Научно-производственное предприятие
«Учтех-Профи»

# Программирование микроконтроллера Atmega 8535 на ассемблере

Методические указания
к проведению лабораторных работ

Часть 1

Челябинск
2013

Хусаинов Р.З., Качалов А.В. Программирование микроконтроллера Atmega 8535 на ассемблере: Методические указания к выполнению лабораторных работ. – Челябинск, Учтех-Профи, 2013. – 88 с.

Методические указания предназначены для студентов средних и высших учебных заведений, изучающих дисциплины по архитектуре и программированию микропроцессорных систем. Методические указания также могут быть использованы для обучения учащихся профессионально-технических училищ и слушателей отраслевых учебных центров повышения квалификации инженерно-технических работников.

## ОГЛАВЛЕНИЕ

# 1. ОПИСАНИЕ ЛАБОРАТОРНОГО СТЕНДА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТ

## 1.1. Описание лабораторного стенда

Лабораторные работы выполняются на лабораторном стенде – модуль микроконтроллера. Внешний вид модуля приведен на рис. 1. Модуль «Микроконтроллер» предназначен для программирования и изучения функций микроконтроллера ATmega8535 семейства AVR, выпускаемого фирмой Atmel.

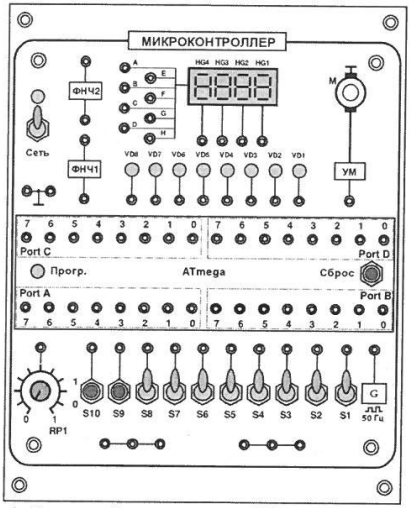



Рис. 1. Внешний вид модуля «Микроконтроллер»

На лицевой панели модуля расположены:

– переключатель «Сеть» со светодиодом индикации наличия напряжения. Переключатель осуществляет коммутацию напряжения, подаваемого на модуль;

– мнемосхему микроконтроллера с клеммами, связанными с портами ввода/вывода микроконтроллера;

– переключатели S1-S8 с выходными клеммами для подачи логических сигналов на микроконтроллер;

– кнопки S9, S10 с выходными клеммами для подачи логических сигналов на микроконтроллер;

– потенциометр RP1 с выходной клеммой для подачи аналогового напряжения на микроконтроллер;

– мнемосхема генератора низкочастотного прямоугольного сигнала 50 Гц и клемма выхода генератора;

– светодиоды VD1 – VD8 с клеммами для их подключения к источнику напряжения (например, к микроконтроллеру);

– электродвигатель постоянного тока M с усилителем мощности и клеммой для подачи на него управляющего напряжения;

– семисегментный четырехсимвольный светодиодный индикатор с клеммами подачи напряжения на сегменты A, B, C, D, E, F, G, H, а также на общую точку каждого сегмента индикатора;

– два фильтра низкой частоты для фильтрации ШИМ-сигналов на выходе микроконтроллера.

Основные характеристики изучаемого микроконтроллера ATmega8535 приведены в табл. 1.

Таблица 1. Краткая характеристика микроконтроллера ATmega8535

| № | Параметр | Значение |
|---|---|---|
| 1 | Объем памяти программ (Flash-память) | 8 кБ |
| 2 | Объем энергонезависимой памяти (память EEPROM) | 512 Б |
| 3 | Объем ОЗУ | 512 Б |
| 4 | Количество программируемых входов/выходов | 32 |
| 5 | 8-битные таймеры/счетчики с ШИМ | 2 шт. |
| 6 | 16-битный таймер/счетчик с ШИМ | 1 шт. |
| 7 | Количество каналов АЦП | 8 |
| 8 | Разрядность АЦП | 10 |
| 9 | Аналоговый компаратор | есть |
| 10 | Источники внешних прерываний | 3 шт. |
| 11 | Универсальный приемопередатчик USART | есть |
| 12 | TWI – интерфейс | есть |
| 13 | SPI – интерфейс | есть |
| 14 | JTAG – интерфейс | нет |
| 15 | Напряжение электропитания | 2,7 – 5,5 В |
| 16 | Диапазон частот кварцевого резонатора | 0…16 МГц |
| 17 | Частота установленного кварцевого резонатора | 8 МГц |

### 1.2. Порядок выполнения лабораторных работ

*При подготовке к лабораторной работе необходимо:*

– ознакомиться с ее содержанием и, пользуясь рекомендованной литературой и лекциями, изучить теоретические положения, на которых базируется работа;

– в соответсвии со своим вариантом задания написать листинг программы;

– выполнить проверку рабоспособности программы на симуляторе AVR-studio;

– ответить на контрольные вопросы к лабораторной работе.

*Перед выполнением лабораторной работы необходимо:*

– представить отчет по предыдущей работе;

– представить листинг программы и необходимые расчет по своему варианту задания к выполняемой лабораторной работе;

– ответить на вопросы, задаваемые преподавателем.

*При выполнении лабораторной работы необходимо:*

– ввести программу в компьютер и показать ее работоспособность преподавателю на AVR-studio;

– произвести сборку схемы;

– *только после разрешения* преподавателя включить питание и приступить к программированию микроконтроллера и проверки его работы;

– представить программу на мироконтроллере на проверку преподавателю;

– по окончании работы привести в порядок рабочее место.

### 1.3. Оформление отчетов по лабораторным работам

Все отчеты должны быть выполнены и сданы на проверку каждым студентом *индивидуально*. Работа считается сданной, если она проверена, не содержит ошибок и принята преподавателем.

Отчет помимо правильно оформленного титульного листа с указанием номера лабораторной работы, ее названия, фамилии и инициалов студента, выполнившего работу, номера группы и фамилии и инициалов преподавателя должен содержать (порядок оформления пунктов также должен соблюдаться):

1. Цель работы.

2. Функциональная схема устройства. Указываются все используемые входы/выходы микроконтроллера, периферийные элементы (тумблеры или потенциометры для подачи дискретных и аналоговых входных сигналов, резисторы, светодиоды, семисегментные индикаторы и т.п. выводимых данных), подключение питания микроконтроллера, подключение кварцевого генератора.

3. Предварительные расчеты (если они требуются). Обычно эти расчеты включают данные, требуемые для выполнения программы:

– выбор необходимых для решения задачи периферийных устройств – таймеров, прерываний, АЦП и т.п.;

– расчеты требуемых характеристик работы периферийных устройств (периодов работы таймеров, разрядности АЦП и т.д.);

– настройка регистров ввода/вывода для задания требуемого режима работы периферийных устройств.

4. Листинг программы. Листинг необходимо приводить обязательно с комментариями по основным элементам программы: пояснения по переменным, назначение группы инструкций в программе (стек, инициализация портов, инициализация таймера T0 и т.д.).

5. Дисассемблер программы. Дисассемблер должен приводиться полностью для всей программы, включая таблицу векторов прерываний и память данных во FLASH-области..

6. Стек (если он используется). Информацию по стеку во время исполнения программы с указанием: вершины стека, информации, которая записывается в стек и необходимыми пояснениями.

7. Другие необходимые пункты в соответствии с требованиями к лабораторной работе.

8. Выводы по работе.

*Примечание*: при составлении отчета необходимо выполнять требования ЕСКД: использовать пунктуацию, записывать название выполняемого пункта, схемы и таблицы должны быть пронумерованы и аккуратно построены.

## 2. ЛАБОРАТОРНЫЕ РАБОТЫ

### Работа № 1. Знакомство со средой программирования и отладки микроконтроллеров AVR Studio

#### Цель работы

Ознакомиться с программной средой программирования и отладки микроконтроллеров AVR Studio фирмы Atmel. Написать и отладить простейшую программу.

#### Программа работы

1. Создать новый проект в среде AVR Studio.
2. Набрать простейшую программу на ассемблере микроконтроллера Atmega8535.
3. Отладить программу в симуляторе.
4. Записать программу в микроконтроллер и проверить правильность ее функционирования.

#### Пояснения к работе

В состав лабораторного стенда входит 8-разрядный микроконтроллер AVR ATmega8535 и необходимые элементы для исследования его периферийных устройств и большинства функциональных возможностей.

Поскольку микроконтроллер является программируемым, пользователь должен освоить его программирование в различных программных средах и на различных языках высокого и низкого уровня.

Среди наиболее популярных методов написания программ отметим:

– написание программ на машинном коде микроконтроллера. Программы, написанные таким способом, являются наиболее быстродействующими, однако для их написания требуется высокая квалификация программиста и глубокое знание архитектуры процессора;

– написание программы на ассемблере. Написание программ на ассемблере существенно проще, чем на машинном коде, однако также требует высокой квалификации программиста. Следует отметить, что язык ассемблера обычно жестко привязан к конкретному типу микропроцессора и может существенно отличаться для разных микропроцессоров одного производителя;

– написание программы на языке высокого уровня (например, Си). Такие программы обычно являются кросс-платформенными, то есть практически не отличаются по синтаксису для микропроцессоров разных типов. Пользователь пишет программу на языке высокого уровня, а компилятор преобразует ее на ассемблер и машинный код конкретного микропроцессора. Однако обычно использование языка высокого уровня при написании программ для микроконтроллеров может существенно ограничить их быстродействие.

#### Создание проекта в среде AVR Studio

Для программирования на ассемблере в данных методических указаниях используется программное обеспечение AVR Studio. Для написания программы необходимо создать проект.

С лабораторным комплексом поставляется программа AVR Studio ver. 4. Действия по созданию проекта и работе с программой в разных версиях AVR Studio могут несколько отличаться, однако большинство действий соответствуют изложенным далее пунктам.

1. Запустить программу AVR Studio. Ярлык для запуска программы находится на рабочем столе или в меню «Пуск» Windows. В появившемся окне приветствия программы будет предложено создать новый проект или открыть существующий (рис. 1).

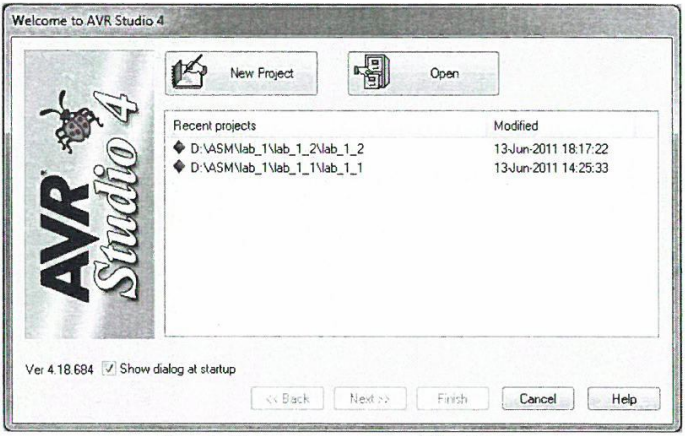



Рис. 1. Окно приветствия AVR Studio

2. При выборе нового проекта появляется окно, в котором предлагается выбрать язык программы – Atmel Avr Assembler или Avr GCC (рис. 2).

