSR. Регистр ACSR предназначен для управления режимами работы компаратора, а константа 0×80, записанная в этот регистр, отключает компаратор. Настройкой компаратора заканчивается подготовительная часть программы. Подготовительная часть занимает строки 1–15 и выполняется всего один раз после включения питания или после системного сброса. Строки 16–18 составляет основной цикл программы. Основной цикл – это часть программы, которая повторяется многократно и выполняет все основные действия. Согласно алгоритму, действия программы состоят в том, чтобы прочитать состояние кнопки и перенести его на светодиод. Есть много способов перенести содержимое младшего разряда порта PD в младший разряд порта PB. В нашем случае реализован самый простой вариант. Мы просто переносим одновременно все разряды. Для этого достаточно двух операторов. Первый из них читает содержимое порта PD и запоминает это содержимое в регистре temp (строка 16). Следующий оператор записывает это число в порт PB (строка 17). Завершает основной цикл программы оператор безусловного перехода (строка 18). Он передает управление по метке main. В результате три оператора, составляющие тело цикла, повторяются бесконечно. Благодаря этому бесконечному циклу все изменения порта PD тут же попадают в порт PB. По этой причине, если кнопка S1 не нажата, логическая единица со входа PDO за один проход цикла передается на выход PBO. И светодиод не светится. При нажатии кнопки S1 логический ноль со входа PDO поступает на выход PBO, и светодиод загорается.

ЗАДАНИЕ

1. Создайте файл в среде AVR Studio.
2. Выберите в качестве начальной части программы стандартный инициализирующий еречислите файл для данной микросхемы.

3. Разработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
4. Проверьте работу программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что собой представляет инициализирующий файл?
2. Что такое директивы транслятора ассемблера?
3. Форма записи программ на Ассемблере.
4. Какие команды и директивы ассемблера нами рассмотрены?
5. Какие существуют виды памяти?
6. Что такое текущий сегмент?
7. Как происходит инициализация портов?
8. Настройка портов ввода – вывода.
9. Как происходит включение внутренних нагрузочных резисторов?
10. Что такое регистр общего назначения?
11. Что такое регистр ввода – вывода?

Лабораторная работа 2.
КОМАНДЫ ИЗ ГРУППЫ УСЛОВНЫХ ПЕРЕХОДОВ

Цель работы: освоить: процедуру ожидания, команды из группы условных переходов, команды сброса в ноль и установки в единицу одного из разрядов порта ввода-вывода.

Постановка задачи

«Разработать устройство управления одним светодиодным индикатором при помощи одной кнопки. При каждом нажатии кнопки светодиод должен поочередно включаться и отключаться. При первом нажатии кнопки светодиод должен включиться, при следующем отключиться».

Принципиальная схема

Так как для новой задачи, как и для предыдущей, необходима всего одна кнопка и всего один светодиод, то придумывать новую схему не имеет смысла. Применим для второй задачи уже знакомую нам схему, показанную на рис. 2.

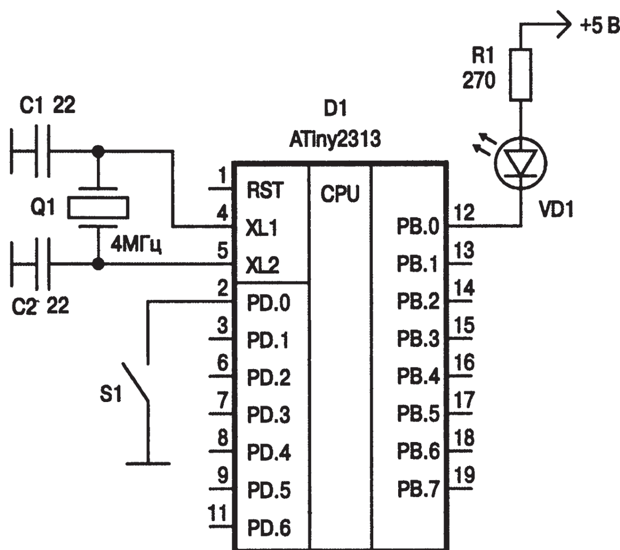


Рис. 2. Доработанная схема управления светодиодом

Алгоритм

Алгоритм задачи номер два начинается так же, как алгоритм нашей первой задачи. То есть с набора команд, выполняющих инициализацию

системы. Так как схема и принцип работы портов ввода -вывода не изменились, то алгоритм инициализации системы будет полностью повторять соответствующий алгоритм из предыдущего примера. После команд инициализации начинается основной цикл программы. Действия, выполняемые основным циклом, будут немного другими.

Опишем эти действия словами.

1. Прочитать состояние младшего разряда порта PD (PD.O).
2. Если значение этого разряда равно единице, перейти к началу цикла.
3. Если значение разряда PD.O равно нулю, изменить состояние выхода PB.O на противоположное.
4. Перейти к началу цикла.

Алгоритм описан словами. Причем это довольно общее описание. Реальный алгоритм немного сложнее. Гораздо нагляднее графический способ описания алгоритма алгоритм работы будущей программы изображен в графическом виде рис. 3 [5, 6].

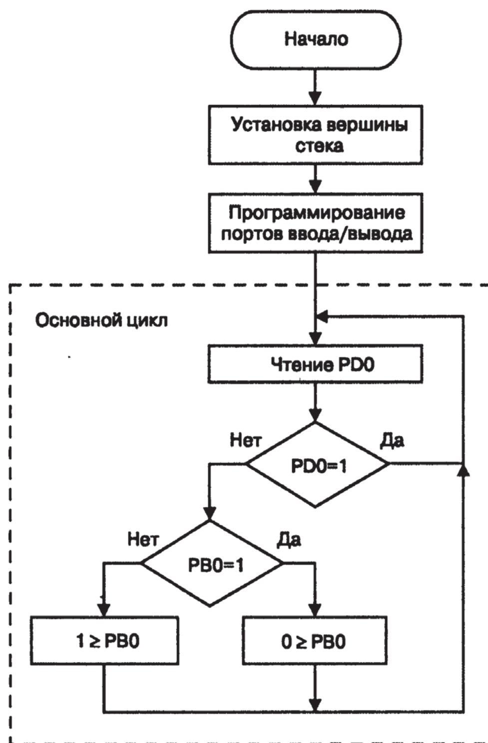


Рис. 3. Алгоритм программы для схема управления светодиодом

Такой способ отображения информации называется графом. Прямоугольниками обозначаются различные действия, выполняемые программой. Допускается объединять несколько операций в один блок и обозначать одним прямоугольником. Последовательность выполнения действий показывается стрелками. Ромбик реализует разветвление программы. Он представляет собой операцию выбора. Условие выбора записывается внутри ромбика. Если условие истинно, то дальнейшее выполнение программы продолжится по пути, обозначенному словом «Да». Если условие не выполнено, то программа пойдет по другому пути, обозначенному стрелкой с надписью «Нет». Прямоугольником со скругленными боками принято обозначать начало и конец алгоритма. В нашем случае программа не имеет конца. Основной цикл программы является бесконечным циклом. Как видно из рис. 3, сразу после старта программы выполняется установка вершины стека. Следующее действие – это программирование портов ввода-вывода. Затем начинается главный цикл программы (обведен пунктирной линией). Внутри цикла ход выполнения программы разветвляется. Первой операцией цикла является проверка состояния младшего разряда порта PD (PDO). Программа сначала читает состояние этой линии, а затем выполняет операцию сравнения. В процессе сравнения значение разряда PDO проверяется на равенство единице. Если условие выполняется, программа переходит к началу цикла (по стрелке ДА). Если условие не выполняется (PDO не равен единице), выполнение программы продолжается по стрелке «Нет», где выполняется еще одна операция сравнения. Это сравнение является частью процедуры переключения светодиода. Для того, чтобы переключить светодиод, нужно проверить его текущее состояние и перевести его в противоположное. Светодиодом управляет младший разряд порта PB (PBO). Поэтому именно его будем проверять и изменять. Работа алгоритма переключения светодиода предельно проста. Сначала оператор сравнения проверяет разряд PBO на равенство единице. Если результат проверки – истина (PBO = 1), то разряд сбрасывается в ноль (0 => PBO). Если ложно, устанавливается в единицу (1 => PBO). Сочетание символов «=>» означает операцию присвоения. Такое обозначение иногда используется в программировании при написании алгоритмов. После переключения светодиода управление передается на начало главного цикла. Итак, алгоритм готов, и можно приступать к составлению программы. Однако, приведенный выше алгоритм хорош лишь для теоретического изучения приемов программирования. На практике же он работать не будет. Дело в том, что микроконтроллер работает с такой скоростью, что за время, пока человек будет удерживать кнопку в нажатом состоянии, главный цикл программы успеет выполниться многократно (до сотни раз). Это произойдет даже в том случае, если человек постарается нажать и отпустить кнопку очень быстро. При каждом проходе главного цикла программа обнаружит

факт нажатия кнопки и переключит светодиод. В результате работа устройства будет выглядеть следующим образом. Как только кнопка будет нажата, светодиод начнет быстро переключаться. На столько быстро, что вы даже не увидите, как он мерцает. Это будет выглядеть как свечение в полнакала. В момент отпускания кнопки процесс переключения остановится, и светодиод окажется в одном из своих состояний (засветится или потухнет). В каком именно состоянии он останется, зависит от момента отпускания кнопки. А это случайная величина. Для того, чтобы решить данную проблему, необходимо усовершенствовать алгоритм. Для этого в программу достаточно ввести процедуру ожидания. Процедура ожидания приостанавливает основной цикл программы сразу после того, как произойдет переключение светодиода. Теперь программа должна ожидать момента отпускания кнопки. Как только кнопка окажется отпущенной, выполнение главного цикла возобновляется. Новый, доработанный алгоритм приведен на рис. 4. Как видно из рисунка, новый алгоритм дополнен всего двумя новыми операциями, которые и реализуют цикл ожидания. Цикл ожидания добавлен после процедуры переключения светодиода. Выполняя цикл ожидания, программа сначала читает значение бита PDO, а затем проверяет его на равенство единице. Если PDO не равно единице (кнопка нажата), то цикл ожидания повторяется. Если PDO равно единице (кнопка отпущена) то цикл ожидания прерывается, и управление перейдет на начало основного цикла. Новый алгоритм вполне работоспособен и может стать основой реальной программы.

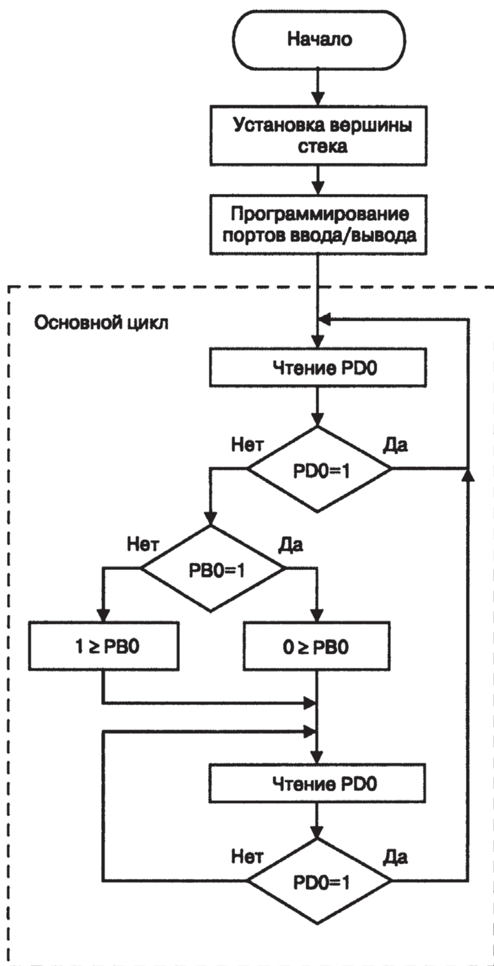


Рис. 4. Доработанный алгоритм программы

Составим такую программу. В программе применены следующие новые для нас команды:

Sbrc
Команда из группы условных переходов. Вызывает пропуск следующей за ней команды, если соответствующий разряд РОН сброшен. У команды два параметра. Первый параметр – имя регистра общего назначения, второй параметр – номер проверяемого бита. В строке 17 программы (пример 2) подобная команда проверяет нулевой разряд регистра temp. Если этот разряд равен нулю, то команда, записанная в строке 18, пропускается, и выполняется команда из строки 19. Если проверяемый бит равен единице, то пропуска не происходит, и выполняется команда в строке 18.

Sbrs
Команда, обратная предыдущей. Пропускает следующую команду, если соответствующий разряд РОН установлен в единицу. Имеет те же два параметра, что и команда sbrc. В строке 26 подобная команда проверяет значение младшего разряда регистра temp. Если проверяемый бит равен единице, то команда в строке 27 пропускается, и выполняется команда в строке 28. Если проверяемый разряд равен нулю, то выполняется строка 27.

sbi
Сброс в ноль одного из разрядов порта ввода-вывода. Команда имеет два параметра: имя порта и номер сбрасываемого разряда. В строке 22 подобная команда выполняет сброс младшего разряда порта portb.

cbi
Установка в единицу одного из разрядов порта ввода-вывода. Имеет такие же два параметра, как и предыдущая команда. В строке 24 подобная команда устанавливает младший разряд порта portb в единицу.