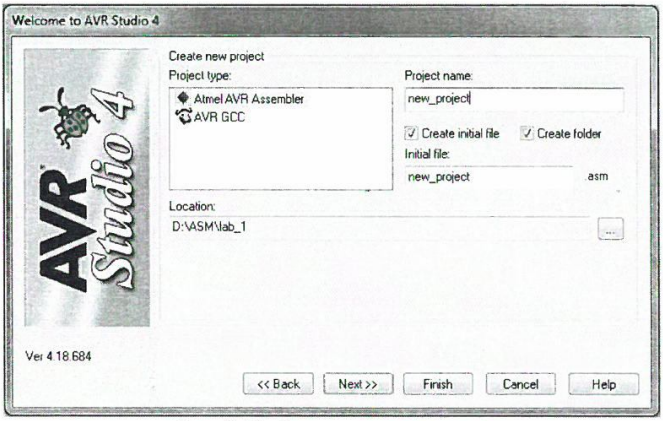



Рис. 2. Выбор языка программы

Здесь необходимо выбрать Atmel Avr Assembler, указать имя проекта и имя инициализационного файла с расширением ***.asm**. Рекомендуется, чтобы имя проекта и инициализационного файла совпадали.

**Очень важно: не допускать в имени файла, проекта или пути символы кириллицы.**

После этого следует нажать кнопку «Next» (Далее).

3. После выбора языка программы и имени проекта появляется окно выбора платформы (Debug platform) и устройства (Device). Здесь необходимо выбрать AVR Simulator и процессор ATmega8535, который используется в лабораторном стенде. После этого следует нажать кнопку «Finish» (рис. 3).

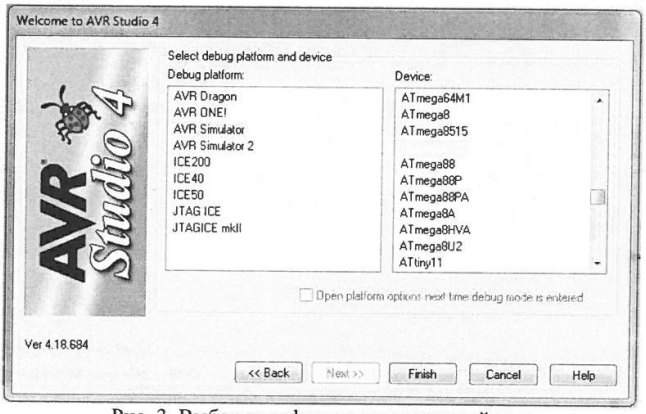



Рис. 3. Выбор платформы и типа устройства

4. При нажатии на кнопку «Finish» в программе открывается рабочее окно программирования (рис. 4).

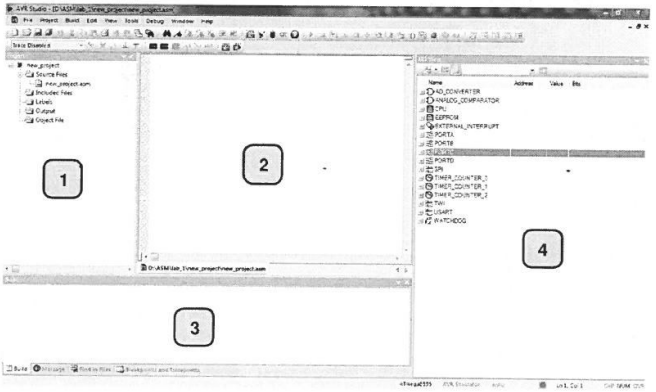


Рис. 4. Рабочее окно программы AVR Studio

Рабочее окно содержит несколько областей:

– окно проекта «Project». Здесь отображается структура проекта, которая содержит его имя и список подключенных файлов и библиотек (рис. 4, окно «1»);

– центральная рабочая область, в которой осуществляется непосредственно набор программы (окно «2»);

– окно текущих сообщений «built» программы (окно «3»);

– окно регистров микроконтроллера «I/O View» (окно «4»).

**Набор и компиляция программы**

Программа на языке ассемблера микроконтроллера ATmega8535 содержит набор команд и директив данного микропроцессора, которые приведены в приложении А методических указаний. Большая часть этих директив будет рассмотрена далее в ходе выполнения последующих лабораторных работ.

В данной работе предлагается набрать простейшую программу, отладить ее в симуляторе и записать в микроконтроллер, на котором затем проверить ее корректную работу. Необходимо отметить, что данная программа является пробной, и основной ее задачей является освоение студентом методики написания программ на языке ассемблера микроконтроллера ATmega8535.

**Задание.** Написать программу, осуществляющую вывод числа 01010101 или 10101010 на PORTC микроконтроллера, в зависимости от состояния сигнала на входе PORTB0.

```
;-----------------------------------------------------------------
;Пробная программа для микроконтроллера ATMEGA8535
;Входы: PORTB0
;Выходы: PORTC0...PORTC7

.include "m8535def.inc"  ;подключение библиотеки контроллера
.cseg                    ;начало сегмента кода
.org 0
reset:
   ldi r16,0xFF
   out DDRC,r16      ;назначение PORTC на вывод
   clr r16
   out DDRB,r16      ;назначение PORTB на ввод
main:
   sbis PINB,0       ;если на PINB0=0, то
   rjmp PINB0_is_0   ;переход на "PINB0_is_0"
   ldi r16,0xAA      ;иначе вывод на PORTC числа 0xAA (1010 1010)
   out PORTC,r16
   rjmp main         ;далее - возврат на main
PINB0_is_0:          ;PINB0=0
   ldi r16,0x55      ;вывод на PORTC числа 0x55 (0101 0101)
   out PORTC,r16
   rjmp main         ;далее - возврат на main
;-----------------------------------------------------------------
```

После набора программы необходимо произвести ее ассемблирование, то есть сборку, при которой ассемблер производит проверку синтаксиса написанной программы и при отсутствии ошибок преобразует код ассемблера в машинный

код микропроцессора. При этом формируется файл с расширением ***.hex**, который далее записывается в микроконтроллер.

Для ассемблирования написанной программы необходимо нажать кнопку **F7 «Assemble»**. При отсутствии ошибок в окне текущих сообщений «built» AVR Studio отображается сообщение об успешном завершении операции, а при наличии ошибок выводится их список и положение в тексте программы.

**Отладка программы на симуляторе**

В программное обеспечение AVR Studio встроен симулятор микроконтроллеров, с помощью которого можно отладить программу, найти в ней ошибки и исправить неточности в алгоритме, не осуществляя непосредственно программирования микросхемы контроллера. Также симулятор может быть полезен при написании программ в домашних условиях.

Симулятор AVR Studio осуществляет симуляцию микроконтроллера, его портов, таймеров, АЦП, прерываний и т.д. Поскольку симулятор работает с **hex**-файлом проекта, то для запуска эмулятора необходимо написать программу и исключить из нее явные ошибки, с которыми создание **hex**-файла невозможно.

Для отладки программы в симуляторе необходимо после ее написания нажать сочетание клавиш **Ctrl+F7 «Assemble and Run»**, после чего будет произведено ассемблирование программы и запуск эмулятора.

После успешной компиляции начало программы будет отмечено желтой стрелкой. Для управления процессами отладки программы в строке меню AVR Studio располагается меню «debug», а также панель функциональных кнопок управления симулятором, назначение которых приведено в табл. 1.

Таблица 1.1. Назначение кнопок управления эмулятором

| Название | Сочетание клавиш | Назначение |
|---|---|---|
| Start Debugging (начать отладку) | Ctrl+Shift+Alt+F5 | Начать процесс отладки программы в эмуляторе. |
| Stop Debugging (остановить отладку) | Ctrl+Shift+F5 | Остановить процесс отладки программы в эмуляторе. |
| Run (запуск эмуляции) | F5 | Эмулятор запускается и циклически производит эмуляцию программы без отображения текущих изменений регистров контроллера на экране. |
| Break (приостановить эмуляцию) | Ctrl+F5 | Команда временно приостанавливает эмулятор без потери данных. |
| Reset (остановить эмуляцию) | Shift+F5 | Команда полностью останавливает эмулятор с потерей данных эмуляции. |
| Step into (Сделать один шаг вперед) | F11 | Команда извлекает только одну инструкцию. После ее завершения все рабочие экраны эмулятора обновляются. |
| Auto Step (Автовыполнение) | Alt+F5 | Команда выполняет эмуляцию программы в пошаговом автоматическом режиме с оперативным обновлением информации на экранах эмулятора после каждого шага. |

| Название | Сочетание клавиш | Назначение |
|---|---|---|
| Toggle breakpoint (Точка остановки) | F9 | Комадна добавляет или убирает пользовательскую точку остановки, при достижении которой программа будет приостановлена. |

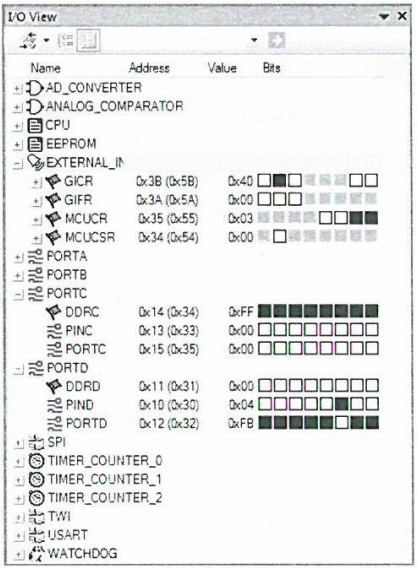



Рис. 5. Контроль состояния регистров микроконтроллера при симуляции

При симуляции программы рекомендуется выполнять пошаговое выполнение инструкций, пользуясь командой Step into (табл. 1). При этом необходимо контролировать содержимое регистров микроконтроллера, а также портов ввода/вывода.

Контроль состояния регистров и отдельных устройств микроконтроллера осуществляется в окне «4» (рис. 4). Каждое устройство можно развернуть и увидеть содержимое его регистров управления и контроля (рис. 5).

**Запись программы в контроллер и проверка ее работоспособности**

После успешной отладки программы необходимо «прошить» ее в микроконтроллер (т.е. записать ее в память микроконтроллера) и проверить работу.

Для «прошивки» программы используется та же программа AVR Studio.

***Примечание:*** в предыдущих версиях лабораторного стенда использовалась программа «USB ISP», описание которой приведено в прил. 6.