Описание программы

```
;################################################
;##                Пример 2                    ##
;################################################
16 main:     in       temp, PORTD   ;
17           sbrc     temp, 0       ;
18           rjmp     main          ;
19           in       temp, PINB    ;
20           sbrc     temp, 0       ;
21           rjmp     m1            ;
```

```
22              sbi         PORTB, 0      ;
23              rjmp        m2            ;
24 m1:          cbi         PORTB, 0      ;
25 m2:          in          temp, PORTD   ;
26              sbrs        temp, 0       ;
27              rjmp        m2            ;
28              rjmp        main          ;
```

Первая часть программы (строки 1–15) полностью повторяет аналогичную часть программы из предыдущего примера (листинг 1). В строке 16 производится чтение порта PORTD. Число, прочитанное из порта, записывается в регистр temp. В строке 17 производится проверка младшего разряда прочитанного числа. Если младший бит равен единице (кнопка не нажата), то управление переходит к строке 18. В строке 18 находится оператор безусловного перехода, который передает управление по метке main, то есть на начало цикла. Таким образом, пока кнопка не нажата, будет выполняться короткий цикл программы (строки 16, 17 и 18). Если кнопка нажата, младший разряд числа в регистре temp окажется равным нулю. В этом случае оператор sbrc в строке 17 передаст управление к строке 19, где начинается модуль переключения светодиода. И начинается он с чтения состояния порта PB. В строке 20 производится проверка младшего бита считанного числа. Если этот бит равен нулю, то строка 21 пропускается, и выполняется строка 22. Если младший бит равен единице, то выполняется строка 21. В строке 22 оператор sbi устанавливает младший бит регистра PORTB в единицу. А в строке 21 находится оператор безусловного перехода, который передает управление по метке m1 на строку 24. Там оператор cbi сбрасывает младший бит регистра PORTB в ноль. Таким образом, происходит переключение в младшем разряде порта PB. Ноль меняется на единицу, а единица на ноль. После переключения светодиода управление передается на строку 25. Это происходит либо при помощи команды безусловного перехода (строка 23), либо естественным путем после строки 24. Строки 25–27 содержат цикл ожидания момента отпускания кнопки. Цикл ожидания начинается с чтения содержимого порта portd (строка 25). Прочитанное значение записывается в регистр temp. Затем производится проверка младшего разряда прочитанного числа (строка 26). Если этот разряд равен нулю (кнопка еще не отпущена), выполняется строка 27 (безусловный переход на метку m 2), и цикл ожидания повторяется снова. Когда при очередной проверке кнопка окажется отпущенной, повинуясь команде sbrc (в строке 26), микроконтроллер пропустит строку 27 и перейдет к строке 28. Расположенный там безусловный переход передаст управление на начало основного цикла (по метке main).

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.
3. Контрольные вопросы.
4. Преимущества графический способ описания алгоритма.
5. Как сформирована процедура ожидания?
6. Объясните команды из группы условных переходов.
7. Как работают команды сброса в ноль и установки в единицу одного из разрядов порта ввода-вывода?

Лабораторная работа 3.
РАБОТА СО СТЕКОМ

Цель работы: Освоить: переход к подпрограмме, выход из подпрограммы, запись содержимого регистра общего назначения в стек, извлечение информации из стека, уменьшение содержимого РОН на единицу (декремент), оператор условного перехода.

Постановка задачи

Обратимся еще раз к схеме на рис. 2. В схеме используется кнопка, имеющая одну группу из двух нормально разомкнутых контактов. А если есть контакты, значит, есть и дребезг этих контактов. Рассмотрим способ борьбы с дребезгом контактов программным путем. Задача будет сформулирована следующим образом: *«Разработать схему управления светодиодом при помощи одной кнопки. При нажатии кнопки светодиод должен изменять свое состояние на противоположное (включен или выключен). При разработке программы принять меры для борьбы с дребезгом контактов».*

Схема

Принципиальная схема остается прежняя.

Алгоритм

Самый простой способ борьбы с дребезгом – введение в программу специальных задержек. Начнем с исходного состояния, когда контакты кнопки разомкнуты. Программа ожидает их замыкания. В момент замыкания возникает дребезг контактов. Дребезг приводит к тому, что на соответствующем разряде порта PD вместо простого перехода с единицы в ноль мы получим серию импульсов. Для того, чтобы избавится от их паразитного влияния, программа должна сработать следующим образом. Обнаружив первый же нулевой уровень на входе, программа должна перейти в режим ожидания. В режиме ожидания программа приостанавливает все свои действия и просто отрабатывает задержку. Время задержки должно быть выбрано таким образом, чтобы оно превышало время дребезга контактов. Такую же процедуру задержки нужно ввести в том месте программы, где она ожидает отпускания кнопки. Доработанный алгоритм, с добавлением операций антидребезговой задержки, приведен на рис. 5. Доработка свелась к включению двух процедур задержки. Одной – после обнаружения факта нажатия кнопки, а второй – после обнаружения факта ее отпускания.

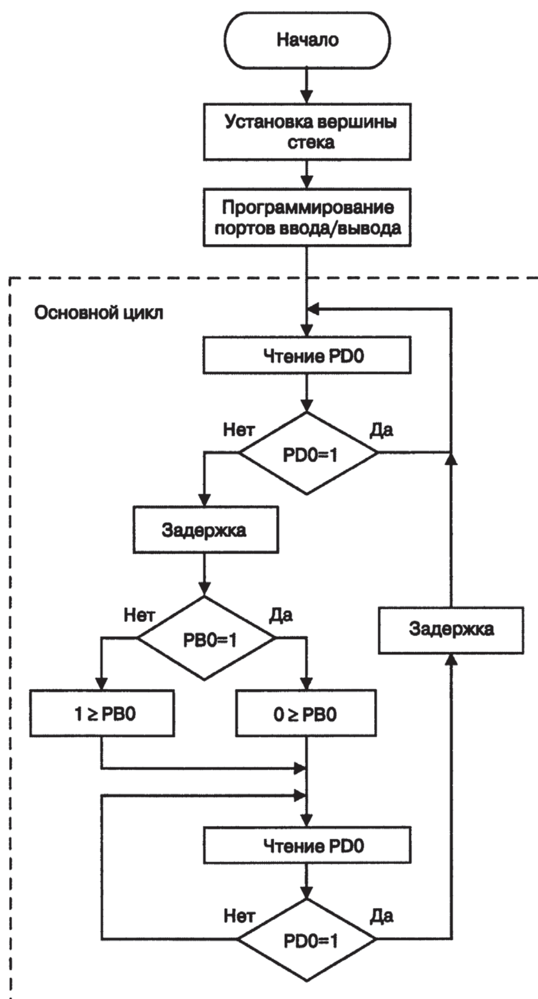


Рис. 5. Доработанный алгоритм, с добавлением операций антидребезговой задержки

Программа на Ассемблере

Новый вариант программы приведен в примере 3. В этой программе используются следующие новые для нас 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 real1, 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 строке 20 программы производится вызов подпрограммы задержки по адресу, соответствующему метке wait.

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. Команду ret можем видеть в конце подпрограммы wait (см. строку 37).

Push

Запись содержмимого регистра общего назначения в стек. У данного оператора всего один операнд – имя регистра, содержимое которого нужно поместить в стек. В строке 32 программы 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 строке 36 программы 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 строке 34 программы уменьшается на единицу содержимое регистра loop.

Brne

Оператор условного перехода. У этого оператора всего один параметр – относительный адрес перехода. Условие перехода звучит как «не равно». Проверяется регистр состояния микроконтроллера (SREG). Каждый бит этого регистра представляет собой флаг. Все флаги регистра предназначены для управления работой микроконтроллера. Кроме флага I (глобальное разрешение прерываний), этот регистр имеет ряд флагов, отражающих результаты работы различных операций. 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 (строка 34). 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. 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. В программе подобный оператор применяется в строке 35.

Описание программы

```
;################################################
;##              Пример 3                      ##
;################################################
5               .def        temp = r16       ;
6               .def        loop = r17       ;
7               ldi         temp, 0×7F       ;
8               out         SPL, temp        ;
9               ldi         temp, 0          ;
10              out         DDRD, temp       ;
11              ldi         temp, 0×FF       ;
12              out         DDRB, temp       ;
13              out         PORTB, temp      ;
14              out         PORTD, temp      ;
15              ldi         temp, 0×80       ;
16              out         ACSR, temp       ;
17 main:        in          temp, PORTD      ;
18              sbrc        temp, 0          ;
19              rjmp        main             ;
20              rcall       wait             ;
21              in          temp, PINB       ;
22              sbrc        temp, 0          ;
23              rjmp        m1               ;
24              sbi         PORTB, 0         ;
25              rjmp        m2               ;
26 m1:          cbi         PORTB, 0         ;
27 m2:          in          temp, PORTD      ;
28              sbrs        temp, 0          ;
29              rjmp        m2               ;
30              rcall       wait             ;
31              rjmp        main             ;
32 wait:        push        loop             ;
```

```
33              ldi         loop, 100       ;
34 wt1:         dec         loop            ;
35              brne        wt1             ;
36              pop         loop            ;
37              ret                         ;
```

В исходную программу добавлены новые элементы, обеспечивающие антидребезговую задержку. Так как задержка нужна в двух разных местах программы, она оформлена в виде подпрограммы. Для формирования задержки используется один дополнительный регистр общего назначения. Поэтому в начале нашей новой программы (строка 4) добавлена команда описания регистра. Регистру r17 и присваивается имя loop. Запись значения в этот регистр эквивалентна присвоению значения переменной. Подпрограмма задержки расположена в строках 32–37. Первой строке подпрограммы присвоена метка wait. По этой метке будет вызываться подпрограмма. Процедура задержки расположена в строках 33–35. Формирование задержки производится путем многократного выполнения пустого цикла. Сначала в регистр loop записывается некое начальное значение (строка 33). В нашем случае оно равно 200. Затем начинается цикл, который постепенно уменьшает значение регистра loop до нуля (строки 34 и 35). Происходит это следующим образом. В строке 34 содержимое регистра уменьшается на единицу, а в строке 35 происходит проверка содержимого на ноль. Если ноль не достигнут, то управление передается по метке wt1, и цикл повторяется. Когда же содержимое loop кажется равным нулю, очередного перехода не произойдет, и цикл задержки закончится. В нашем случае цикл задержки выполнится 200 раз. Команда brne выполняется:

- за один такт, если не вызывает перехода;
- за два такта, если вызывает переход.

Поэтому один цикл задержки будет выполняться за 3 такта. Рассмотрим две команды работы со стеком. Они предназначены для сохранения в стеке и последующего восстановления содержимого регистра loop. В начале подпрограммы (строке 32) значение loop сохраняется а перед выходом их подпрограммы (строка 36) – восстанавливается. Подобный прием придает программе одно полезное свойство. После окончания работы подпрограммы значения всех регистров микроконтроллера остается без изменений. 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. В соответствии с алгоритмом (рис. 5)

подпрограмма задержки в программе вызывается два раза. Первый раз – после окончания цикла ожидания нажатия кнопки (строка 20). Второй раз – после окончания цикла ожидания отпускания (строка 30).

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом в разработанной программе происходит переход к подпрограмме и выход из подпрограммы?
2. Как происходит запись содержимого регистра общего назначения в стек, извлечение информации из стека?
3. Что такое декремент?
4. Опишите работу оператора условного перехода.

Лабораторная работа 4.
ВЛОЖЕННЫЕ ЦИКЛЫ

Цель работы: Освоить получение необходимой задержки с помощью вложенных циклов.

Постановка задачи

«Создать устройство с одним светодиодом и одной управляющей кнопкой. Кнопка должна включать и выключать мигание светодиода. Пока кнопка отпущена, светодиод не должен светиться. Все время, пока кнопка нажата, светодиод должен мигать с частотой 5 Гц».

Воспользуемся схемой, изображенной на рис. 2.

Алгоритм

Алгоритм такой программы тоже состоит из алгоритма начальной установки и алгоритма основного цикла. Начальная установка в данном случае не отличается от начальной установки всех предыдущих программ рис. 5.

Алгоритм основного цикла программы можно описать следующим образом:

1. Произвести чтение порта PD;
2. Проверить младший разряд полученного числа (если его значение равно нулю, включить алгоритм мигания);
3. Если значение младшего разряда PD равно единице, выключить алгоритм мигания и потушить светодиод;
4. Перейти к началу основного цикла (пункт 1).

Для того, чтобы выполнить все предыдущие пункты, нам нужно описать алгоритм мигания светодиода. Он будет выглядеть следующим образом:

1. Зажечь светодиод;
2. Выдержать паузу;
3. Потушить светодиод;
4. Выдержать паузу;
5. Перейти к началу алгоритма мигания (пункт 1).

Программа содержит всего одну новую команду.

breg

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

Описание программы

```
;##################################################
;##              Пример 4                        ##
;##################################################
5              .def      temp = r16       ;
6              .def      loop1 = r17      ;
7              .def      loop2 = r18      ;
8              .def      loop3 = r19      ;
9              ldi       temp, 0×7F       ;
10             out       SPL, temp        ;
11             ldi       temp, 0          ;
12             out       DDRD, temp       ;
13             ldi       temp, 0×FF       ;
14             out       DDRB, temp       ;
15             out       PORTB, temp      ;
16             out       PORTD, temp      ;
17             ldi       temp, 0×80       ;
18             out       ACSR, temp       ;
19 main:       sbi       PORTB, 0         ;
20             in        temp, PORTD      ;
21             sbrc      temp, 0          ;
22             rjmp      main             ;
23             sbi       PORTB, 0         ;
24             rcall     wait1            ;
25             cbi       PORTB, 0         ;
26             rcall     wait1            ;
27             rjmp      main             ;
28 wait1:       push     loop1            ;
29             push      loop2            ;
30             push      loop3            ;
31             ldi       loop1, 15        ;
32 wt1:        dec       loop1            ;
33             breq      wt4              ;
34             ldi       loop2, 100       ;
35 wt2:        dec       loop2            ;
36             breq      wt1              ;
37             ldi       loop3, 255       ;
38 wt3:        dec       loop3            ;
39             brne      wt3              ;
40             rjmp      wt2              ;
41 wt4:        pop       loop3            ;
42             pop       loop2            ;
```

```
43              pop             loop1           ;
44              ret                             ;
```

Для новой задачи пришлось создать новую подпрограмму задержки. Это произошло потому, что приведенная в предыдущем примере подпрограмма не способна обеспечить задержку достаточно большой длительности. Новая подпрограмма задержки использует не один,а целых три вложенных цикла. В блок инициализации новой программы включены три оператора, определяющие три вспомогательные переменные loop1, 1oop2 и 1oop3 (строки 4, 5, 6). Теперь он занимает строки 1–18. Основной цикл программы занимает строки 19–27. Он начинается с установки единицы в младшем разряде порта PB (строка 19). В результате светодиод выключается. Следующая команда читает содержимое порта PD и помещает его в регистр temp (строка 20). В строке 21 содержимое младшего разряда полученного числа проверяется на равенство единице. Если младший разряд равен единице (кнопка отпущена), то управление передается по метке main. И цикл замыкается. Так происходит все время, пока кнопка не нажата. При каждом проходе оператор sbi (строка 19) подтверждает единицу на выходе PBO. Светодиод остается не включенным. Как только кнопка будет нажата, младший бит считанного из порта PD числа окажется равным нулю. По команде сравнения в строке 21, микроконтроллер пропустит строку 22, и управление перейдет к строке 23. В строке 23 начнется процедура мигания светодиода. Она реализует один цикл мигания и работает следующим образом. Оператор cbi (строка 23) устанавливает на выходе PBO низкий логический уровень (включает светодиод). Далее происходит вызов подпрограммы задержки (строка 24). По окончании задержки команда sbi (строка 25) переводит разряд PBO в единицу (выключает светодиод). Далее снова задержка (строка 26). Оператор безусловного перехода (строка 27) передает управление на начало основного цикла программы. Далее вся процедура повторится. Снова проверка нажатия кнопки. Если кнопка нажата, то цикл мигания повторяется. Если же кнопка окажется отпущенной, продолжения мигания не произойдет. Программа потушит светодиод и войдет в цикл ожидания нажатия кнопки (строки 19–22). Перейдем к подпрограмме формирования задержки. Текст этой подпрограммы занимает строки 28–44. Так как требуемая частота мигания должна быть равна 5 Гц, подпрограмма должна обеспечивать время задержку 1/5 = 0,2 с (200 мс). Как уже говорилось, подпрограмма представляет собой три вложенных друг в друга цикла. Самый внутренний цикл организован при помощи регистра loop 1 и включает в себя строки 38 и 39. Перед началом цикла в регистр loop l записывается число 255 (строка 37). Поэтому цикл повторяется 255 раз. Число 255 – это самое большое значение, которое можно записать в один восьмиразрядный

регистр. Как уже говорилось, задержка, формируемая таким циклом, может быть лишь чуть больше, чем 200 мкс. Для увеличения задержки организован второй цикл. Он использует регистр loop 2. Перед началом цикла в этот регистр записывается число 100 (строка 34). Цикл организован при помощи оператора dec (строка 35), который последовательно уменьшает содержимое регистра loop 2 оператор сравнения breg (строка 36), проверяет, не достигло ли значение регистра нуля. Пока значение 1oop2 не равно нулю, выполняется тело цикла (строки 37–40). В тело второго цикла включен первый цикл, использующий регистр loop1. Таким образом, цикл loop1 выполняется при каждом проходе цикла 1oop2. Общее суммарное количество проходов обоих циклов будет равно 255×100 = 25500. Этого недостаточно для создания нужной задержки. Поэтому вокруг первых двух циклов организован третий. Третий цикл использует регистр 1oop3 и построен точно так же, как второй. Перед началом работы в регистр 1oop3 записывается число 15 (строка 31). Выполнение цикла обеспечивают оператор dec (строка 32) и оператор сравнения (строка 33). При каждом проходе цикла 1oop3 выполняются вложенные циклы loop1 и 1oop2. В результате общее количество проходов встроенного цикла возрастает еще в 15 раз, что обеспечивает требуемую задержку. Кроме встроенного цикла, подпрограмма задержки содержит уже известные операторы сохранения и восстановления используемых регистров. В нашем случае подпрограмма использует три регистра. Поэтому в начале подпрограммы содержимое всех трех регистров сохраняется в стеке (строки 28, 29, 30). Перед выходом из подпрограммы содержимое всех этих регистров восстанавливается (строки 41, 42, 43). Восстановление регистров происходит в порядке, обратном порядку их запоминания. Регистр, который был записан в стек последним, извлекается первым.

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом в разработанной программе происходит формирование необходимой задержки?
2. Объясните назначение команды breg.
3. Что такое флаг Z в регистре SREG?
4. С помощью какого оператора организован цикл?

Лабораторная работа 5.
ОПЕРАТОРЫ СДВИГА

Цель работы: Освоить операторы сдвига, переход по условию «нет переноса», оператор «исключающее ИЛИ».

Постановка задачи

«Разработать автомат «Бегущие огни» для управления составной гирляндой из восьми отдельных гирлянд. Устройство должно обеспечивать «движения» огня в двух разных направлениях. Переключение направления «движения» должно осуществляться при помощи переключателя».

Схема

В соответствии с поставленной задачей устройство должно управлять восемью световыми герляндами. Для этого задействуем все восемь выходов одного из портов. Кроме того, необходимо подключать переключатель направления. Для этого понадобится еще один порт. Схема бегущих огней со светодиодами приведена на рис. 6.

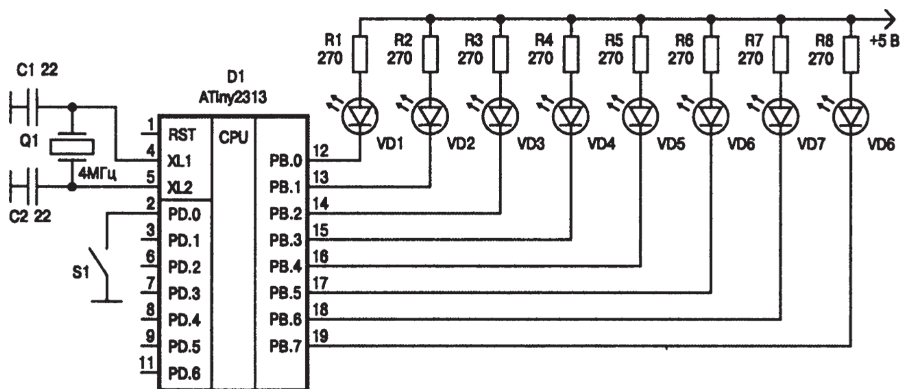


Рис. 6. Схема бегущих огней

Как видно из рис. 6, схема представляет собой доработанный вариант схемы управления светодиодом. К схеме добавлены семь дополнительных светодиодов, включенных таким же образом, как и светодиод VD1.

Алгоритм

Для создания эффекта «бегущих огней» удобнее всего воспользоваться операторами сдвига, которые имеются в системе команд микроконтроллера. Эти 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 PB, 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. Считывать состояние переключателя управления;
2. Если контакты переключателя разомкнуты, перейти к процедуре 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

Рассмотрим, как выполняется 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 нужно число с единицей в самом старшем разряде (0b10000000). Для сдвига влево в единицу устанавливается младший разряд (0b00000001).
2. Вывести значение рабочего регистра в порт PB.
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. С точки зрения наблюдателя мерцание огней выглядело бы как слабое свечение всех светодиодов.
4. Сдвинуть содержимое рабочего регистра вправо (влево) на один разряд.
5. Проверить, не окончился ли полный цикл сдвига.
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

Вариант программы приведен в примере 5. В программе будем использовать новый флаг. 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. Содержимое флага переноса может служить условием для оператора условного перехода.

lsr

Логический сдвиг вправо. Этот оператор имеет всего один параметр – имя регистра, содержимое которого сдвигается. Содержимое младшего разряда переносится в флаг переноса C, 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. То есть в данном случае в C 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 C и осуществляет переход по относительному адресу в том случае, если флаг C 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

```
;################################################
;##              Пример 5                      ##
;################################################
8           .def        loop3 = r19     ;
9           .def        rab = r20       ;
10          ldi         temp, 0×7F      ;
11          out         SPL, temp       ;
```

```
12          ldi     temp, 0          ;
13          out     DDRD, temp       ;
14          ldi     temp, 0×FF       ;
15          out     DDRB, temp       ;
16          out     PORTB, temp      ;
17          out     PORTD, temp      ;
18          ldi     temp, 0×80       ;
19          out     ACSR, temp       ;
20 main:    in      temp, PORTD      ;
21          sbrs    temp, 0          ;
22          rjmp    m3               ;
23 m1:      ldi     rab, 0b10000000;
24 m2:      ldi     temp, 0×FF       ;
25          eor     temp, rab        ;
26          out     PORTB, temp      ;
27          rcall   wait1            ;
28          lsr     rab              ;
29          brcc    m2               ;
30          rjmp    main             ;
31 m3:      ldi     rab, 0b00000001;
32 m4:      ldi     temp, 0×FF       ;
33          eor     temp, rab        ;
34          out     PORTB, temp      ;
35          rcall   wait1            ;
36          lsl     rab              ;
37          brcc    m4               ;
38          rjmp    main             ;
39 wait1:   push    loop1            ;
40          push    loop2            ;
41          push    loop3            ;
42          ldi     loop1, 15        ;
43 wt1:     dec     loop1            ;
44          breq    wt4              ;
45          ldi     loop2, 100       ;
46 wt2:     dec     loop2            ;
47          breq    wt1              ;
48          ldi     loop3, 255       ;
49 wt3:     dec     loop3            ;
50          brne    wt3              ;
51          rjmp    wt2              ;
52 wt4:     pop     loop3            ;
53          pop     loop2            ;
```

```
54            pop          loop1          ;
55            ret                         ;
```

Модуль инициализации новой программы остался таким же, как в предыдущих примерах. В новой программе он занимает строки 1–19. Дополнен лишь блок описания переменных. Кроме уже знакомых нам регистров loop1, 1oop2 и 1oop3, понадобится еще один дополнительный регистр. Этот регистр используем как рабочий в операциях сдвига. В строке 7 в качестве такого регистра выбран регистр г20, которому присваивается имя rab. В строке 20 начинается основной цикл программы. Начинается он с чтения содержимого порта PD. Результат помещается в регистр temp. В строке 21 происходит оценка младшего разряда прочитанного числа. Если этот разряд равен единице, то оператор безусловного перехода в строке 22 пропускается, и программа переходит к выполнению процедуры сдвига вправо (начало процедуры – строка 23). Если младший разряд считанного из PD числа равен нулю, то оператор rjmp в строке 22 передает управление по метке m3, и программа переходит к процедуре сдвига влево (начало процедуры – строка 31). Процедура «сдвиг вправо» работает следующим образом. В строке 23 рабочему регистру rab присваивается начальное значение. Для наглядности это число записано в двоичном формате. Затем начинается цикл сдвига (строки 24–30). Первой операцией цикла сдвига, в соответствии с алгоритмом, должна быть операция вывода содержимого регистра rab в порт PB. Однако существует одно небольшое препятствие. Если просто вывести содержимое rab в порт PB, 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 PB. Для инвертирования содержимого регистра rab воспользуемся командой eor. Если произвести операцию «Исключающее ИЛИ» между двумя байтами, значение одного из которых будет равно 0×FF, то в результате этой операции мы получим инверсное значение второго байта. Для выполнения такой операции используется вспомогательный регистр temp. В строке 24 в регистр temp записывается число 0×FF. В строке 25 производится операция «Исключающее ИЛИ» между содержимым регистров temp и rab. Результат этой операции помещается в temp. Содержимое регистра rab при этом не изменяется. В строке 26 содержимое регистра temp выводится в порт PB. Следующий этап процедуры сдвига – вызов подпрограммы задержки. Вызов этой подпрограммы происходит в строке 27. В строке 28 производится сдвиг содержимого регистра rab на один бит вправо. В строке 29 оператор brcc проверяет состояние признака переноса. Эта проверка

позволяет обнаружить момент, когда закончится полный цикл сдвига. В результате, после восьмого шага она оказывается в ячейке признака переноса. Пока С равно нулю, оператор brcc в строке 29 передает управление по метке m 2, и цикл сдвига продолжается. После восьмого шага признак переноса С станет равен единице. Поэтому перехода на начало цикла в строке 29 не произойдет, и управление перейдет к строке 30.В результате очередного девятого цикла сдвига не произойдет. Оператор безусловного перехода в строке 30 передаст управление на начало основного цикла, и программа снова приступит к проверке состояния кнопки. Процедура сдвига влево занимает строки 31–38. Эта процедура работает точно так же, как и процедура сдвига вправо. Отличия:

- начальное значение, записываемое в регистр rab (см. строку 31), равно 0b0000000l;
- вместо оператора Isr (сдвиг вправо) в строке 36 использован оператор Isl (сдвиг влево).