Последовательность действий с использованием AVR Studio следующая:

– включить переключатель «Сеть» модуля «Микроконтроллер» для подачи на него напряжения питания;

– в меню «Tools» AVR Studio выбрать пункт «Program AVR», в котором указать способ соединения «Auto Connect». При правильном подключении к персональному компьютеру модуль микроконтроллера инициализируется как USB COM-порт. Необходимо, чтобы номер этого порта был не более COM9. В противном случае необходимо найти этот порт в диспетчере оборудования Windows и переназначить номер порта.

– если номер USB COM-порта соответствует указанным требованиям, то происходит подключение микороконтроллера к среде AVR-Studio и появляется окно программирования контроллера (рис. 6);

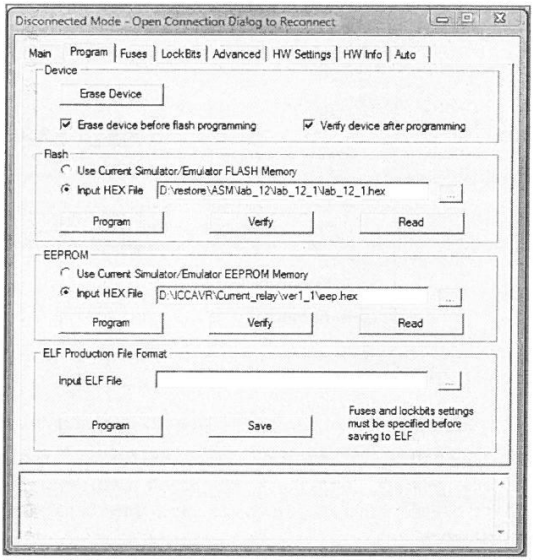



Рис. 6. Окно программирования микроконтроллера

– в окне программирования необходимо выбрать вкладку main, в которой необходимо выбрать тип используемого контроллера, после чего перейти на вкладку «Program», в которой выбрать графу «Flash». В этой графе требуется указать путь к *.hex файлу проекта, после чего произвести запись программы в микроконтроллер нажатием кнопки «Program».

По завершении записи программы необходимо проверить ее корректную работу на микроконтроллере и показать результат преподавателю.

## Работа № 2. Порты ввода/вывода микроконтроллера Atmega8535

### Цель работы

Освоить работу с портами ввода/вывода микроконтроллеров AVR.

### Программа работы

1. Изучить необходимый теоретический материал, быть готовым ответить на вопросы преподавателя.

2. Изучить представленные в лабораторной работе примеры программ по работе с портами ввода/вывода.

3. Написать и отладить собственную программу логического уравнения в соответствии с вариантом.

### Пояснения к работе

Порты ввода/вывода микроконтроллера предназначены для передачи и приема информации и последующей ее обработки. Микроконтроллеры различных типов содержат различное количество портов ввода/вывода. Изучаемый в данном курсе микроконтроллер Atmega8535 содержит 4 порта ввода/вывода, выполненных 8-разрядными: PORTA, PORTB, PORTC, PORTD.

Порты ввода/вывода непосредственно связаны с выводами микросхемы микроконтроллера, при этом каждый конкретный вывод микроконтроллера жестко «привязан» к конкретному разряду порта ввода/вывода. На рис. 1 представлено расположение выводов микроконтроллера Atmega8535, выполненного в DIP-корпусе.

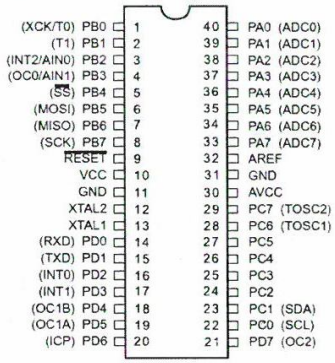



Рис. 1. Расположение выводов микросхемы контроллера ATmega8535 (DIP40)

По умолчанию выводы микросхемы контроллера предназначены для выполнения функций ввода/вывода информации в соответствии с настройками регистров портов ввода/вывода. Однако функции большинства выводов микросхемы могут быть программно изменены. При этом к выводам микросхемы могут быть присоединены выходы таймеров, приемопередатчиков, входы аналогово-цифрового преобразователя, контроллера внешних прерываний.

Альтернативные функции выводов микроконтроллера представлены на рис. 1 (в скобках).

Каждый порт состоит из трех регистров, с помощью которых осуществляется установка направления работы порта и выдача/сбор информации (табл. 1).

Таблица 1. Регистры портов ввода/вывода

| Наименование порта | Регистры | | |
|---|---|---|---|
| | Регистр направления | Регистр данных | Регистр состояния |
| Порт A | DDRA | PORTA | PINA |
| Порт B | DDRB | PORTB | PINB |
| Порт C | DDRC | PORTC | PINC |
| Порт D | DDRC | PORTD | PIND |

*Регистры направления* определяют режим работы портов ввода/вывода. Если в каком-либо разряде регистра установлена логическая «1», то соответствующий вывод микросхемы контроллера (рис. 1) работает на вывод информации из микроконтроллера. В противном случае соответствующий вывод микросхемы (рис. 1) работает на ввод информации в микроконтроллер.

*Регистры данных* предназначены для передачи данных на выводы микросхемы контроллера. Если в каком-либо разряде регистра установлена логическая «1», а соответствующий вывод микросхемы сконфигурирован как выход с помощью регистра направления, то на вывод микросхемы контроллера подается сигнал, соответствующий логической «1». В противном случае на вывод микросхемы контроллера подается сигнал логического «0». Регистры используются для вывода информации из микроконтроллера.

*Регистры состояния* предназначены для отображения текущего состояния сигналов на выводах микросхемы контроллера. Так, если на выводе микросхемы находится сигнал логической «1», то соответствующий разряд регистра направления находится в состоянии логической «1». Регистры используются для ввода информации в микроконтроллер.

**Инициализация порта на ввод и на вывод информации**

Для того, чтобы освоить принципы установки порта на ввод или на вывод информации, необходимо иметь представление о реализации его структуры. Каждый вывод порта выполнен по схеме, представленной на рис. 2.

При инициализации порта на ввод информации микроконтроллер принимает сигналы, поступающие от внешнего объекта. Эти сигналы подаются на выводы микросхемы (рис. 2, вывод PIN).

Диоды VD1 и VD2 выполняют функцию защиты микропроцессора от сигналов, величина которых находится за пределами диапазона 0…5В. Так, если напряжение на входе микроконтроллера превышает +5В, то открывается диод VD1, а если напряжение оказывается меньше 0В, то открывается диод VD2 (рис. 2). Конденсатор C1 выполняет защиту микроконтроллера от импульсных помех.

Поскольку микропроцессор имеет высокое входное сопротивление, его вход является восприимчивым к воздействию помех. По этой причине важно, чтобы сигнал, подаваемый на микропроцессор, имел однозначное значение логического «0» или логической «1». Для этого в микроконтроллере присутствуют так называемые подтягивающие резисторы (Pull Up).

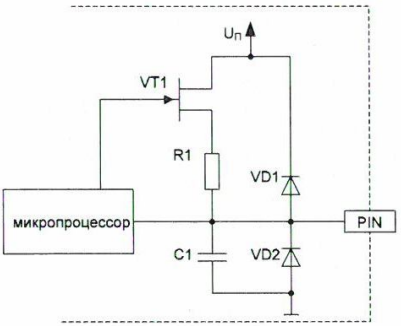



Рис. 2. Структура реализации вывода портов микроконтроллера

Подтягивающий резистор имеет высокое сопротивление, измеряемое десятками кОм, и не оказывает влияния на подключаемые к контроллеру сигналы. Через резистор R1 к порту ввода/вывода подключается напряжение питания с помощью транзистора VT1. Если при этом к выводу микросхемы контроллера не подключена внешняя цепь и вывод «висит в воздухе», то на микропроцессор через подтягивающий резистор R1 подается сигнал логической «1». Это надежно защищает вывод микроконтроллера от воздействия внешних помех.

Для инициализации порта на ввод информации и подключения подтягивающих резисторов необходимо:

– задать в регистре направления DDR работу порта на ввод информации установкой нулевых значений в его разрядах;

– включить подтягивающие резисторы порта ввода/вывода установкой сигналов логической «1» в разрядах регистра PORTX.

Считывание данных с порта в данном режиме осуществляется путем опроса регистра состояния PIN.

Пример инициализации порта A на «ввод данных»:

```
ldi r16,0x00  ;в РОН r16 записывается число 0000 0000
out DDRA,r16  ;командой out значение r16 посылается в DDRA
ldi r16,0xFF  ;в РОН записывается число 1111 1111
out PORTA,r16 ;командой out значение r16 посылается в PORTA
in r16, PINA  ;командой in вводится значение PINA в РОН r16
```

При инициализации порта на вывод информации код, выдаваемый микропроцессором, поступает непосредственно на вывод микросхемы контроллера. Подтягивающие резисторы при этом должны быть отключены.

Для инициализации порта на вывод информации необходимо:

– задать в регистре направления DDR работу порта на вывод информации установкой его разрядах сигналов логической «1»;

– вывести необходимую информацию на порт записью в регистр данных PORT необходимых значений.

Пример инициализации порта A на «вывод данных»:

```
ldi r16,0xFF   ;в РОН r16 записывается число 1111 1111
out DDRA,r16   ;командой out значение r16 посылается в DDRA
ldi r16,0x35   ;в r16 записывается любое число, например, 0x35
out PORTA,r16  ;командой out значение r16 посылается в PORTA
```

**Пример №1.** Написать программу, осуществляющую сложение двух младших и двух старших бит порта C с выводом результата на порт D.

```
;-----------------------------------------------------------------
;Программа для сложения двух двухбитных чисел A (биты PC7:PC6) и B
(биты PC1:PC0) с последующим выводом результата на PORTD
;Входы: PINC7:PINC6 и PINC1:PINC0
;Выходы: PORTD
.include "m8535def.inc"  ;Подключение библиотеки ATmega8535
.cseg                    ;Начало сегмента кода
.org 0
reset:                   ;Инициализация портов ввода/вывода
     ldi r16,0xFF        ;Установка PORTD на вывод информации: r16←0xFF
     out DDRD,r16        ;PORTD←r16
     clr r16             ;Установка PORTC на ввод информации:  r16←0
     out DDRC,r16        ; PORTD←r16
main:
     in r16,PINC         ;ввод данных из порта C в регистр: r16←PINC
     mov r17,r16         ;Копирование результата r17←r16
     andi r16,0x03       ;наложение маски: обнуление всех бит,кроме 0 и 1
     andi r17,0xC0       ;наложение маски: Обнуление всех бит,кроме 6 и 7
     lsr r17             ;Логический сдвиг r17 вправо на 6 бит
     lsr r17
     lsr r17
     lsr r17
     lsr r17
     lsr r17
     add r16,r17         ;Сложение r16 и r17: r16←r16+r17
     out PORTD,r16       ;Вывод результата в порт D: PORTD←r16
     rjmp main           ;возврат на main
;-----------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. Подключение стандартной библиотеки контроллера. Строка