В качестве подпрограммы задержки применена уже известная нам подпрограмма с тремя вложенными циклами. Текст этой подпрограммы полностью скопирован из предыдущего примера (листинг 4) и расположен в строках 39–55.

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом происходит сдвиг влево?
2. Каким образом происходит сдвиг вправо?
3. Как организован переход по условию «нет переноса»?
4. Что такое флаг С в регистре SREG?
5. С помощью какого оператора выбирается направление сдвига?
6. Зачем в программе используется оператор eor?

Лабораторная работа 6.
ФУНКЦИИ ТАЙМЕРА-СЧЕТЧИКА

Цель работы: освоить формирование интервалов времени при помощи встроенного таймера/счетчика, псевдооператор присвоения, сравнение содержимого РОН с константой, переход по условию «меньше».

Постановка задачи

В предыдущих примерах для формирования задержки использовался один или несколько вложенных программных циклов. Однако такой способ приемлем далеко не всегда. Главный недостаток подобного метода состоит в том, что он полностью загружает центральный процессор. Пока микроконтроллер занят формированием задержки, он не может выполнять других задач. Еще один недостаток – невозможно с достаточной точностью выбрать время задержки. Гораздо лучшие результаты дает другой способ – формирование интервалов времени при помощи одного из встроенных таймеров/счетчиков микроконтроллера. Любой из таймеров/счетчиков может работать как с использованием прерываний, так и без прерываний. Далее рассмотрим оба эти варианта. Начнем с более простого случая. Заново сформулируем нашу задачу: *«Доработать программу «Бегущие огни», изменив процедуру формирования задержки. Новая процедура должна использовать один из внутренних таймеров/счетчиков и не использовать прерывания».*

Схема

Так как мы разрабатываем не самостоятельное устройство, а лишь усовершенствуем управляющую программу, то схема устройства не изменяется.

Алгоритм

В микроконтроллере ATtiny2313 имеются два встроенных таймера-счетчика. Поэтому сначала нужно выбрать, какой из них будем использовать. Исходить будем из заданного времени задержки 200 мс. Как известно, для формирования временных интервалов таймер/счетчик просто подсчитывает тактовые импульсы от системного генератора. Частота сигнала этого генератора в нашем случае равна 4 МГц. А период импульсов 1/4=0,25 мкс. Для того, чтобы получить на выходе 200 мс, необходимо иметь коэффициент деления, равный $800 \cdot 10^3$ (восемьсот тысяч раз). Микросхема ATiny2313 содержит два таймера. Один восьмиразрядный и один шестнадцатиразрядный. Восьмиразрядный таймер имеет максимальный коэффициент пересчета $2^8 = 256$, а шестнадцатиразрядный $2^{16} = 65536$. То есть даже шестнадцатиразрядного таймера

нам не хватит для формирования требуемой задержки. Придется воспользоваться предварительным делителем. Этот делитель производит предварительное деление тактового сигнала перед тем, как тот поступит на вход таймера-счетчика. Программным путем можно выбрать один из четырех фиксированных коэффициентов деления. Выберем самый большой возможный коэффициент деления (1024). Тогда на его выходе мы получим сигнал с частотой $4 \cdot 10^6/1024 = 3906$ Гц. Период такого сигнала будет равен 0,256 мс. Именно этот сигнал поступает на вход таймера, который обеспечивает окончательное деление. Посчитаем коэффициент деления, который должен обеспечить таймер: 200/0,256 = 780. Такой коэффициент деления может обеспечить только таймер T1. Программным путем можно выбрать один из четырех фиксированных коэффициентов деления (см. приложение). Выберем самый большой возможный коэффициент деления предделителя (1024). Тогда на его выходе мы получим сигнал с частотой $4 \cdot 10^6/1024 = 3906$ Гц. Период такого сигнала будет равен $1/3906 = 0{,}256 \cdot 10^{-3}$ с или 0,256 мс. Именно этот сигнал поступает на вход нашего таймера, который обеспечивает окончательное деление. Посчитаем коэффициент деления, который наш таймер должен нам обеспечить: 200/0,256 = 780. Такой коэффициент пересчета нам может обеспечить только таймер T1. Итак, мы определились как с выбором таймера, так и с его настройками. Теперь можно приступать к созданию новой подпрограммы задержки. Прежде, чем это сделать, попробуем описать алгоритм ее работы. Данный алгоритм предполагает, что все необходимые настройки таймера предварительного делителя произведены до первого вызова подпрограммы, таймер запущен и находится в режиме непрерывного счета. Алгоритм подпрограммы задержки представлен ниже. Записать в счетный регистр таймера T1 нулевое значение. Начать цикл проверки содержимого счетного регистра. В теле цикла программа должна многократно считывать содержимое счетного регистра таймера и проверять, не достигло ли оно своего конечного значения (то есть значения 780).

1. При достижении счетным регистром конечного значения, завершить цикл проверки.
2. Выйти из подпрограммы задержки.

Программа на Ассемблере

Программа «Бегущие огни» с новым вариантом подпрограммы задержки приведена в листинге 6. Новая подпрограмма задержки использует таймер T1 и описанный выше алгоритм работы. Рассмотрим подробнее, как работает такая программа. А начнем, как обычно, с описания новых для нас операторов.

.equ
Псевдооператор присвоения. Название оператора происходит от английского слова «эквивалентно» (equality). Используется для присвоения имен различным константам. В строке 5 листинга 6 числу 780 присваивается имя kdel. Теперь в любом месте программы вместо числа 780 можно применять константу kdel. Имя для константы имеет то же значение, что и имя для переменной. Во-первых, по осмысленному имени легко понять назначение константы. Например, kdel расшифровывается, как «коэффициент деления». А, во-вторых, это удобно при смене значения. Поменяйте в строке 5 число 780, к примеру, на 800, и везде, где бы ни встретилась константа kdel, она уже будет иметь новое значение.

cpi
Сравнение содержимого РОН с константой. Эта команда имеет два параметра. Первый параметр – имя регистра общего назначения, содержимое которого подлежит сравнению. Второй параметр некая константа, с которой сравнивается содержимое РОН. По результатам сравнения устанавливаются все флаги регистра SREG. Флаги устанавливаются точно так же, как если бы содержимое РОН вычиталось из константы. А именно: флаг переноса С устанавливается в том случае, если при вычитании данных чисел возникает перенос в старший разряд (содержимое регистра меньше константы), и сбрасывается, если нет переноса. Флаг Z (нулевой результат) устанавливается при равенстве содержимого РОН и константы и сбрасывается в случае их неравенства. Все остальные флаги устанавливаются в соответствии со своим назначением. После того, как значения флагов определены, они могут быть использованы различными условными операторами. В строке 44 программы (пример 6) оператор cpi производит сравнение содержимого регистра temp с числом 0×D0.

brlo
Переход по условию «меньше». Имеется в виду, что в предыдущей команде, в результате сравнения (или вычитания) двух операндов, первый операнд оказался меньше, чем второй. Для проверки этого условия оператор использует флаг переноса С. Переход происходит лишь в том случае, если С = 1. В строке 45 программы условный переход используется для того, чтобы передать управление на строку с меткой wt1 в том случае, если по результатам предыдущего сравнения (строка 44) оказалось, что содержимое регистра temp меньше, чем число 0×D0.

Описание программы

```
;###################################################
;##            Пример 6                          ##
;###################################################
3             .def        temp = r16        ;
4             .def        rab = r17         ;
5             .equ        kdel = 800        ;
6             .cseg                         ;
7             .org        0                 ;
8             ldi         temp, 0×7F        ;
9             out         SPL, temp         ;
10            ldi         temp, 0           ;
11            out         DDRD, temp        ;
12            ldi         temp, 0×FF        ;
13            out         DDRB, temp        ;
14            out         PORTB, temp       ;
15            out         PORTD, temp       ;
16            ldi         temp, 0×05        ;
17            out         TCCR1B, temp      ;
18            ldi         temp, 0×80        ;
19            out         ACSR, temp        ;
20 main:      in          temp, PORTD       ;
21            sbrs        temp, 0           ;
22            rjmp        m3                ;
23 m1:        ldi         rab, 0b10000000;
24 m2:        ldi         temp, 0×FF        ;
25            eor         temp, rab         ;
26            out         PORTB, temp       ;
27            rcall       wait1             ;
28            lsr         rab               ;
29            brcc        m2                ;
30            rjmp        main              ;
31 m3:        ldi         rab, 0b00000001;
32 m4:        ldi         temp, 0×FF        ;
33            eor         temp, rab         ;
34            out         PORTB, temp       ;
35            rcall       wait1             ;
36            lsl         rab               ;
37            brcc        m4                ;
38            rjmp        main              ;
39 wait1:     push        temp              ;
40            ldi         temp, 0           ;
```

```
41              out         TCNT1H, temp ;
42              out         TCNT1L, temp ;
43 wt1:         in          temp, TCNT1L ;
44              cpi         temp, low(kdel) ;
45              brlo        wt1          ;
46 wt2:         in          temp, TCNT1H ;
47              cpi         temp, high(kdel);
48              brlo        wt1          ;
49              pop         temp         ;
```

Данная программа является модификацией программы из предыдущего примера (см. листинг 5). Основное отличие новой программы от старой – полная переработка подпрограммы задержки. В связи с тем, что новая подпрограмма задержки использует таймер, для ее нормальной работы пришлось также доработать модуль инициализации основной программы. Первая доработка модуля инициализации – команда в строке 5. Эта команда описывает константу kdel, то есть коэффициент деления таймера. Обратите внимание, что значение этой константы равно 780. Если перевести это значение в двоичную форму, то количество разрядов такого числа будет больше восьми. А это значит, что для представления константы в двоичном виде потребуется не менее двух байтов. Далее в программе с этой константой сравнивается содержимое счетного регистра таймера T1, который тоже имеет шестнадцать разрядов. Однако микроконтроллеры AVR работают лишь с восьмиразрядными величинами. Счетный регистр таймера T1 представляет собой два восьмиразрядных регистра TCNT1L и TCNT1H. Используемый для сравнения оператор brio также работает с восьмиразрядными величинами. Рассмотрим как выполняется такое сравнение. Сравнение происходит в два этапа. Сначала сравниваются младшие разряды обеих величин, затем старшие. Младшее и старшее значение счетного регистра хранятся в двух соответствующих регистрах TCNT1L и TCNT1H. А для выделения младшего и старшего байта константы в языке Ассемблер существуют специальные функции low и high. Рассмотрим действие этих функций на конкретном примере. Значение нашей константы kdel равно 780.