```
.include "m8535def.inc"
```

подключает библиотеку контроллера Atmega8535, которая содержит всю необходимую информацию о контроллере, а именно список регистров с их адресами, объемы памяти, список периферийных устройств и другие особенности.

2. Выбор сегмента кода и адреса начала написания программы. Строки

```
.cseg
```

```
.org 0
```

устанавливается начало сегмента кода на нулевой адрес.

3. Инициализация портов ввода/вывода. Рассмотрим следуюшие строки программы:

```
ldi r16,0xFF
out DDRD,r16
clr r16
out DDRC,r16
```

Эти команды инициализируют порт C на ввод данных, а порт D – на вывод. Поскольку порт D необходимо инициализировать на вывод, в регистр DDRD требуется записать 0xFF (1111 1111 в двоичном коде). Для этого сначала в регистр общего назначения r16 записывается число 0xFF (инструкция `'ldi'`), а затем содержимое этого регистра переписывается в регистр DDRD (инструкция `'out'`). Аналогичная операция производится с портом C.

4. Ввод данных и запись в регистры общего назначения. В главной программе согласно заданию необходимо опросить состояние регистра PINC и выделить два числа: в первом хранятся два младших бита, во втором – два старших бита. Для этого вначале выполняется ввод всего регистра PINC регистры r16 и r17 (инструкции `'in'` и `'mov'`). Далее для корректного считывания чисел необходимо наложить, так называемые, маски на оба числа: в первом числе обнулить все биты кроме двух младших, во втором – все биты кроме двух старших. Это осуществляется с помощью команды «побитовое умножение» (инструкция `'andi'`): содержимое регистра r16 перемножается с константой 0x03 (0000 0011), а регистра r17 – с константой 0xC0 (1100 0000). Эти операции выполняются строками:

```
in r16,PINC
mov r17,r16
andi r16,0x03
andi r17,0xC0
```

5. Приведение переменных к одному весовому коэффициенту. Полученные значения переменных в регистрах r16 и r17 имеют разные весовые коэффициента необходимо преобразовать число XX00 0000, содержащееся в r17, в число вида 0000 00XX. Для этого в программе используется 6 раз одна и таже инструкция логического сдвига вправо `lsr` содержимого регистра r17 на 1 бит.

**6.** После выполнения операции сдвига r17 можно произвести сложение r16 и r17 и вывод результата сложения на PORTD:

```
add r16,r17
out PORTD,r16
```

**7.** В конце программы ее необходимо «зациклить», т.е. вернуться на метку «main» для ее повторного выполнения:

```
rjmp main
```

**Пример 2.** Реализовать на микроконтроллере расчет логического уравнения $D = A \bullet (\overline{A} \oplus B + C)$, где $A$, $B$, $C$ логические сигналы, поступающие на входы,

например, PA0 (0 бит порта A), PA1 (1 бит порта A), PA2 (2 бит порта A), а *D* – результат решения логического уравнения, который выводится на 0 бит порта D.

```
;---------------------------------------------------------------------
;Программа для решения логического уравнения D=A*(NA*B+C)
;Входы:   A=PINA0
;         B=PINA1
;         C=PINA2
;Выход:   D=PORTD0

.include "m8535def.inc"  ;Подключение библиотеки Atmega8535
.def A=r20               ;Объявление переменных и присвоение их имен
.def B=r21               ;регистрам общего назначения r20…r23
.def C=r22
.def D=r23
.cseg                    ;начало сегмента кода
.org 0

reset:                   ;инициализация портов ввода/вывода
     ldi r16,0x01
     out DDRD,r16        ;PORTD0 - на вывод информации
     clr r16
     out DDRA,r16        ;PINA0 …PINA2 - на ввод информации
main:
     in r16,PINA         ;Ввод значения PINA в РОН r16
     mov A,r16           ;Копирование r16 в регистры A, B, C
     mov B,r16
     mov C,r16
     andi A,0x01         ;Выделение 0 бита числа A
     andi B,0x02         ;Выделение 1 бита числа B
     andi C,0x04         ;Выделение 2 бита числа C
     lsr B               ;Сдвиг B вправо на 1 бит
     lsr C
     lsr C               ;Сдвиг C вправо на 2 бита
     mov r16,A           ;Копирование A в РОН r16
     com r16             ;Инверсия содержимого r16
     andi r16,0x01       ;Формирование числа NA
     and r16,B           ;Расчет r16=NA*B
     or r16,C            ;Расчет r16=r16+C
     and r16,A           ;Расчет r16=r16*A
     out PORTD,r16       ;Вывод результата на PORTD0
     rjmp main           ;Возврат на main
;---------------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. Инициализация контроллера и переменных. Вначале выполняются инициализация микроконтроллера и присвоение имен A, B, C, D регистрам общего назначения r20 … r23. Для этого используется директива '.def A=r20':

```
.include "m8535def.inc"
.def A=r20
.def B=r21
.def C=r22
.def D=r23
.cseg
.org 0
```

2. Инициализация портов ввода/вывода. В данной программе производится аналогично предыдущему примеру с следующими отличиями: порт D инициализирован на вывод только 1 младшего бита, порт A – полностью на ввод данных.

3. Ввод данных. В основной программе сначала в регистр r16 вводится значение порта A, затем это значение переписывается в регистры A, B и C, и далее «наложение» маски на эти регистры: в этих регистрах выполняется выделение только отдельных битов: в A – 0 бит, B – 1 бит, C – 2 бит:

```
in r16,PINA
mov A,r16
mov B,r16
mov C,r16
andi A,0x01
andi B,0x02
andi C,0x04
```

4. Выравнивание весовых коэффициентов переменных. Биты в регистрах A, B, C имеют разные весовые коэффициенты, для их выравнивания выполняется смещение значений регистров B и C вправо на 1 и 2 бита соответсвенно.

5. Формирование инверсного значения переменной. Для формирования инверсного значения регистра A это значение копируется в регистр r16, далее оно инвертируется (инструкция `'com'`) и опять выделяется только младший бит:

```
mov r16,A
com r16
andi r16,0x01
```

6. Вычисление логического выражения и вывод данных. Логические операции выполняются согласно уравнению $D = A \cdot (\overline{A} \cdot B + C)$, далее результат выводится на индикацию в порт D и программа зацикливается:

```
and r16,B
or r16,C
and r16,A
out PORTD,r16
rjmp main
```

**Задание на выполнение**

1. На базе примера №1 составить программу для вычисления:
а) суммы двух 3-разрядных чисел;
б) суммы двух 8-разрядных чисел;
в) разности двух 2-разрядных чисел;
г) разности двух 4-разрядных чисел.

2. Разработать логическую систему автоматизации - составить программу для своего варианта по реализации заданного логического уравнения. Дома проверить работу программы на AVR-Studio и используя описание инструкций к микроконтроллеру AVR (Instruction Set) составить для своей программы таблицу в машинных кодах – для каждой инструкции привести: мнемокод на Ассемблере,

двоичной форме, машинный код в шестнадцатеричном коде, комментарий по инструкции.

В лаборатории ввести программу и проверить ее работоспособность на контроллере и проверить машинные коды.

Варианты задания приведены в таблице.

| | | |
|---|---|---|
| 1. PC3=$\overline{(PA0+PA1)}$·PA3 | 11. PD5=$\overline{PA1+PA2}$ ⊕ PD0 | 21. PC3=PD0·$\overline{PD2 \oplus PD3}$ |
| 2. PD4=PB1⊕ PB2+PB3·PB4 | 12. PD3=PB7+$\overline{PB6 \cdot PB5}$ | 22. PA2=(PC6+PC7)·PB0⊕PB1 |
| 3. PD0=$\overline{PA1}$+(PA2⊕ PA3·PA4) | 13. PC3=$\overline{(PA0+PB0)}$+PD7 | 23. PB1=PA2+$\overline{PC3 \oplus PD3}$ |
| 4. PD7=$\overline{PA7 \oplus PA6 \cdot PA5}$ | 14. PC0=PD7⊕ $\overline{PD1+PD2}$ | 24. PA1=PC7+PC0⊕PB0·PD0 |
| 5. PC0=$\overline{(PA3 \cdot PA4)}$+PA5 | 15. PA0=PC0⊕$\overline{PC1}$·PA7 | 25. PB0=$\overline{PD0}$⊕$\overline{PD1}$+$\overline{PD2}$ |
| 6. PC7=PD2⊕$\overline{PD3+PD4}$ | 16. PC1=PC2+PC3·PC4 | 26. PC2=$\overline{PA0}$·PA1⊕$\overline{PA2}$ |
| 7. PD1=(PC0·PC1)⊕(PB0·PB1) | 17. PA0=PC0·PC1+PD0⊕PD1 | 27. PD3=(PA0·$\overline{PA1}$)+PA2 |
| 8. PC1=PB2+$\overline{PB3 \cdot PB4}$ | 18. PD7=$\overline{PA0 \cdot PB1}$⊕PC2 | 28. PD0=PC0·$\overline{PA0}$+PB0 |
| 9. PA1=PC0⊕$\overline{PC1}$+$\overline{PC6}$⊕PC7 | 19. PC1=$\overline{PA0 \cdot PA1+PA3}$ | 29. PA0=PA1+PA2·PA3 |
| 10. PD7=$\overline{PD0 \cdot PD1}$+PD2 | 20. PA7=PB0⊕PB2 ·$\overline{PB1}$+PB3 | 30. PA1=$\overline{PC0}$⊕PC1·PD0⊕PD1 |

3. Составить программу и проверить ее работоспособность на микроконтроллере, в которой число набранное в порт А изменяется (используя битовые операции) и выводится в порт С:

а) увеличивается в 2 раза;

б) уменьшается в 4 раза;

в) выводится в обратном коде;

г) выводится в дополнительном коде.

4. Составить программу и проверить ее работоспособность на микроконтроллере, в которой вводится два четырехразядных числа А (биты PA0…PA3 и В (биты PB0…PB3), результат выводится в порт С и выполняется следующая операция:

а) ариметическая сумма чисел;

б) поразрядная логическая сумма;

в) арифметическое произведение чисел;

г) поразрядное логическое произведение;

д) арифметическая разность.

**Контрольные вопросы**

1. Сколько портов имеет микроконтроллер ATmega8535?

2. Какие регистры определяют режим работы порта? Поясните их назначение.

3. Определите регистры работы порта С если известно, что 2 бита порта работаю на ввод данных, остальные на вывод.

4. Чем отличается в Ассемблере директива от инструкции?

5. Для каких целей используется директивы ‘.def’, ‘.cseg’, ‘.org’?

6. Какие инструкции по выполнению логических операций вы знаете?

7. Как наложить маску на считываемое значение регистра состояния?

## Работа № 3. Специальный регистр состояния SREG

### Цель работы

Освоить теоретический и практический материал по назначению и использованию регистра SREG. Применить приобретенные навыки при составлении программ.

### Программа работы

1. Изучить необходимый теоретический материал о регистре SREG.
2. Изучить представленные в лабораторной работе примеры программ по использованию стека и регистра SREG.
3. Написать и отладить собственную программу по реализации программной задержки.

### Пояснения к работе

В микроконтроллерах фирмы Atmel присутствует так называемый регистр состояния **SREG** (Status REGister).

Регистр состояния содержит информацию о результатах наиболее часто извлекаемых арифметических операциях. Эта информация может быть использована при написании программы для создания переходов, циклов, сравнения чисел и т.д.

Разряды регистра **SREG** называются флагами. Всего этих флагов восемь:

| № бита | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Флаг | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |

Назначение разрядов регистра **SREG**:

**Бит 0: C – флаг переноса.** Флаг переноса индицирует появление переноса при выполнении арифметических или логических операций. Флаг переноса незаменим при совершении операций с n-байтными числами.

**Пример 1.** Прибавить к числу 0xFE произвольное 8-разрядное число. Результат – шестнадцатиразрядное число.

```
clr r18          ;очистка регистра r18
ldi r16,0xFE     ;запись в r16 значения 0xFE
ldi r17,0x05     ;запись любого числа в r17
mov r20,r16      ;копирование r16 в r20
add r20,r17      ;сложение r20 и r17
adc r21,r18      ;если при этом возникает перенос, то в r21
                 ;прибавляется 0x01. Таким образом, результат
                 ;формируется в r21:r20
```

**Бит 1: Z – флаг нуля.** Этот флаг индицирует нулевой результат при выполнении арифметических или логических операций. Этот флаг может быть полезен при выполнении такой операции, как сравнение.

**Пример 2.** Осуществить сравнение двух чисел, задаваемых на портах A и C. При их равенстве выдать сигнал логической «1» на PORTD0.

```
main:
    in r16,PINA      ;ввод первого числа
    in r17,PINC      ;ввод второго числа
    cp r16,r17       ;сравниваются значения регистров r16 и r17
```

```
                    ;(проведением операции вычитания r16-r17). При
                    ;их равенстве устанавливается в '1' флаг Z
    breq m1         ;при Z=1 осуществляется переход на метку m1
    cbi PORTD,0     ;иначе бит PORTD0 очищается
    rjmp main       ;и идет возврат на main
m1:
    sbi PORTD,0     ;при переходе на m1 устанавливается бит PORTD0
    rjmp main       ;и идет возврат на main
```

**Бит 2: N – флаг отрицательного значения.** Этот флаг индицирует наличие отрицательного числа как результата выполнения арифметических или логических операций. В микроконтроллерах отрицательное число получается из положительного путем дополнения до двух. Для этого все разряды числа инвертируются, а потом к числу прибавляется единица младшего разряда. Так, если 8-разрядное число +1 записывается в двоичном коде как 0000 0001, то отрицательное число –1 записывается как 1111 1111. При этом старший разряд (бит 7) числа определяет его знак.

Флаг отрицательного значения может быть полезен при совершении операции сравнения двух чисел.

**Пример 3.** Осуществить сравнение двух чисел A и B, задаваемых на портах A и C. Если число, A≥B, то на PORTD0 подается логическая «1». Иначе на PORTD0 подается логический «0».

```
main:
    in r16,PINA     ;ввод числа A
    in r17,PINC     ;ввод числа B
    cp r16,r17      ;сравнение чисел A и B вычитанием A-B
    brmi m1         ;если при этом установлен флаг N, то
                    ;осуществляется переход на m1
    sbi PORTD,0     ;иначе устанавливается разряд PORTD0
    rjmp main       ;и идет возврат на main
m1:
    cbi PORTD,0     ;по метке m1 очищается разряд PORTD0
    rjmp main       ;и идет возврат на main
```

**Бит 3: V – флаг переполнения.** Этот бит устанавливается при переполнении регистра во время совершения операций над числами. Так, если в регистре было записано число 1111 1111 и к нему прибавили +2, то происходит переполнение. Результатом такой операции будет число 0000 0001 и установленный флаг V.

Необходимо различать флаг переполнения V и флаг переноса C. Флаг переполнения предназначен, в первую очередь, для работы с дополнительным кодом. Как было сказано выше, в дополнительном коде старший разряд определяет знак числа, а значение числа ограничивается 7 разрядами.

Если при сложении двух положительных чисел в дополнительном коде происходит изменение старшего бита, то это означает, что произошло переполнение числа, хотя переноса не происходит (флаг C не меняется). Например, при сложении чисел:

```
      0110 0010
+     0110 1111
=     1101 0001
```

флаги устанавлвиются следующим образом:

– поскольку произошло изменение старшего разряда, то флаг переполнения V устанавливается: V=1;

– так как не произошло переполнение разрядов регистра, то флаг переноса C не возникает: C=0.

Если при сложении двух отрицательных чисел в дополнительном коде происходит изменение старшего бита, то это означает, что также произошло переполнение числа, при этом может возникнуть и флаг переноса. Например, при сложении чисел:

```
          1001 1110
+         1001 0001
=     1 0010 1111
```

меняются как старший разряд, так и появляется бит переполнения:

– поскольку произошло изменение старшего разряда, устанавливается флаг переполнения V;

– поскольку произошло переполнение разрядов регистра, устанавливается и флаг переноса C.

**Пример 4.** Используя инструкцию 'add', сложить два положительных числа в дополнительном коде. Если происходит переполнение результата, то результат необходимо ограничить максимальным или минимальным числом. Результат выводится на PORTD.

```
main:
    in r16,PINA      ;ввод первого числа A
    in r17,PINC      ;ввод второго числа B
    add r16,r17      ;сложение A и B
    brvc m1          ;если нет переполнения (флаг V=0), то
                     ;осуществляется переход на m1
    brcs m2          ;иначе проверка флага переноса. Если флаг C=1,
                     ;то осуществляется переход на m2
    ldi r16,0x7F     ;если переноса нет, то результат -
                     ;максимальное число +127 (0111 1111)
    rjmp m1          ;далее - переход на m1
m2:
    ldi r16,0x80     ;если флаг переноса C=1, то результат -
                     ;минимальное число -128 (1000 0000).
m1:
    out PORTD,r16    ;по метке m1 - вывод результата на PORTD
    rjmp main        ; и возврат на main
```

**Бит 4: S – флаг знака.** Этот бит всегда является результатом совершения операции исключающего ИЛИ между флагом отрицательного числа N и флагом переполнения V. Так, если при совершении операции не происходит переполнения (флаг V=0), то знак определяется флагом N. Если же происходит переполнение (V=1), то флаг знака принимает инвертированное значение флага N.

**Бит 5: H – флаг половинного переноса.** Половинным переносом называется процесс переноса из первой половины байта во вторую. Так, если в

байте была комбинация 0000 1111 и к нему прибавили +1, то происходит половинный перенос, а именно 0001 0000, при этом формируется флаг H.

**Пример 5.** Организовать бегущий огонь в младшем полубайте PORTC.

```
     ldi r16,0xFF   ;в r16 записывается значение 0xFF
     out DDRC,r16   ;порт C инициализируется на вывод информации
     ldi r16,0x01   ;в r16 записывается единица младшего разряда
main:
     out PORTC,r16  ;значение r16 выводится на PORTC
     lsl r16        ;r16 сдвигается влево на один разряд
     brhc main      ;если флаг H снят, то переход на main
     ldi r16,0x01   ;иначе в r16 записывается 0x01
     rjmp main      ;и осуществляется переход на main
```

**Бит 6: T – хранение бита информации.** Инструкции копирования бит используют бит T регистра SREG. При копировании бита из какого-либо регистра он сохраняется в бите T регистра SREG, а затем извлекается из него при копировании в какой-либо другой регистр.

**Пример 6.** Скопировать из регистра r16 в регистр r19 пятый бит, не изменяя содержимое регистра r16.

Без использования флага T:

```
in r16,PINA     ;запись в r16 любого числа
mov r17,r16     ;копирование r16 в любой свободный РОН (r17)
andi r17,0x20   ;выделение 5-го бита в r17
or r19,r17      ;логическое ИЛИ r19 и r17
```

С использованием флага T:

```
in r16,PINA     ;запись в r16 любого числа
bst r16,5       ;копирование 5-го бита в SREG
bld r19,5       ;копирование содержимого флага T в 5-й бит r19
```

**Бит 7: I – общее разрешение прерываний.** Назначение этого флага будет пояснено в работах с использованием прерываний таймеров.

**Пример 7.** Произвести вычитание двух чисел. Первое число задается младшим полубайтом порта A и имеет формат:

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| A= | 0 | 0 | 0 | 0 | PA3 | PA2 | PA1 | PA0 |

второе число задается старшим полубайтом порта A и имеет формат:

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| B= | PA7 | PA6 | PA5 | PA4 | 0 | 0 | 0 | 0 |

Результат должен выводиться в следующем виде:

– младшие 4 разряда порта C – модуль разности (A-B);

– старший бит порта C – знак результата.

```
//---------------------------------------------------------------
.include "m8535def.inc"  ;подключение библиотеки ATmega8535
.cseg                    ;начало сегмента кода
.org 0
    ldi r16,0xFF         ;инициализация портов ввода/вывода
    out PORTA,r16        ;порт A - на ввод информации
    out DDRC,r16         ;порт C - на вывод информации
```

```
main:
    in r16,PINA          ;ввод данных PINA в РОН r16
    mov r17,r16          ;копирование r16 в r17
    andi r16,0x0F        ;выделение числа A
    andi r17,0xF0        ;выделение числа B
    clr r18              ;очистка РОН r18
m1:
    lsr r17              ;сдвиг r17 на 4 разряда вправо
    inc r18
    cpi r18,0x04
    brne m1
    sub r16,r17          ;выполнение вычитания r16-r17 (A-B)
    brmi m2              ;если в результате установлен флаг N, то
                         ;осуществляется переход на метку m2
    rjmp m3              ;иначе - переход на метку m3
m2:                      ;выделение модуля результата
    com r16              ;инверсия результата
    inc r16              ;увеличение результата на +1
    ori r16,0x80         ;прибавление к результату сигнала «знак»
m3:
    out PORTC,r16        ;вывод результата на PORTC
    rjmp main            ;и возврат на main
//-------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. Инициализация портов ввода/вывода: порт A – на ввод, порт C – на вывод информации:

```
ldi r16,0xFF
out PORTA,r16
out DDRC,r16
```

2. В начале основного цикла производится опрос регистра PINA и приведение чисел к одному виду:

– результат заносится в регистр r16 (`in r16,PINA`) и r17 (`mov r17,r16`);

– затем в регистре r17 осуществляется сдвиг числа на 4 разряда вправо для того, чтобы преобразовать его из числа вида «XXXX 0000» в число «0000 XXXX»:

```
main:
    in r16,PINA
    mov r17,r16
    andi r16,0x0F
    andi r17,0xF0
    clr r18
m1:
    lsr r17
    inc r18
    cpi r18,0x04
    brne m1
```

3. Операция вычитания, после выполнения которой флаги в регистре SREG выставятся определенным образом:

```
sub r16,r17
```

4. Для обработки результатов вычитания необходимо проверить флаг отрицательного значения N:

– если он отсутствует, то результат – положительное число и его сразу можно выводить на индикацию;

– если же флаг N установлен, то результат – отрицательное число, и перед выводом на индикацию необходимо выделить его модуль:

```
brmi m2
rjmp m3
```

Команда `brmi m2` осуществляет проверку флага N. Если он установлен, осуществляется переход на метку m2, при переходе на которую осуществляется выделение модуля числа. Если флаг N не установлен, выполняется следующая за командой директива `rjmp m3`. По этой команде выполняется переход на метку m3, по которой осуществляется вывод результата на индикацию.

5. При переходе на метку m2 результат сначала инвертируется (`com r16`), а затем к нему прибавляется +1 (`inc r16`). Таким образом, осуществляется выделение из дополнительного кода модуля результата вычитания:

```
m2:
     com r16
     inc r16
     ori r16,0x80
m3:
     out PORTC,r16
     rjmp main
```

Командой `ori r16,0x80` осуществляется установка сигнала «Знак» в старшем разряде регистра r16, который затем выдается на индикацию на PORTC `out PORTC,r16`. После этого основной цикл программы замыкается.

### Задание на выполнение

1. Используя логические операции и биты регистра состояния реализовать на микроконтроллере схему:

– полусумматора;

– полного одноразрядного сумматора.

2. Используя биты регистра состояния вычислить модуль разности двух 5-х разрядных чисел.

3. Используя биты регистра состояния реализовать компаратор: сравниваются два 4-х разрядных числа (порты A и B): если равны включается 0 бит порта D, если первое число больше – 1 бит, если второе – 2 бит.

4. Используя бит хранения информации регистра состояния реализовать:

– RS-триггер;

– D-триггер;

– T-триггер.

### Контрольные вопросы

1. Для чего предназначен регистр состояния?

2. Перечислите биты регистра состояния и их назначение.

3. При выполнении каких операций изменяется для флага нуля? Флаг отрицательного значения? Флаг знака?

4. При суммировании двух 4-х разрядных чисел какие биты регистра сосятония могут изменить свое значение? Двух восьмиразрядных?

5. Какой бит регистра состояния отвечает за разрешение работы всех прерываний?

6. Как связаны друг с другом флаги: знак, отрицательное значение, переполнение?

7. Для каких целей используется бит T?

## Работа № 4. Стек. Реализация программной задержки

### Цель работы

Освоить теоретический и практический материал по использованию стека. Ознакомиться с назначением стека, порядком его определения в программе и последовательностью исполнения подпрограмм. Применить приобретенные навыки при написании программы задержки времени.

### Программа работы

1. Изучить необходимый теоретический материал о стеке.
2. Изучить представленные в лабораторной работе примеры программ по использованию стека.
3. Написать и отладить собственную программу по реализации программной задержки.

### Пояснения к работе

При написании программ для микроконтроллеров Atmel и других производителей с использованием ассемблера этих микроконтроллеров необходимо иметь представление о реализации стека. Без теоретических знаний в этой области совершенно невозможно написать даже относительно несложную программу.

#### Указатель стека

Для того, чтобы разъяснить понятие стека, необходимо внести ясность в понимание структуры программы микроконтроллера.

При выполнении программы извлечение и выполнение директив (команд) ведется последовательно. То есть сначала исполняется первая в списке команда, потом – вторая, третья и т.д.

Все микроконтроллеры и микропроцессоры имеют так называемый счетчик команд (Program Counter). В нем хранится адрес текущей выполняемой команды. При запуске программы счетчик команд равен 0, затем он инкрементируется по мере выполнения команд на 1 или на 2, в зависимости от длины команды (Приложение А). Если по какой-либо причине в программе осуществляется переход в другое место программы (безусловный или условный переход), то содержимое счетчика команд скачком изменяется, указывая на новый адрес вызванной команды.

Если при выполнении программы возникает необходимость осуществить переход на какую-либо подпрограмму, выполнить ее, а затем вернуться на старое место и продолжить выполнение программы, становится необходимым запоминать адрес возврата. Для этого и предназначен указатель стека.

*Указатель стека* – это специальный регистр, представляющий собой буфер, в котором реализован принцип «Last In – First Out» LIFO (последним пришел – первым уйдешь). Этот принцип означает, что адрес перехода, который был записан в стек самым последним, будет считан самым первым. Таким образом, если в программе последовательно выполняются несколько переходов в

подпрограммы, никогда не возникнет путаницы с порядком возврата из подпрограмм обратно.

В микроконтроллере Atmega8535 стек реализован в двух 8-разрядных регистрах SPH, SPL, которые вместе образуют 16-разрядный регистр SPH:SPL.

Принцип работы стека поясняется на рис. 1.

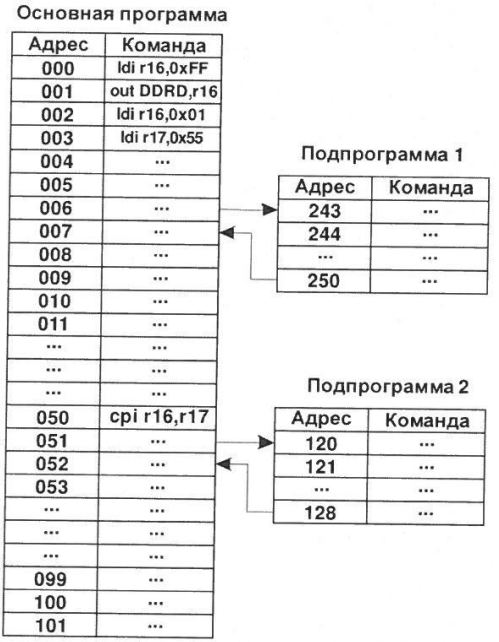



Рис. 1. Принцип работы указателя стека

При работе программы каждая ее команда имеет адрес, присваиваемый счетчиком команд (рис. 1). Если при выполнении программы возникает необходимость сделать переход на какую-либо подпрограмму, расположенную по какому-либо адресу, например, как показано на рис. 1, по адресу 243, необходимо иметь информацию, на какой адрес необходимо вернуться после выполнения подпрограммы. Поэтому при переходе, например, с адреса 006 на адрес 243 в указатель стека записывается адрес возврата, то есть 007.

При вызове следующей подпрограммы, расположенной по адресу 120, в указатель стека записывается также адрес возврата. В данном случае – 052.

Адрес возврата записывается в вершину стека.

*Вершиной стека* называется адрес ОЗУ процессора, в которую ведется запись адреса возврата.

Адрес вершины стека программист обязан указать самостоятельно, при этом общепринято, что вершина стека находится в конце ОЗУ процессора. Это

делается для того, чтобы область программы случайно не перекрылась с областью стека, так как если это произойдет, то адрес возврата, записанный в область стека, будет потерян.

Возможен случай, когда при выполнении подпрограммы происходит запрос на следующий переход на другую подпрограмму (рис. 2).

Когда происходит переход из основной программы в подпрограмму 1, в вершину стека записывается адрес возврата 007. Если далее из подпрограммы 1 происходит переход в подпрограмму 2, то в стек записывается адрес 245 возврата в подпрограмму 1.

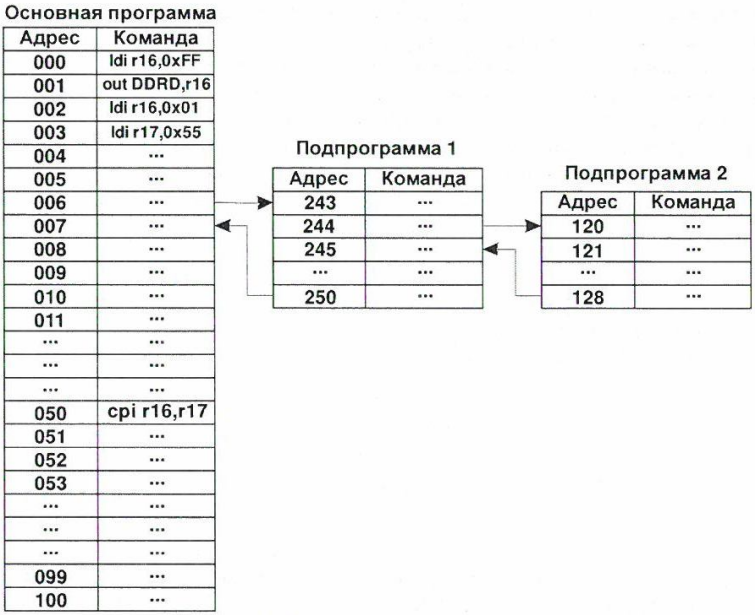



Рис. 2. Принцип заполнения стека

При окончании подпрограммы 2 первым будет прочитан адрес 245, а затем при окончании подпрограммы 1 – адрес 007. Таким образом, благодаря принципу LIFO полностью исключается неправильный переход в неправильное место программы.

В микроконтроллере Atmega8535 указатель стека состоит из двух 8-разрядных регистров SPH и SPL, которые вместе составляют 16-разрядный регистр SPH:SPL. Применение 16-разрядного регистра объясняется объемом памяти программ микроконтроллера – он составляет 8кБ (4096 слов памяти программ) и для обращения к конкретной ячейке памяти необходимо указания 2 байт адреса.

При написании программ вызов подпрограмм обычно осуществляется командой `rcall`, а возврат из подпрограммы осуществляется по директиве `ret`.

При вызове директивы `rcall` в регистр SPH:SPL записывается адрес ячейки памяти, в которую будет сохранен адрес возврата из подпрограммы, и адрес возврата автоматически записывается в стек, а при вызове директивы `ret` происходит чтение из стека по указанному в регистре SPH:SPL адресу.

**Программная задержка**

Одним из наиболее наглядных применений указателя стека является реализация программной задержки времени.

В микроконтроллере ATmega8535 присутствуют три таймера, с помощью которых можно достаточно просто реализовать любую временную задержку. Однако простейшую программную задержку необходимо уметь реализовывать без использования дополнительных аппаратных средств микроконтроллера.

Суть программной задержки состоит в том, чтобы заставить процессор выполнять циклически одно и то же действие, например, инкремент какого-либо числа от нуля до максимума, с дальнейшим переходом при окончании выполнения этого действия к дальнейшему выполнению программы.

Ввиду того, что процессор производит операции с большой скоростью, для сколько-нибудь ощутимой задержки приходится производить операции с большими числами.

Пусть нам задан некоторый элемент программы. Рассчитаем примерное время исполнения этого элемента (в комментариях прописано количество тактов процессора для выполнения данной инструкции):