В шестнадцатиричном виде это выглядит как 0×030C. Используя вышеописанные функции, мы можем найти старший и младший байты числа:

$$high(kdel) = 0\times03, \quad low(kdel) = 0\times0C.$$

Данные функции используются в строках 44 и 47 программы. Следующая доработка модуля инициализации это две команды, выбирающие режим работы таймера (строки 16, 17). Эти команды записывают в регистр

TCCR1B константу 0×05. В качестве вспомогательного регистра используется temp. Регистр TCCR1B – это один из двух регистров выбора режимов работы таймера T1. При записи кода 0×05 в этот регистр устанавливается коэффициент предварительного деления 1/1024, и таймер переходит в режим счета. Второй регистр конфигурации таймера называется TCCR1 A. Его значение нужно оставить по умолчанию. Последние изменения основной части программы коснулись команд вызова подпрограммы задержки. Вызов задержки происходит в строках 27 и 35. Других изменений основной части программы не потребовалось. Новая подпрограмма задержки занимает строки 39–50. Начинается подпрограмма традиционно сохранением содержимого всех используемых ею регистров. В данном случае потребовалось сохранить лишь содержимое одного регистра temp (см. строку 39). Следующие три команды производят запись нулевого значения в счетный регистр таймера T1. Сначала ноль записывается в регистр temp (строка 40). А затем содержимое temp поочередно помещается в регистры TCNT1H и TCNT1L (строки 41, 42).Порядок записи информации в пару регистров TCNT1H, TCNT1L неслучайный. Эти два регистра обладают свойством так называемой двойной буферизации. Правила работы с такими регистрами требуют, чтобы при записи значения в эти регистры сначала записывался старший регистр TCNT1H, а потом младший TCNT1L. Дело в том, что при записи старшего байта в регистр TCNT1H он не попадает сразу по назначению, а сохраняется в специальном внутреннем регистре. Когда же поступает команда записи младшего байта в регистр TCNT1L, оба байта записываются одновременно. В этом и состоит двойная буферизация. Использование двойной буферизации позволяет менять значение счетного регистра на ходу, не останавливая таймера. После записи нулевого значения в счетный регистр начинается цикл проверки. Он занимает строки 43–48 программы. Сравнение происходит в два этапа. В строках 43–45 сравнивается младшая часть счетного регистра с младшим байтом коэффициента деления. В строках 46–48 сравниваются старшие байты. Рассмотрим это подробнее. В строке 43 содержимое регистра TCNT1L помещается в регистр temp. В строке 44 происходит сравнение содержимого регистра temp с младшим байтом константы. Команда условного перехода в строке 45 передает управление на начало цикла сравнения только в том случае, если содержимое регистра еще не достигло требуемого значения. В строках 46–48 такие же операции сравнения производится для регистра TCNT1H. При этом используется старший байт константы kdel. Если старший разряд счетного регистра не достиг требуемого значения, то управление передается на метку wt1. То есть в этом случае программа опять повторяет сравнение младших разрядов. Такой порядок также диктуется наличием двойной буферизации. При чтении младшей части регистра старшая его часть запоминается в специальном внутреннем буфере.

Команда чтения старшей части регистра на самом деле читает содержимое этого буфера. Пока происходит цикл сравнения, счетчик находится в режиме счета. Содержимое счетного регистра постепенно увеличивается и, в конце концов, достигает требуемого значения. Пока происходит очередной цикл проверки, содержимое счетного регистра может даже превысить значение константы.В этом случае переходов в строке 45 и в строке 48 не произойдет. В результате подпрограмма перейдет к своему завершению. В строке 49 происходит восстановление содержимого регистра temp. А в строке 50 – выход из подпрограммы.

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом происходит присвоения имен различным константам?
2. Как организован переход по условию меньше?
3. Как организована операция сравнения содержимого РОН с константой?
4. Зачем нужны регистры TCNT1H, TCNT1L?
5. Какая разрядность у таймера T1?
6. Какая разрядность у таймера T0?

Лабораторная работа 7.
ОБРАБОТКА ПРЕРЫВАНИЙ

Цель работы: освоить использование прерываний по таймеру, выбор сегмента памяти данных, оператор резервирования памяти, оператор завершения подпрограммы обработки прерывания, команда записи содержимого РОН в ОЗУ, оператор завершения подпрограммы обработки прерывания, команда чтения информации ячейки памяти, команда разрешения прерываний.

Постановка задачи

В предыдущей работе мы использовали таймер для формирования задержки, но не использовали его главного преимущества: способности вызывать прерывания. На практике подобным образом почти никогда не поступают. Чаще всего в подобных случаях применяют прерывания по таймеру. Это позволяет более точно формировать интервалы времени, но главное – позволяет разгрузить центральный процессор. Пока таймер формирует задержку, программа может выполнять любые другие действия. Создадим новую программу «бегущих огней» с использованием прерываний по таймеру.

Схема

Схему оставим без изменений (см. рис. 6).

Алгоритм

Поставленная выше задача потребует полной переделки всей нашей программы. Ведь изменится режим работы таймера. В данном конкретном случае удобнее всего использовать режим совпадения. Точнее, его подрежим «сброс при совпадении». В этом режиме таймер сам периодически вырабатывает запросы на прерывание с заранее заданным периодом. Все функции управления «движением огней» выполняет процедура обработки прерывания. При каждом вызове прерывания процедура производит сдвиг «огней» на один шаг в нужном направлении. Для того, чтобы обеспечить такую же скорость движения «огней», как в предыдущей работе, используем те же самые коэффициенты деления. Для начала необходимо включить предварительный делитель и выбрать для него коэффициент деления 1/1024. Второй коэффициент деления (780) поместим в специальный системный регистр – регистр совпадения. Сравнение содержимого счетного регистра с содержимым регистра совпадения будет происходить на аппаратном уровне. В режиме «сброс при совпадении» таймер работает следующим образом. Сразу после запуска значение счетного регистра начнет увеличиваться. Когда это значение окажется равным

значению регистра совпадения, таймер автоматически сбросится и продолжит работу с нуля. В момент сброса таймера формируется запрос на прерывание. Для имитации бегущих огней, как и в предыдущих примерах, будем использовать операции сдвига. При этом понадобится специальный рабочий регистр, в котором будет храниться текущее состояние наших «бегущих огней». В начале программы в рабочий регистр необходимо записать исходное значение. То есть число, один из разрядов которого равен единице, а остальные – нулю. В результате операций сдвига эта единица будет перемещаться вправо или влево, создавая эффект бегущего огня. Проверка состояния кнопки и сдвиг на один шаг будет производиться при каждом вызове процедуры обработки прерывания.

Алгоритм работы программы состоит из двух независимых алгоритмов.

Во-первых, это алгоритм основной программы, а во-вторых, алгоритм процедуры обработки прерывания.

Алгоритм основной программы:

1. Настроить стек и порты ввода-вывода микроконтроллера;
2. Настроить таймер и систему прерываний;
3. Записать в рабочий регистр исходное значение;
4. Разрешить работу таймера;
5. Разрешить прерывания;
6. Перейти к выполнению основного цикла.

Так как все операции, связанные с движением огней, выполняет процедура обработки прерываний, в основном цикле программы нам ничего делать не нужно. Для простоты оставим основной цикл пустым.

Алгоритм процедуры обработки прерывания:

1. Проверить состояние переключателя режимов;
2. Если контакты переключателя разомкнуты, произвести сдвиг всех разрядов рабочего регистра на один разряд вправо. Если в результате этого сдвига единичный бит выйдет за пределы байта, создать новый единичный бит в крайней левой позиции;
3. Если контакты переключателя замкнуты, произвести сдвиг всех разрядов рабочего регистра на один разряд влево. Если в результате этого сдвига единичный бит выйдет за пределы байта, создать новый единичный бит в крайней правой позиции;
4. Вывести содержимое рабочего регистра в порт PB, предварительно проинвертировав его;
5. Закончить процедуру обработки прерывания.

Программа на Ассемблере

Текст варианта программы на языке Ассемблер приведен в примере 7. В программе встречаются несколько новых операторов.

I
Флаг глобального разрешения прерывания. Флаг I, так же, как флаги C и Z, является одним из разрядов регистра SREG. Однако управление флагом I происходит совсем по-другому. На него не влияют ни арифметические, ни логические операции, а тем более операции сравнения. Для установки и сброса этого флага в системе команд предусмотрены две специальные команды (описаны ниже). Если флаг I сброшен, то все прерывания в микроконтроллере запрещены. Если флаг установлен, работа системы прерываний разрешается. Рассмотрим теперь по порядку все новые для нас операторы.

.dseg
Оператор выбора сегмента памяти данных, то есть сегмент ОЗУ. В программе в строке 6 производится выбор именно этого сегмента.

.byte
Оператор резервирования памяти. Это один из операторов, которые действуют в сегменте памяти данных. Оператор позволяет зарезервировать один или несколько байтов (ячеек ОЗУ) для того, чтобы затем использовать их в программе. Основная цель резервирования – учет и распределение памяти. Если программист будет произвольно, по своему усмотрению, выбирать адреса ячеек ОЗУ для той либо иной задачи, то ему придется внимательно следить за тем, чтобы не выбрать повторно одну и ту же ячейку для хранения разных значений. Иначе программа при записи одного значения испортит второе, что приведет к ошибке в ее работе. Механизм резервирования памяти позволяет транслятору контролировать использование памяти и исключать двойное использование ячеек. Кроме того, подобный механизм вообще избавляет программиста от необходимости запоминать адреса. Все происходит автоматически. Оператор .byte имеет всего один параметр – количество ячеек, которые нужно зарезервировать. В нашей программе применяется лишь одна команда, резервирующая память (строка 8). В данном случае резервируется всего одна ячейка памяти. Метка buf, поставленная перед оператором, используется для обращения к зарезервированной ячейке.

reti
Оператор завершения подпрограммы обработки прерывания. Действие этого оператора аналогично действию оператора ret. Он извлекает адрес из стека и передает управление по этому адресу. Различие состоит в том, что команда reti еще и устанавливает в единицу флаг глобального разрешения прерываний I.

sis
Команда записи содержимого РОН в ОЗУ. Имеет два параметра. Первый параметр – адрес ячейки памяти, куда записываются данные. Второй параметр – имя регистра источника данных. В строке 49 программы содержимое регистра rab записывается в ОЗУ по адресу, определяемому меткой buf.

Lds
Команда чтения информации из ячейки памяти. Прочитанная информация записывается в один из РОН. Команда также имеет два параметра. Первый параметр-имя РОН, куда записываются считанные данные. Второй параметр-адрес ячейки памяти (источника данных).

sei
Команда разрешения прерываний. Эта команда устанавливает флаг I. То есть разрешает все прерывания.

Описание программы

```
;#################################################
;##              Пример 7                       ##
;#################################################
8 buf:        .byte     1          ;
9             .cseg                ;
10            .org      0          ;
11 start:     rjmp      init       ;
12            reti                 ;
13            reti                 ;
14            reti                 ;
15            rjmp      prtim1     ;
16            reti                 ;
17            reti                 ;
18            reti                 ;
19            reti                 ;
20            reti                 ;
21            reti                 ;
22            reti                 ;
23            reti                 ;
24            reti                 ;
25            reti                 ;
26            reti                 ;
27            reti                 ;
28            reti                 ;
29            reti                 ;
```

```
30 init:    ldi     temp, 0×7F        ;
31          out     SPL, temp         ;
32          ldi     temp, 0           ;
33          out     DDRD, temp        ;
34          ldi     temp, 0×FF        ;
35          out     DDRB, temp        ;
36          out     PORTB, temp       ;
37          out     PORTD, temp       ;
38          ldi     temp, 0×D         ;
39          out     TCCR1B, temp      ;
40          ldi     temp, high(kdel)  ;
41          out     OCR1AH, temp      ;
42          ldi     temp, low(kdel)   ;
43          out     OCR1AL, temp      ;
44          ldi     temp, 0×40        ;
45          out     TIMSK, temp       ;
46          ldi     temp, 0×80        ;
47          out     ACSR, temp        ;
48 main:    ldi     rab, 0b00010000   ;
49          sts     buf, rab          ;
50          sei                       ;
51 m1:      rjmp    m1                ;
52 prtim1:  push    temp              ;
53          push    rab               ;
54          lds     rab, buf          ;
55          in      temp, PORTD       ;
56          sbrs    temp, 0           ;
57          rjmp    p2                ;
58 p1:      lsr     rab               ;
59          brcc    p3                ;
60          ldi     rab, 0b10000000   ;
61          rjmp    p3                ;
62 p2:      lsl     rab               ;
63          brcc    p3                ;
64          ldi     rab, 0b00000001   ;
65 p3:      ldi     temp, 0×FF        ;
66          eor     temp, rab         ;
67          out     PORTB, temp       ;
68          sts     buf, rab          ;
69          pop     rab               ;
70          pop     temp              ;
71          reti                      ;
```

Начало программы (строки 1–5) вызывать затруднений не должно. Здесь выполняется присоединение библиотечного файла, описание двух переменных (temp и rab) и описание константы kdel. Подобные операции выполнялись в предыдущей работе. Различия начинаются в строке 6. Тут мы сталкиваемся с процедурой резервирования ячеек ОЗУ. Зарезервируем для начала всего одну ячейку. Процесс резервирования похож на процесс автоматического размещения команд в программной памяти. Здесь также используется указатель текущего адреса. При резервировании ячеек указатель перемещается от нулевого адреса вверх, в сторону увеличения адресов. Если очередная директива byte резервирует N ячеек памяти, то и указатель перемещается на N позиций. В программе весь процесс резервирования занимает всего три строки (строки 6–8):

- в строке 6 выбирается нужный нам сегмент памяти (сегмент памяти данных);
- в строке 7 выбирается новое значение для указателя в этом сегменте;
- в строке 8 происходит собственно резервирование.