```
//-------------------------------------------------------------
     clr r16         ; 1 такт процессора
m1:                  ; не выполняется, нет задержки
     inc r16         ; 1 такт процессора
     cpi r16,0x20    ; 1 такт процессора
     brne m1         ; 2 такта процессора, если условие выполняется
                     ; (переход к метке m1) и 1 такт процессора,
                     ; если условие не выполняется (выход из цикла)
//-------------------------------------------------------------
```

В программе значение регистра РОН r16 увеличивается на 1, пока значение в r16 не достигнет значения 0x20 (десятичное число 32). При этом постоянно идет возврат на метку m1, и только после того, как значение, записанное в r16, достигает значения 0x20, разрешается выполнение последующих команд программы. Таким образом, в программе реализован цикл по переменной r16, во время исполнения которого она «ничего не делает», т.е. как бы «зависает» на некоторое время. Это процесс и называется *программной задержкой*.

Зная время выполнения команд (прил. А) можно рассчитать время исполнения любой части программы. Расччитаем время исполнения рассмотренной части программы.

При попадании в цикл суммарное количество тактов процессора будет N=4, то есть при подсчете чисел от 1 до 32 суммарное количество тактов будет равняться $N=4 \cdot 32=128$. После этого необходимо вычесть один цикл, который остался неучтенным при выполнении ложного условия инструкции `brne m1` в

конце подсчета r16 и прибавить один такт на выполнение команды `clr r16`. Итого, суммарное количество тактов равно: N=4·32–1+1=128.

Если умножить полученное число на время выполнения одного такта процессора, которое обратно частоте колебаний кварцевого резонатора $f_{OSC}$ (в нашем случае – 8 МГц), то можно найти время задержки $T_З$:

$$T_З = N \cdot \frac{1}{f_{osc}} = 128 \cdot \frac{1}{8 \cdot 10^6} = 16{,}0 мкс$$

Для реализации более существенных временных задержек приходится иметь дело с большими числами или вложенными циклами. Пример программной задержки с применением вложенных циклов приведен далее.

**Пример.** Написать программу бегущего огня на PORTC. При этом время задержки между переключениями разрядов порта задается с помощью программной задержки.

```
;---------------------------------------------------------------------
.include"m8535def.inc"   ;стандартная библиотека Atmega 8535
.cseg                    ;начало сегмента кода
.org 0
   ldi r16,$5f           ;размещение вершины стека по адресу
   ldi r17,$02           ;старшей ячейки ОЗУ
   out spl,r16
   out sph,r17
   ldi r16,0xFF          ;инициализация портов ввода/вывода
   out DDRC,r16          ;PORTC - на вывод
   ldi r16,0x01          ;занесение в РОН r16 числа 1
main:                    ;начало главной программы
   out PORTC,r16         ;вывод на PORTC значения r16
   in r17,SREG           ;сохранение регистра SREG
   rcall delay1          ;вызов подпрограммы задержки
   out SREG,r17          ;восстановление SREG после возврата
   rol r16               ;цикличный поворот r16 вправо
   rjmp main             ;возврат на main
delay1:        ;подпрограмма №1
   clr r18               ;очистка регистра r18
 met1:
   rcall delay2          ;переход на 2 подпрограмму - delay2
   inc r18               ;инкремент r18
   cpi r18,0x10          ;сравнение значения r18 с числом 16
   brne met1             ;если значение в r18≠15, то переход на met1
   ret                   ;иначе - возврат в основную программу
delay2:        ;подпрограмма №2
   clr r19               ;очистка регистра r19
 met2:
   rcall delay3          ;переход на 3 подпрограмму - delay3
   inc r19               ;инкремент r19
   cpi r19,0xFA          ;сравнение значения r19 с числом 250
   brne met2             ;если значение в r19≠255, то переход на met2
   ret                   ;иначе - возврат в подпрограмму №1
delay3:        ;подпрограмма №3
```

```
   clr r20           ;очистка регистра r20
  met3:
   inc r20           ;инкремент r20
   cpi r20,0xFA      ;сравнение значения r20 с числом 250
   brne met3         ;если значение в r20≠255, то переход на met3
   ret               ;иначе - возврат в подпрограмму №2
;-----------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. В начале программы вершина стека помещается по адресу последней ячейки ОЗУ:

```
ldi r16,$5F
ldi r17,$02
out spl,r16
out sph,r17
```

2. Инициализируется порт ввода/вывода С, а регистру r16 присваиваются значение r16=0x01 (начальная позиция бегущего огня):

```
ldi r16,0xFF
out DDRC,r16
ldi r16,0x01
```

3. В главной программе значение регистра r16 выводится на индикацию на PORTC, после чего вызывается подпрограмма задержки (`rcall delay`), по прошествии которой значение r16 сдвигается на один разряд вправо (`rol r16`). Далее программа замыкается:

```
main:
     out PORTC,r16
     in r17,SREG
     rcall delay
     out SREG,r17
     rol r16
     rjmp main
```

Здесь необходимо отметить, что перед вызовом подпрограммы задержки значение регистра SREG необходимо сохранить, так как иначе в результате выполнения подпрограммы этот регистр может изменить свое значение. После возврата из подпрограммы значение SREG восстанавливается.

4. Подпрограмма задержки реализована по принципу, рассмотренному выше. В подпрограмме реализованы вложенные циклы:

```
delay1:
     clr r18
     met1:
     rcall delay2
     inc r18
     cpi r18,0x10
     brne met1
     ret
delay2:
     clr r19
     met2:
     rcall delay3
     inc r19
     cpi r19,0xFA
     brne met2
```

```
    ret
delay3:
    clr r20
    met3:
    inc r20
    cpi r20,0xFA
    brne met3
    ret
```

Так, при входе в первый цикл, ограниченный меткой `met1` и условием `brne met1`, перед инкрементом регистра r18 происходит вызов подпрограммы `delay2`, в которой данная операция повторяется (выполняется второй цикл, вложенный в первый) и осуществляется переход на подпрограмму `delay3`, в которой выполняется третий цикл, вложенный во второй. Таким образом, сначала происходит выполнение цикла в подпрограмме `delay3` (инкремент регистра r20 по метке `met3`), потом – цикла в подпрограмме `delay2` (инкремент r19 по метке `met2`), после чего снова осуществляется переход на `delay1` (инкремент r18 по метке `met1`)и т.д.

**Дисассемблирование программы**

При оформлении отчета по проделанной работе необходимо произвести дисассемблирование написанной программы. Дисассемблирование представляет собой инструмент AVR Studio, благодаря которому можно увидеть работу программы, включая указатель стека и счетчик команд.

Дисассемблирование программы выполняется выбором в меню *View* AVR Studio пункта *Disassembler*. В этом случае на экране появляется программа, в которой указывается адрес каждой команды и необходимая служебная информация.

Одновременно на экран необходимо вывести окно контроля содержимого памяти *Toggle memory window* (для вызова набрать комбинацию Alt+0), в котором необходимо выбрать память данных (*Data*). В конце области этой памяти будет располагаться указатель стека.

В отчете по работе необходимо привести таблицу дисассемблера со значениями, записанными в стеке во время исполнения программы (ее первого прохода). В таблице необходимо указать:

– все инструкции и метки программы;

– адрес памяти всех инструкций;

– текущее значение указателя стека во время исполнения программы;

– значения всех ячеек ОЗУ, в которые записается стек, во время исполнения программы (первого прохода). Пример такой таблицы приведен в табл. 1.

Таблица 1: Дисассемблер программы со стеком

| Адрес памяти программ | Команда / метка | Указатель стека SPH:SPL | Стек (ячейки ОЗУ) | | |
|---|---|---|---|---|---|
| | | | 025A:025B | 025C:025D | 025E:025F |
| +00000000 | ldi r16,$5F | 00 00 | FF:FF | FF:FF | FF:FF |
| +00000001 | ldi r17,$02 | 00 00 | FF:FF | FF:FF | FF:FF |
| +00000002 | out spl,r16 | 00 5F | FF:FF | FF:FF | FF:FF |
| +00000003 | out sph,r17 | 02 5F | FF:FF | FF:FF | FF:FF |

| Адрес памяти программ | Команда / метка | Указатель стека SPH:SPL | Стек (ячейки ОЗУ) | | |
|---|---|---|---|---|---|
| | | | 025A:025B | 025C:025D | 025E:025F |
| +00000004 | ldi r16,0xFF | 02 5F | FF:FF | FF:FF | FF:FF |
| +00000005 | out DDRC,r16 | 02 5F | FF:FF | FF:FF | FF:FF |
| +00000006 | ldi r16,0x01 | 02 5F | FF:FF | FF:FF | FF:FF |
| | main: | | | | |
| +00000007 | out PORTC,r16 | 02 5F | FF:FF | FF:FF | FF:FF |
| +00000008 | in r17,SREG | 02 5F | FF:FF | FF:FF | FF:FF |
| +00000009 | rcall delay1 | 02 5D | FF:FF | FF:FF | 00:0A |
| +0000000A | out SREG,r17 | 02 5F | 00:15 | 00:0F | 00:0A |
| +0000000B | rol r16 | 02 5F | 00:15 | 00:0F | 00:0A |
| +0000000C | rjmp main | 02 5F | 00:15 | 00:0F | 00:0A |
| | delay1: | | | | |
| +0000000D | clr r18 | 02 5D | FF:FF | FF:FF | 00:0A |
| | met1: | | | | |
| +0000000E | rcall delay2 | 02 5B | FF:FF | 00:0F | 00:0A |
| +0000000F | inc r18 | 02 5D | 00:15 | 00:0F | 00:0A |
| +00000010 | cpi r18,0x10 | 02 5D | 00:15 | 00:0F | 00:0A |
| +00000011 | brne met1 | 02 5D | 00:15 | 00:0F | 00:0A |
| +00000012 | Ret | 02 5F | 00:15 | 00:0F | 00:0A |
| | delay2: | | | | |
| +00000013 | clr r19 | 02 5B | FF:FF | 00:0F | 00:0A |
| | met2: | | | | |
| +00000014 | rcall delay3 | 02 59 | 00:15 | 00:0F | 00:0A |
| +00000015 | inc r19 | 02 5B | 00:15 | 00:0F | 00:0A |
| +00000016 | cpi r19,0xFA | 02 5B | 00:15 | 00:0F | 00:0A |
| +00000017 | brne met2 | 02 5B | 00:15 | 00:0F | 00:0A |
| +00000018 | Ret | 02 5D | 00:15 | 00:0F | 00:0A |
| | delay3: | | | | |
| +00000019 | clr r20 | 02 59 | 00:15 | 00:0F | 00:0A |
| +0000001A | inc r20 | 02 59 | 00:15 | 00:0F | 00:0A |
| +0000001B | cpi r20,0xFA | 02 59 | 00:15 | 00:0F | 00:0A |
| +0000001C | brne met3 | 02 59 | 00:15 | 00:0F | 00:0A |
| +0000001D | ret | 02 5B | 00:15 | 00:0F | 00:0A |

**Задание на выполнение**

1. Разработать программу «бегущий огонь» с заданной по вариантам программной задержкой, которая изменяется в зависимости от состояния входов. В отчете привести:

– функциональную схему;
– листинг программы;
– дизассемблированную программу;
– алгоритм;
– представить расчет времени задержки по количеству команд;
– представить таблицу значений стека во время исполнения программы.