Так как в строке 7 указателю присваивается значение 0×60, то именно по этому адресу будет располагаться ячейка памяти, резервируемая в строке 8. Рассмотрим почему выбран такой адрес. Ячейки ОЗУ с адресами от 0 до 0×1F совмещены с файлом регистров общего назначения, ячейки с адресами 0×20–0×5F совмещены с регистрами ввода – вывода. Ячейка с адресом 0×60 – это первая ячейка ОЗУ, предназначенная исключительно для хранения данных. Зарезервированная нами ячейка далее в программе будет использоваться в качестве буфера для хранения содержимого рабочего регистра rab в промежутке между двумя вызовами прерывания. Именно из этих соображений для нее выбрано имя buf. Резервированием памяти заканчивается модуль определений. Далее начинается непосредственно программный код, то есть код, помещаемый в программную память. Поэтому мы выбираем программный сегмент памяти (строка 9). В строке 10 устанавливается начальное значение указателя для этого сегмента. Далее начинается код самой нашей программы. Но начинается код программы совсем не так, как мы уже привыкли во всех предыдущих примерах. Строки 11–29 занимает блок команд переопределения векторов прерываний. До сих пор в наших программах мы не имели подобного блока команд, потому что до сих пор мы не использовали прерываний. Векторами прерываний называется несколько специально зарезервированных адресов в начале программной памяти, предназначенных для обслуживания прерываний. Микроконтроллер ATiny2313 имеет таблицу векторов прерываний, состоящую из 19 адресов (с адреса 0×0000 по адрес 0×0012). Каждый из этих адресов, по сути, является адресом начала процедуры обработки одного из видов прерываний. Переопределение векторов

состоит в том, что в каждую такую ячейку можно поместить команду безусловного перехода, передающую управление на адрес в программной памяти, где уже действительно начинается соответствующая процедура прерываний. Обычно программа не использует сразу все заложенные в микропроцессор прерывания. В нашем случае используется лишь одно прерывание – прерывание по совпадению таймера. Поэтому переопределение производят только для тех векторов, которые используются в данной программе. Однако и все остальные векторы принято не оставлять без внимания. По всем остальным адресам таблицы принято ставить команды-заглушки. Назначение команды – заглушки: предотвратить негативные последствия в случае ошибочного вызова незадействованного прерывания. Иногда в качестве такой заглушки применяют безусловный переход по нулевому адресу. Но удобнее всего использовать команду завершения процедуры обработки прерывания (reti). Если ненужное нам прерывание все же сработает, то оно тут же завершится, не нанеся никакого урона. Определим вектора прерываний переопределяемые в нашей программе. Во-первых, вектор нулевого адреса. По адресу 0×0000 (строка 11 программы) помешается команда безусловного перехода по метке init. В строке с этой меткой начинается основная процедура нашей программы. Как известно, нулевой адрес – это вектор начального сброса микроконтроллера. Именно с этого адреса начинается выполнение программы после системного сброса. Безусловный переход с нулевого адреса позволяет «перепрыгнуть» таблицу векторов прерываний и разместить основную программу за пределами этой таблицы. Второй переопределяемый вектор – это вектор прерываний по совпадению таймера/счетчика T1. Его адрес равен 0×0004. Сюда мы помещаем команда безусловного перехода на метку prtim1 (строка 15 программы). Именно с этой метки начинается процедура обработки данного прерывания. По всем остальным адресам таблицы помещены команды reti. Сразу за таблицей векторов прерываний начинается модуль инициализации. Модуль инициализации обязательно входит в любую программу. Наша программа – это всего лишь новый вариант программы для уже знакомой нам схемы бегущих огней. Режимы работы большинства систем микроконтроллера не изменяются. Поэтому модуль инициализации новой программы почти полностью повторяет соответствующий модуль из предыдущего примера. В предыдущей программе (листинг 6) подобный модуль занимал строки 8–19. Но есть отличия. В новой программе немного по-другому происходит инициализация таймера/счетчика.

Теперь таймер должен быть переведен в режим сброса при совпадении. Возможны два варианта реали0зации такого режима:

- сброс при совпадении в канале A;
- сброс при совпадении в канале B.

Для каждого из каналов имеется свой собственный регистр совпадения. Выберем канал А. Для того, чтобы перевести таймер/счетчик в выбранный нами режим, достаточно в регистр конфигурации таймера TCCR1B записать код 0×0D (строки 38, 39). Этот код не только переводит таймер в выбранный нами режим, но и устанавливает коэффициент предварительного деления, равный 1/1024. После того, как режим таймера выбран, нужно записать код совпадения в соответствующий регистр. Для канала А этот регистр называется OCR1A. Он имеет шестнадцать разрядов и физически состоит из двух отдельных регистров OCR1AH и OCR1AL. В каждую из этих половинок регистра записывается своя часть кода совпадения. В регистр OCR1AH записывается старший байт (строки 40, 41), а в регистр OCR1AL – младший байт (строки 42, 43) кода. Данный регистр совпадения обладает свойством двойной буферизации. Поэтому и здесь важен порядок записи двух его половинок. Сначала нужно записывать старший байт кода, а затем младший. После инициализации таймера необходимо инициализировать систему прерываний. Инициализация системы прерываний сводится к выбору нового значения маски прерываний по таймеру. Значение маски записывается в регистр TIMSK. В данном случае нам нужно разрешить лишь один вид прерываний: прерывания по совпадению в канале А. Для этого соответствующий выбранному прерыванию бит в байте маски должен быть установлен в единицу. Остальные биты должны оставаться равными нулю. Запись маски производится в строках 44 и 45. Во всем остальном новый модуль инициализации полностью соответствуют аналогичному модулю в программе из предыдущего примера. За модулем инициализации начинается основная программа. В нашем случае она занимает всего четыре строки (строки 48–51). В строках 48, 49 происходит присвоение начального значения рабочему регистру rab и сохранение этого значения в буфере buf. Как и в предыдущих примерах, рабочий регистр будет использоваться для операций сдвига, имитирующих движение нашего «огня». Начальное значение должно представлять собой двоичное число, один двоичный разряд которого равен единице, а все остальные – нулю. Затем процедура обработки прерывания будет двигать этот разряд вправо и влево. Поэтому будет логично, если первоначально нашу единичку мы расположим где-то посередине. То есть выберем в качестве начального значения, например, число 0b00010000. Что и сделано в строке 48. Так как в промежутках между двумя прерываниями эта величина будет храниться в буфере buf, в строке 49 содержимое rab помещается в этот буфер. Теперь все готово к запуску системы прерываний. В строке 50 находится команда, разрешающая все прерывания. Обратите внимание, что к этому моменту таймер/счетчик уже находится в режиме счета. Он начал работать сразу после записи значения в регистр TCCR1B. Однако все прерывания до сих пор были запрещены. Теперь, когда прерывания

разрешены, система бегущих огней сразу начинает работать. В строке 51 основная программа завершается. Так как все операции по управлению движением «огней» выполняет процедура обработки прерывания, то основной программе больше ничего делать не нужно. Поэтому в строке 51 организован бесконечный цикл. Он представляет собой безусловный переход сам в себя. Попав в такой цикл, программа будет бесконечно выполнять один и тотже оператор. Рассмотрим как происходит вызов прерывания. Таймер/счетчик непрерывно производит подсчет тактовых импульсов системного генератора. В момент, когда содержимое счетного регистра совпадет с содержимым регистра OCR1A, счетчик сбрасывается и начинает счет сначала. При очередном совпадении все повторяется. В момент сброса счетчика вызывается прерывание. Таким образом, процедура обработки прерывания выполняется периодически, каждый раз, когда счетчик досчитает до момента совпадения. Коэффициент предварительного деления и величину кода совпадения выбраны таким образом, что период, с которым происходит вызов прерывания, равен 200 мс. То есть соответствует техническое заданию. Процедура обработки прерывания заканчивается гораздо быстрее. Время выполнения этой процедуры примерно равно 6 мкс. Поэтому к тому времени, когда прерывание будет вызвано повторно, процедура обработки предыдущего прерывания уже давно закончится. Перейдем к самой процедуре обработки прерывания. Текст этой процедуры занимает строки 52–71. Начинается процедура с сохранения всех регистров, которые она в дальнейшем будет использовать (строки 52, 53), в том числе регистр rab. Теперь, в случае необходимости, основная программа сможет использовать этот регистр для своих целей. Так как в промежутке между двумя прерываниями содержимое rab хранится в буфере ОЗУ, то в строке 54 мы извлекаем это значение из буфера и помещаем в rab. Теперь все готово к операции сдвига. Но сначала нужно определить направление этого сдвига. Для этого достаточно проверить состояние контактов переключателя. Проверка производится в строках 55–57. В строке 55 читается содержимое порта PD и записывается в регистр temp. В строке 56 проверяется значение младшего разряда считанного значения. Если значение этого разряда равно единице (контакты переключателя разомкнуты), то строка 57 пропускается, и программа переходит к процедуре сдвига вправо, которая начинается в строке 58. Если контакты замкнуты, то выполняется безусловный переход в строке 57, и управление передается по метке p2, где начинается процедура сдвига влево. Процедура сдвига вправо занимает строки 58–61. Собственно сдвиг происходит в строке 58. В строке 59 происходит проверка, не дошла ли в результате этого сдвига сдвигаемая единица до последнего разряда. Также, как и в предыдущих примерах, признаком достижения конечной позиции служит появление единицы в флаге переноса. Проверка флага

переноса производится в строке 59. Если значение флага равно нулю, строка 60 программы пропускается. Если же значение флага окажется равным единице, то команда в строке 60 записывает в регистр rab новое значение. После записи этого значения единица окажется в самом старшем разряде. Таким образом организуется движение единицы по кругу (дойдя до крайней правой позиции, единица появляется слева). В строке 61 процедура сдвига вправо завершается. Управление передается по метке p3. То есть к процедуре вывода сдвинутого значения в порт. Процедура сдвига влево (строки 62–64) работает аналогично предыдущей процедуре. Отличие состоит лишь в том, что здесь применяется другая команда сдвига (строка 62). Кроме того, при достижении крайней позиции регистру присваивается другое начальное значение. Теперь единица окажется в самом младшем разряде. Таким образом организуется кольцевое движение, но в другую сторону. В строке 65 начинается процедура вывода содержимого rab в порт PB. Процедура занимает строки 65–67. Точно такая же процедура применялась и в двух предыдущих версиях программы бегущих огней. В строках 68–70 происходит подготовка к завершению процедуры обработки прерывания. Сначала содержимое rab сохраняется в буфере ОЗУ (строка 68). Затем в строках 69, 70 восстанавливаются значения регистров temp и rab. И, наконец, в строке 71 процедура обработки прерывания завершается.

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие преимущества дает использование прерываний по таймеру?
2. Зачем нужен сегмент памяти данных?
3. Что происходит при завершении подпрограммы обработки прерывания?
4. Что происходит при выполнении команды записи содержимого РОН в ОЗУ?
5. Что происходит при выполнении команды чтения информации ячейки памяти?
6. Зачем нужена команда разрешения прерываний?

Лабораторная работа 8.
ОБЕСПЕЧЕНИЕ СИСТЕМЫ ПРИОРИТЕТОВ ПРЕРЫВАНИЙ

Цель работы: Освоить использование прерываний по таймеру в режиме сброс по совпадению, обеспечение системы приоритетов прерываний, команду условного перехода по признаку переноса, команду арифметическая операция сложения, команду сложение с переносом, команду сброс вскх разрядов РОН, команду передачи данных между двумя РОН, команду чтения байта данных из программной памяти, ее модификации, директиву описания данных, инкремент.

Постановка задачи

Сформулируем задачу следующим образом: *«Разработать электронное устройство, имеющее семь входов и один звуковой выход. К каждому из входов подключен датчик, состоящий их двух нормально разомкнутых контактов. При замыкании контактов любого из датчиков устройство должно вырабатывать звуковой сигнал определенной частоты. Каждому датчику должна соответствовать своя собственная частота звукового сигнала. Если контакты всех датчиков разомкнуты, звуковой сигнал, а выходе должен отсутствовать»*. В общем случае задача формирования звука не составляет большого труда. Достаточно взять за основу схему с мигающим светодиодом, подключить вместо светодиода звуковой излучатель (например, телефонный капсюль), а в соответствующей программе (листинг 7) поменять константу задержки таким образом, чтобы частота «мигания» повысилась и достигла звукового диапазона. Диапазон частот, которые может услышать человек, лежит в пределах примерно от 50 Гц до 15 кГц. Светодиод в упомянутой выше программе мигает с частотой 4 Гц. Если уменьшить время задержки в 1000 раз, то можно получить частоту сигнала на выходе, равную 4 кГц. Эта частота как раз входит в звуковой диапазон. Предлагаемый выше способ формирования звукового сигнала реализует эту задачу программным путем. Однако для формирования звука гораздо удобнее использовать таймеры/счетчики микроконтроллера. Разработаем простейшее сигнальное устройство, которое при нажатии разных клавиш будет издавать звуки разной частоты. Допустим, мы имеем семь кнопок (датчиков). Назовем наше устройство звуковой Сигнализатор.

Схема

Поставленная выше задача решается при помощи микроконтроллера Atiny2313. Микроконтроллер имеет два встроенных таймера/счетчика. Для формирования звука лучше подходит шестнадцатиразрядный таймер. Чем больше разрядов, тем с большей точностью можно выбирать его коэффициент деления. Это очень важно для создания нотного стана.

Определимся с режимом работы таймера. Как и в случае с бегущими огнями, для генерации звука удобнее всего использовать режим СТС (сброс по совпадению). Нам просто нужно выбрать такой коэффициент деления, чтобы на выходе таймера получить колебания в звуковом диапазоне частот. Прежде всего, нужно отказаться от предварительного деления. Если частота кварцевого генератора и код, помещаемый в регистр совпадения, останутся такими же, как в предыдущем примере, то в новом варианте частота повысится более чем в тысячу раз и, как раз, попадет в нужный нам диапазон. Определимся с тем, как наш сигнал будет попадать на внешний вывод микроконтроллера. Конечно, это можно сделать программно, при помощи процедуры обработки соответствующего прерывания. Но микроконтроллер предусматривает прямой вывод сигнала на один из своих выходов. Причем предусмотрены отдельные выходы для каждого из каналов совпадения. Для канала А подобный выход называется OC1A. Он совмещен с третьим разрядом порта PB и является альтернативной функцией данного контакта. Подключение и отключение сигнала совпадения к внешнему выводу OC1A производится программным путем. Это позволяет программе в нужный момент включать или выключать звук. Так как для вывода звука будем использовать один из разрядов порта PB, то для подключения датчиков воспользуемся другим портом. А именно портом PD. Вариант принципиальной схемы описанного выше устройства показан на рис. 8 [7].

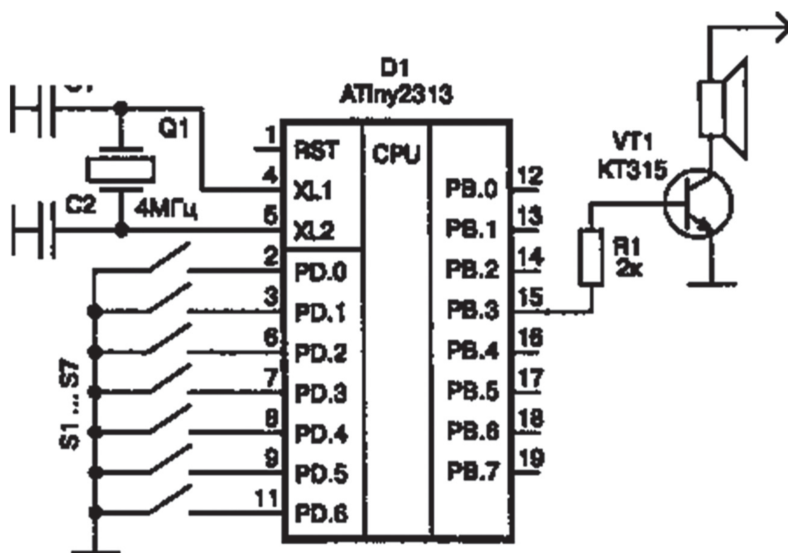


Рис. 7. Схема звукового сигнализатора

Как видно из рисунка, мы применили внешний кварцевый резонатор (Q1), естественно, не забыв при этом цепи согласования (C1, C2). При подключении датчиков используется та же схема, что использовалась до сих пор для подключения контактов переключателя. Датчики подключаются ко всем разрядам порта PD. При этом для правильной работы датчиков для каждого разряда порта PD должны быть активизированы встроенные резисторы нагрузки. Для подключения звукоизлучателя (динамика) применяется ключевой каскад на транзисторе VT1. Это самый простой способ получить звук достаточной громкости, учитывая, что наш сигнал – это прямоугольные импульсы с амплитудой, почти равной напряжению питания. Транзисторный каскад нужен лишь для повышения нагрузочной способности. Подобная схема имеет свой недостаток. В отсутствии звукового сигнала на выходе 15 микроконтроллера обязательно нужно установить низкий логический уровень. Высокий логический уровень приведет к тому, что транзистор VT1 будет постоянно открыт. Это вызовет недопустимо большой ток через головку B. Постоянно протекающий ток через обмотку динамика вызовет излишнюю потерю мощности и может даже вызвать выход из строя как транзистора, так и динамика. При составлении программы мы должны учесть этот момент.

Алгоритм

На первый взгляд алгоритм такого устройства очень простой. При замыкании контактов любого из датчиков микроконтроллер должен загрузить в регистр совпадения нужный коэффициент и подключить выход таймера к выводу OC1B. При размыкании контактов датчика микроконтроллер должен отключить сигнал от внешнего вывода OC1B и подать на него низкий логический уровень. Если контакты всех датчиков разомкнуты, то внешний вывод должен оставаться отключенным. Однако схема построена таким образом, что ничто не мешает одновременно замкнуться сразу нескольким контактам. Возникает вопрос: что делать в этом случае? Самый правильный ответ – обеспечить систему приоритетов. При замыкании нескольких контактов программа должна реагировать лишь на один из них. На тот, приоритет которого выше. Обычно в таких случаях используется следующий прием. Программа поочередно проверяет состояние всех датчиков, например, справа налево. Обнаружив первый же замкнутый контакт, программа прекращает сканирование и выдает звуковой сигнал, соответствующий этому датчику. Примем, что датчику, подключенному к входу PD.0, будет соответствовать нота «До». Следующему датчику – нота «Ре», и так далее до ноты «Си». Коэффициенты деления для каждой из нот выбираются по законам музыкального ряда.

Программа на Ассемблере

Возможный вариант программы на Ассемблере показан в примере 8. В программе использованы следующие новые для нас команды.

Brcs

Условный переход по признаку переноса. Выполняет передачу управления в случае, если признак переноса С равен единице. Данная команда является полной противоположностью уже знакомой нам команды brcc, которая, напротив, вызывает переход при отсутствии переноса.

add

Арифметическая операция сложения. Производит сложение содержимого двух РОН. Эта команда имеет два операнда, в качестве которых выступают имена складываемых регистров. В строке 54 программы к содержимому регистра count прибавляется содержимое регистра ZL. Результат помещается в ZL.

adc

Сложение с переносом. Этот оператор тоже выполняет сложение. Но в процессе сложения он учитывает перенос, возникший в предыдущей операции сложения. Команда сложения с учетом переноса используется при составлении программ, позволяющих складывать большие числа. Если каждое из слагаемых занимает больше, чем один байт, то входящие в них байты складывают в несколько этапов. Сначала складывают младшие байты, а затем старшие. При сложении младших байтов может возникнуть бит переноса (если результат оказался больше, чем 0×FF). Этот перенос и нужно учесть при сложении старших байтов. Команда adc производит сложение содержимого двух регистров, имена которых указаны в качестве ее операндов. К полученной сумме добавляется значение признака переноса. В строке 56 программы к содержимому регистра temp прибавляется содержимое регистра ZH с учетом значения признака переноса. Результат помещается в zh.

clr

Сброс вскх разрядов РОН. Команда имеет один параметр – имя РОН, разряды которого нужно сбросить. Действие этой команды равносильно записи в РОН числа 0×00.

mov

Передача данных между двумя РОН. Эта команда имеет два операнда. Первый операнд – имя регистра, получателя данных. Второй операнд – имя регистра-источника. Команда копирует содержимое одного регистра в другой.

lmp

Чтение байта данных из программы памяти. Микроконтроллеры AVR имеют отдельную память данных и отдельную память программ. Однако некоторые виды данных удобно хранить в памяти программ. К таким данным относятся наборы различных констант. В нашем случае в памяти программ удобно хранить набор коэффициентов деления для всех наших нот. Для извлечения данных из памяти программ используется команда lpm.

Хранение данных в программной памяти имеет свои особенности. Дело в том, что память программ состоит из набора шестнадцатиразрядных ячеек. Коды команд также имеют шестнадцать разрядов. Данные же нужно хранить в виде отдельных байтов. То есть в виде восьмиразрядных двоичных чисел. Для того, чтобы эффективнее использовать программную память, она организована таким образом, что в каждой шестнадцатиразрядной ячейке программной памяти можно хранить два разных байта данных. Команда lpm может читать каждый такой байт по отдельности. Для этого используется альтернативная адресация. Благодаря альтернативной адресации, программная память в режиме чтения данных имеет в два раза больше ячеек, чем при чтении кодов команд. Это нужно учитывать при использовании команды lpm. Если вы знаете адрес размещения данных согласно основного способа адресации, прежде, чем использовать его в команде lpm, ее значение необходимо умножить на два, чтобы получить адрес того же байта в альтернативной адресации. В команде lpm нет операнда, определяющего адрес ячейки, содержимое которой требуется прочитать. Этот адрес предварительно должен быть записан в регистровую пару Z. Команда lpm имеет три модификации. Ниже приведен формат всех трех модификаций этой команды:

lpm

lpm Rd, Z

lpm Rd, Z+

Первая версия команды не имеет никаких операндов. Выполняя эту команду, микроконтроллер читает содержимое ячейки программной памяти, адрес которой записан в регистровой паре Z и помещает прочитанную информацию в регистр R0. Напоминаю, что в Z нужно помещать адрес ячейки в альтернативной адресации. Вторая версия команды имеет два операнда:

- первый операнд – это имя регистра, куда будет помещен считанный байт;

- второй операнд всегда равен Z. Новая версия команды работает так же, как предыдущая. Различие только в том, что прочитанная этой командой информация помещается в РОН имя которого указано в качестве первого параметра.

Третья версия команды является модификацией второй. Она тоже имеет два операнда:
- первый операнд это имя регистра, куда помещается прочитанный байт;
- второй операнд всегда равен Z+.

От второго варианта третий отличается тем, что сразу после чтения байта происходит автоматическое увеличение содержимого регистровой пары Z на единицу. Данную команду удобно использовать для последовательного чтения ряда констант из программной памяти. При каждом последующем вызове команда будет читать следующую константу. Первая модификация команды использовалась в старых версиях микроконтроллеров и оставлена для совместимости. В новых микроконтроллерах гораздо удобнее пользоваться второй и третьей модификациями.

.dw
Директива описания данных. При помощи этой директивы описываются данные, помещаемые в память программ или в энергонезависимую память. Директива .dw описывает «слова» данных. То есть шестнадцатиричные числа, каждое из которых записывается в память в виде пары байтов. В правой части, сразу после оператора, помещается список чисел через запятую. При трансляции программы эти числа помещаются в программную память (или в EEPROM) одно за другим так же, как туда помещаются команды. В строке 63 программы в программную память помещается набор коэффициентов деления. Создается своеобразная таблица коэффициентов деления. Адрес начала таблицы соответствует метке tabkd. С точки зрения основной адресации, каждый коэффициент занимает одну шестнадцатиразрядную ячейку памяти. С точки зрения альтернативной адресации, каждый коэффициент — это два байта данных, записываемых в две соседние ячейки. Причем сначала записывается младший байт, а затем старший. При чтении этих данных используется альтернативная адресация. Если мы будем последовательно читать таблицу, начиная с адреса tabkd*2, мы получим следующую цепочку данных: 0×8C, 0×12, 0×84, 0×10, 0×B8, 0×0E, 0×E4, 0×0D, 0×60, 0×0C, 0×36, 0×0B, 0×D2, 0×09.
Первое число таблицы 4748 в шестнадцатиричном виде выглядит как 0×128C. То есть, его старший байт равен 0×12, а младший 0×8C. Шестнадцатиричное значение второго члена таблицы 4228 равно 0×1084. И так далее.

inc
Инкремент. Эта команда увеличивает содержимое одного из регистров общего назначения на единицу. У этой команды всего один параметр – имя регистра, содержимое которого нужно увеличить.

Описание программы

```
;################################################
;##              Пример 8                      ##
;################################################
5              .cseg                     ;
6              .org           0          ;
7 start:       rjmp           init       ;
8              reti                      ;
9              reti                      ;
10             reti                      ;
11             reti                      ;
12             reti                      ;
13             reti                      ;
14             reti                      ;
15             reti                      ;
16             reti                      ;
17             reti                      ;
18             reti                      ;
19             reti                      ;
20             reti                      ;
21             reti                      ;
22             reti                      ;
23             reti                      ;
24             reti                      ;
25             reti                      ;
init:
26             ldi            temp, RAMEND;
27             out            SPL, temp  ;
28             ldi            temp, 0×08 ;
29             out            DDRB, temp ;
30             ldi            temp, 0×00 ;
31             out            DDRD, temp ;
32             out            PORTB, temp ;
33             ldi            temp, 0×7F ;
34             out            PORTD, temp ;
35             ldi            temp, 0×80 ;
36             out            ACSR, temp ;
```

```
37              ldi        temp, 0×09     ;
38              out        TCCR1B, temp ;
39 m1:          ldi        temp, 0×00     ;
40              out        TCCR1A, temp ;
41 main:        clr        count          ;
42              in         temp, PIND     ;
43 m2:          lsr        temp           ;
44              brcc       m3             ;
45              inc        count          ;
46              cpi        count, 7       ;
47              brne       m2             ;
48              rjmp       m1             ;
49 m3:          lsl        count          ;
50              mov        YL, count      ;
51              ldi        YH, 0          ;
52              ldi        ZL, low(tabkd*2);
53              ldi        ZH, high(tabkd*2);
54              add        ZL, YL         ;
55              adc        ZH, YH         ;
56              lpm        YL, Z+         ;
57              lpm        YH, Z          ;
58              out        OCR1AH, YH     ;
59              out        OCR1AL, YL     ;
60              ldi        temp, 0x40     ;
61              out        TCCR1A, temp ;
62              rjmp       main           ;
63 tabkd:       .dw        4748,4480,4228,3992,3768,3556,3356;
```

Четыре первые строки программы (строки I—4) пояснений не требуют. Все эти команды знакомы нам по предыдущему примеру. Строки 5—25 занимает таблица переопределения векторов прерываний. Мы применили эту таблицу, несмотря на то, что данная программа не использует прерываний. Именно поэтому во всех ячейках таблицы, кроме ячейки с нулевым адресом, поставлены команды-заглушки. Для серьезных проектов применение таблицы считается обязательным. Это повышает надежность работы программы, а также повышает ее наглядность. Строки 26—40 занимает модуль инициализации. Начинается он с инициализации стека (строки 26, 27). Затем производится инициализация портов ввода-вывода. Команды инициализации портов занимают строки 28—34.Порт PB настраивается таким образом, что линия PB.3 работает в режиме вывода информации, а остальные линии — в режиме ввода. Все разряды порта PD настраиваются на ввод. В регистр PORTB записывается

нулевое значение. При этом на выходе PB.3 появляется низкий логический уровень, закрывающий ключ VT1. В регистр PORTD записывается код 0×7F, который включает внутренние резисторы нагрузки. Код 0×7F в двоичном виде выглядит как 0×01111111, поэтому он включает нагрузки для семи младших разрядов порта. Для старшего, восьмого разряда нагрузку включить невозможно, так как этот разряд просто отсутствует. В строках 35, 36 производится нициализация компаратора. И, наконец, в строках 37–40 производится инициализация таймера T1. Сначала настраиваются его режимы работы. Для этого в регистр TCCR1B записывается код 0×09 (строки 37,38). Этот код переводит таймер в режим CTC и выбирает коэффициент редварительного деления равным единице. Затем в регистр TCCR1A записывается ноль (строки 39, 40). Этому действию соответствует «выключение звука». В данном случае подобное утверждение справедливо. Одна из функций регистра TCCR1A – управление подключением сигнала от таймера на внешний выход OC1 A, который служит выходом звука. Включением и отключением выхода OC1A управляет разряд номер 6 регистра TCCR1A. Единица в шестом разряде подключает таймер к выходу OC1A (включает звук). При нулевом значении шестого разряда выход OC1A отключается, а соответствующему контакту микросхемы возвращается его основная функция. Он становится просто выводом порта PB, в который записан логический ноль. Этот ноль появляется на выходе, закрывая ключ. Таким образом, при выключенном звуке на выходе всегда будет ноль. Из всего вышеизложенного понятно, что запись в регистр TCCR1A кода 0×00 равносильна отключению звука. В строке 41 начинается основной цикл программы. В первой части цикла (строки 41–48) расположена процедура опроса датчиков. Для работы этой процедуры используется вспомогательный регистр count. Программа сканирует датчики один за другим, а регистр count используется для подсчета уже отсканированных датчиков. Сканирование заканчивается тогда, когда обнаружится первый же датчик с замкнутыми контактами. При этом в регистре count останется номер этого датчика. Рассмотрим работу процедуры сканирования подробнее. Перед началом сканирования содержимое регистра count обнуляется (строка 41). Затем производится чтение сигнала с контактов порта PD (строка 42). Считанный код помещается в регистр temp. Теперь код, находящийся в регистре temp, содержит полную информацию о состоянии всех семи датчиков. Каждому из семи датчиков будет соответствовать один из семи младших разрядов кода в регистре temp:

- если в момент считывания контакты датчика были разомкнуты, то соответствующий разряд будет равен единице;
- если контакты датчика были замкнуты, соответствующий разряд будет равен нулю.

Сканирование датчиков сводится к проверке семи младших разрядов кода в регистре temp. Цикл сканирования составляют строки 43—48. В процессе сканирования программа просто сдвигает содержимое регистра temp вправо. В результате каждого сдвига содержимое очередного разряда попадает в флаг признака переноса. По значению этого флага и определяется состояние датчика. Как только очередной разряд окажется равным нулю, это значит, что контакты соответствующего датчика были замкнуты. Поэтому цикл сканирования прекращается, и программа приступает к процедуре формирования звука. Цикл сканирования повторяется семь раз (по количеству датчиков). Если после семи сдвигов нулевой бит не обнаружен, значит, контакты всех семи датчиков были незамкнуты. В этом случае управление передается по метке m1, где происходит выключение звука. Затем снова считывается состояние порта, и весь цикл сканирования повторяется сначала. Логический сдвиг вправо разрядов регистра temp выполняется в строке 43. В строке 44 производится проверка признака переноса. Если он равен нулю (контакты датчика замкнуты), то происходит переход к строке 49 по метке m3. Там начинается вычисление параметров для формирования звука. Если признак переноса равен единице (контакты датчика не замкнуты), цикл сканирования продолжается. Следующая команда (строка 45) увеличивает содержимое регистра count, осуществляя подсчет датчиков. В строке 46 содержимое count сравнивается с числом 7. Таким образом ограничивается количество проходов цикла сканирования. Если содержимое count еще не достигло семи, то выполняется переход по метке m2, и цикл сканирования продолжается. Если же содержимое count окажется равным семи, то это означает, что все семь датчиков уже перебрали. В этом случае выполняется оператор безусловного перехода в строке 48, который передает управление по метке m1. Рассмотрим, что же происходит, когда процедура сканирования обнаружит сработавший датчик. В этом случае управление передается к строке 49 (метка m3). Регистр count к этому моменту содержит номер сработавшего датчика. Для датчика, подключенного к входу PD.0, этот код будет равен нулю. Для PD.1 код будет равен единице. И так далее. Теперь нужно сгенерировать звук, соответствующий коду датчика. Для этого нужно извлечь соответствующий коэффициент деления из программной памяти, поместить его в регистр совпадения и подключить сигнал с таймера на внешний выход. Сначала извлечем коэффициент деления. Как уже говорилось выше, все коэффициенты записаны в программную память и составляют таблицу коэффициентов деления (см. строку 63). Для извлечения нужного коэффициента воспользуемся командой 1pm. Но сначала необходимо вычислить адрес соответствующей ячейки памяти. Для этого вспомним, что каждый коэффициент представляет собой два байта, записанные в две соседние ячейки памяти (по альтернативной адресации). Если адрес начала таблицы

равен tabkd, то в альтернативной адресации он будет равен tabkd × 2. Очевидно, что коэффициент деления для датчика номер ноль будет занимать ячейки с адресами tabkd × 2 и tabkd × 2+1. Коэффициент деления для датчика номер один мы найдем в ячейках tabkd × 2+2 и tabkd × 2+3. И так далее. В общем случае адрес ячейки, содержащей первый байт нужного нам коэффициента мы найдем по формуле:

$$Tabkd \times 2 + Nd \times 2,$$

где Nd – это номер датчика.

Возникает задача: используя номер, записанный в регистре count, вычислить адрес ячейки, где хранится нужный коэффициент деления. Так как любой адрес – это шестнадцатиразрядное двоичное число, прийдется производить операцию шестнадцатиразрядного сложения. В системе команд микроконтроллеров AVR такая команда отсутствует. Поэтому будем складывать шестнадцатиразрядные числа побайтно. Команды вычисления адреса занимают строки 49–55. В строке 49 происходит удвоение содержимого регистра count (умножение кода датчика на два). Для удвоения используется команда логического сдвига lsl. Дело в том, что в двоичной системе логический сдвиг на один бит влево эквивалентен умножению на два. Теперь полученное в результате удвоения число необходимо прибавить к адресу начала таблицы. Для этого сформируем два шестнадцатиразрядных слагаемых. Первое слагаемое запишем в регистровую пару Y, а второе слагаемое – в регистровую пару Z. Младший байт первого слагаемого (удвоенный код датчика) берем из регистра count и помещаем в YL (строка 50). Так как датчиков всего семь (максимальное значение удвоенного кода равно 14), то старший байт первого слагаемого всегда будет равен нулю. Запишем этот ноль в регистр YH (строка 51). В качестве второго слагаемого будем использовать число, равное удвоенному значению метки tabkd. Запишем младший и старший байты этого числа в регистровую пару Z (строки 52, 53). После того, как оба слагаемых сформированы, приступаем к процессу сложения. Сначала складываем младшие байты (строка 54). Затем складываем старшие байты с учетом переноса (строка 55). В результате сложения в регистре Z получим искомый адрес. Теперь, используя этот адрес, приступим к извлечению требуемого коэффициента деления. В строке 56 извлекается первый байт коэффициента. Причем используется версия команды 1pm, которая автоматически увеличивает содержимое регистровой пары Z. Извлеченный байт помещается в младшую часть регистровой пары Y (регистр YL). Так как в содержимое Z стало на единицу большим, то очередная команда в строке 57 извлекает очередной байт коэффициента деления. Извлеченный байт помещается в старшую часть

регистровой пары Y (регистр YH). В строках 58 и 59 прочитанный только что коэффициент деления записывается в регистр совпадения OCR1 A.

При этом соблюдается правило записи для регистров с двойной буферизацией:

- сначала записывается старшая часть регистра (строка 58);
- затем записывается младшая (строка 59).

Сразу после записи коэффициента деления таймер начнет вырабатывать сигнал с нужной нам частотой. Теперь остается лишь подключить этот сигнал на выход (выполнить включение звука). Включение звука происходит в строках 60, 61. Для этого в регистр TCCR1A записывается код 0×40. Этот код имеет единицу в шестом разряде, которая и подключает сигнал от таймера к выводу OC1 A. В результате на выходе появляется звуковой сигнал, который через транзисторный ключ VT1 поступает на звуковой излучатель. Команда безусловного перехода в строке 62 завершает основной цикл программы. Она передает управление по метке main. В результате весь описанный выше процесс повторяется. Если в результате нового опроса датчиков обнаружится, что их состояние не изменилось, программа просто подтвердит все настройки таймера и генерация звука не прервется. Если состояние датчиков изменилось, то изменится и частота звука. Если же при очередном сканировании контакты всех датчиков окажутся незамкнутыми, управление перейдет по метке m1, и звук прекратится.

ЗАДАНИЕ

1. Доработайте свой вариант решения данной задачи с использованием микропроцессора AT90S8515.
2. Проверьте работоспособность программы в симуляторе AVR Studio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие преимущества дает использование прерываний по таймеру в режиме сброса по совпадению?
2. Зачем нужена система приоритетов прерываний?
3. Что происходит при выполнении команды условного перехода по признаку переноса?
4. Что происходит при выполнении команды арифметическая операция сложения?
5. Что происходит при выполнении команды сложение с переносом?
6. Что происходит при выполнении команды чтения байта данных из программной памяти?

ЛИТЕРАТУРА

1. *Мортон Дж.* Микроконтроллеры AVR. Вводный курс. Пер. с англ. – М.: Додека, 2006. – 272 с.
2. *Голубцов М.С.* Микроконтроллеры AVR от простого к сложному. – М.: СОЛОН-Пресс, 2003. – 288 с.
3. *Кравченко А.В.* 10 практических устройств на AVR микроконтроллерах – М.: Додека, 2008. – 224 с.
4. *Трамперт В.* AVR-RISC микроконтроллеры. Пер. с нем. – К.: МК-Пресс, 2006. – 464 с.
5. *Хартов В.Я.* Микроконтроллеры AVR. Практикум для начинающих. – М.: Изд. МГТУ им. Н.Э.Баумана, 2007. – 240 с.
6. *Баранов В.Н.* Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.:Додека, 2006. – 288 с.
7. *Евстигнеев А.В.* Микроконтроллеры AVR СЕМЕЙСТВА Tiny и Mega фирмы AATMEL. – М.: Додека, 2004.

СОДЕРЖАНИЕ

Предисловие. 3

Порядок выполнения и оформления лабораторных работ 3

Лабораторная работа 1. Регистры общего назначения 4

Лабораторная работа 2. Команды из группы условных переходов. 16

Лабораторная работа 3. Работа со стеком 23

Лабораторная работа 4. Вложенные циклы. 29

Лабораторная работа 5. Операторы сдвига 33

Лабораторная работа 6. Функции таймера-счетчика 39

Лабораторная работа 7. Обработка прерываний 46

Лабораторная работа 8. Обеспечение системы приоритетов
прерываний 56

Литература 68

CONTENTS

Introduction 3

The order of execution and documentation of laboratory work. 3

Laboratory work 1. General-purpose registers. 4

Laboratory work 2. Conditional state transition commands. 16

Laboratory work 3. Stack handing 23

Laboratory work 4. Nested loop 29

Laboratory work 5. Shift operators 33

Laboratory work 6. Timer functions 39

Laboratory work 7. Interrupt handling 46

Laboratory work 8. Interruption handling priority system 56

Literature 68

У Ч Е Б Н О Е   И З Д А Н И Е

Анатолий Витальевич Переспелов

МИКРОПРОЦЕССОРЫ

Лабораторный практикум

Редактор: *И.Г. Максимова*
Компьютерная верстка: *Ю.И. Климов*

ЛР № 020309 от 30.12.96.

Подписано в печать 29.10.13. Формат 60×90 1/16. Гарнитура Newton.
Печать цифровая. Усл. печ. л. 4,5. Тираж 150 экз. Зак. № 223.
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр. 98.
Отпечатано в ЦОП РГГМУ