Варианты программы «бегущий огонь»

| № вар. | 1 такт | 2 такт | 3 такт | 4 такт |
|---|---|---|---|---|
| 1 | PD0, PD1 - 2 с | PD1, PD2 -1 с | PD2, PD3 -0,5 с | – |

| | | | | |
|---|---|---|---|---|
| 2 | PA0…PA3 -0,1 с | PA4…PA7 – 0,2 с | PC0…PC3 -0,3 с | PC4…PC7 – 0,4 с |
| 3 | PC0 – 0,5 с | PC2 – 0,5 с | PC4 – 1,5 с | PC6 – 1,5 с |
| 4 | PB0 – 1 с | Нет – 0,5 с | PB1 – 1 с | Нет – 0,5 с |
| 5 | PC7 – 5 с | PC6, PC7 – 5 с | PC5…PC7 – 5 с | PC4…PC7 – 5 с |
| 6 | PA0 – 2 с | PA1 – 4 с | – | – |
| 7 | PC0…PC3 - 10 с | PC1…PC4 -10 с | PC2…PC5 - 10 с | – |
| 8 | PD7, PC7 -0,5 с | Нет – 2 с | PD0, PC0 - 0,5 с | Нет - 2 с |
| 9 | PC0 – 0,2 с | PC1 – 0,2 с | PC2 – 0,2 с | Нет – 1 с |
| 10 | PA0…PA3 – 0,1 с | Нет – 0,2 с | PA4…PA7 – 0,1 с | Нет – 0,2 с |
| 11 | Порт D – 5 с | Нет – 1 с | Порт C – 5 с | – |
| 12 | PA3 – 2 с | Нет – 4 с | – | – |
| 13 | PC7 – 1 с | Нет – 5 с | PC6 – 1 с | Нет – 5 с |
| 14 | PA0…PA3 – 2 с | Нет – 1 с | PD0…PD3 – 2 с | |
| 15 | PC0…PC3 – 0,1 с | PC1…PC3 -0,1 с | PC2…PC3 – 0,1 с | PC3 – 0,5 с |
| 16 | Порт D – 2 с | Порт A – 2 с | Нет – 4 с | |
| 17 | PC0… PC2 - 2 с | PC1…PC3 - 2 с | PC2…PC4 - 2 с | Нет – 5 с |
| 18 | Порт A -0,1 с | Нет – 1 с | Порт C -0,1 с | Нет – 1 с |
| 19 | PC0 – 0,1 с | PC1 – 0,1 с | PC2 – 0,1 с | Нет – 0,5 с |
| 20 | PB0 – 5 с | PB1 – 2,5 с | PB2 – 1,25 с | Нет – 5 с |
| 21 | PD7 – 1 с | PD6 – 1 с | PD0…5 – 5 с | – |
| 22 | PA6 – 1 с | PA7 – 4 с | – | – |
| 23 | PC0, PC3 – 2 с | PC1, PC4 – 2 с | Нет – 5 с | – |
| 24 | PD4…PD7 – 2,5 с | Нет – 5 с | PD0…PD3 – 2,5 с | Нет – 5 с |
| 25 | Нет – 1,5 с | Порт C – 0,5 с | Нет – 1,5 с | Порт A – 1,5 с |
| 26 | PB0 – 5 с | Нет – 10 с | – | – |
| 27 | PC6,7 – 2 с | PC4, PC5 – 2 с | PC2…PC3 – 2 с | PC0…PC1 – 4 с |
| 28 | PA0…PA3 – 2 с | PA4…PA7 – 2 с | PA0…PA7 – 2 с | Нет – 6 с |
| 29 | PD0,PD2,PD4 – 5 с | PD1, PD3,PD5 – 5 с | Нет – 10 с | – |
| 30 | PC0 – 1,5 с | PC1 – 3 с | PC2 – 4,5 с | – |

*Примечание*: символ '–' обозначает что такт отсутствует, например, в 1 варианте программа выполняется только за 3 такта.

**Контрольные вопросы**

1. Поясните назначение стека. Что такое указатель стека? Вершина стека?
2. В каких случаях в программе необходимо выполнять инициализацию стека?
3. Что произойдет с программой, если определить следующие значения указателя стека:

а) SPH=0, SPL=0 б) SPH=0, SPL=0x5F в) SPH=0x02, SPL=0

4. Каким образом посчитать время исполнения инструкции? Части программы?
5. Какие инструкции выполняются за 1 такт? Какие за 2 такта?
6. Напишите простейший цикл и поясните, как считается время исполнения цикла?
7. Что такое программная задержка и как она реализуется?
8. Какую максимальную задержку в микросекундах можно получить в простейшей программе цикла?
9. Можно ли получить задержку в 1 сек используя только один цикл?

10. Изменится ли время исполнения простейшего цикла в примере, если поменять местами инструкции `inc r16` и `cpi r16,0xFF`?

11. Что такое дисассемблирование программы и для чего оно необходимо?

## Работа № 5. 8-ми разрядные таймеры в режиме широтно-импульсной модуляции

### Цель работы

Освоить теоретический и практический материал по работе 8-разрядных таймеров микроконтроллера Atmega 8535 в режиме широтно-импульсной. Применить приобретенные навыки при написании программы.

### Программа работы

1. Изучить необходимый теоретический материал о режиме «Быстрый ШИМ» таймеров T0 и T2.
2. Изучить необходимый теоретический материал о режиме «Фазово-корректный ШИМ» таймеров T0 и T2.
2. Разобраться в программах по использованию ШИМ, представленных в лабораторной работе.
3. Написать и отладить собственную программу с использованием таймеров в режиме ШИМ в соответствии с вариантом.

### Пояснения к работе

**1. Назначение таймеров. Понятие режима широтно-импульсной модуляции.**

При использовании микроконтроллеров очень часто требуется решать задачи создания точных временных задержек сигналов, оценивать длительность импульсов какого-либо внешнего сигнала, знать его частоту и скважность.

Все эти задачи решаются с помощью использования таймеров/счетчиков.

По сути, таймеры представляют собой счетчики, которые ведут подсчет импульсов либо от внешнего сигнала, либо от внутреннего генератора. Кроме этого таймеры/счетчики имеют возможность работать в режиме широтно-импульсной модуляции (ШИМ).

*Широтно-импульсной модуляцией* называется формирование среднего значения сигнала с помощью сигналов логического «0» и логической «1» за период модуляции T (рис. 1).

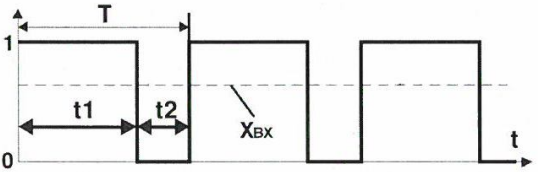



Рис. 1. Принцип широтно-импульсной модуляции сигнала

ШИМ дает возможность с помощью логических сигналов получать на выходе микроконтроллера аналоговый сигнал, среднее значение которого за период можно плавно изменять от 0 до 1, точнее от 0 до напряжения питания. Значение этого сигнала можно рассчитать по следующей формуле:

$$X_{BX}=1\cdot t_1/T,$$

где $t_1$ – время включенного состояния (логической единицы);

$t_2$ – время выключенного состояния (логического нуля);

$T$– период выходного сигнала.

## 2. Таймеры микроконтроллера

Микроконтроллер ATmega8535 содержит три таймера общего назначения: два восьмиразрядных таймера T0 и T2, один шестнадцатиразрядный таймер T1. Восьмиразрядные таймеры T0 и T2 определяются и работают практически идентично друг другу. Работа таймера T1 имеет сущетсвенные отличия и рассматривается в отдельной лабораторной работе. В данной работе изучается работа только таймеров T0 и T2.

Для управления работой таймера T0 используются 3 регистра ввода/вывода:

– счетный регистр TCNT0;

– регистр сравнения OCR0;

– регистр управления TCCR0.

Соответствующие регистры таймера T2 называются TCNT2, OCR2, TCCR2.

Для подключения входов/выходов таймеров к выводам микросхемы контроллера (т.е. соединения таймеров с внешним миром) используются альтернативные функции портов ввода/вывода. После включения альтернативной функции данный бит порта используется только для ввода информации в таймер (например, вход T0/PB0), или вывода информации с выхода таймера (например, вывод OC0/PB3). Альтернативные функции портов для работы с таймерами приведены в табл. 1 (см. прил. 1).

Таблица 1. Альтернативные функции портов, используемые таймерами

| Таймер | Обозначение | Описание | Вывод порта |
|---|---|---|---|
| T0 | T0 | Внешний вход таймера T0 | PB0 |
| | OC0 | Внешний выход таймера T0 | PB3 |
| T1 | T1 | Внешний вход таймера T1 | PB1 |
| | OC1A | Внешний выход A таймера T1 | PD5 |
| | OC1B | Внешний выход B таймера T1 | PD4 |
| T2 | TOSC1 | Внешний вывод 1 для подключения резонатора таймера T2 | PC6 |
| | TOSC2 | Внешний вывод 2 для подключения резонатора для таймера T2 | PC7 |
| | OC2 | Внешний выход таймера T2 | PD7 |

*Примечание*: для использования альтернативных функции портов соответствующие биты портов предварительно необходимо сконфигурировать на ввод или вывод. Например, при использовании альтернативной функции OC0 для вывода информации из таймера T0 вывод микросхемы PB3 должен быть определен как выход: бит DDB3 равен 1.

## 3. Регистры таймера T0

Рассмотрим работу таймера при его работе от источника тактового сигнала процессора (рис. 1), в качестве которого в лабораторном стенде выступает кварцевый резонатор с тактовой частотой 8 МГц.

Счет импульсов источника частоты ведется в 8-разрядном счетном регистре таймера TCNT0 (Timer CouNTer T0). Перед тем, как попасть в схему счета импульсов, тактовый сигнал поступает на схему деления частоты, которая, в соответствии с параметрами, установленными при инициализации таймера, производит деление частоты тактового сигнала $f_{CLK}$ в следующем соотношении:

– без делителя частоты тактового сигнала $f_{CLK}$;
– с делителем частоты тактового сигнала $f_{CLK}/8$;
– с делителем частоты тактового сигнала $f_{CLK}/64$;
– с делителем частоты тактового сигнала $f_{CLK}/256$;
– с делителем частоты тактового сигнала $f_{CLK}/1024$.

Каждый импульс с предделителя частоты считается и инкрементирует значение, содержащееся в счетном регистре TCNT0 (рис. 1).

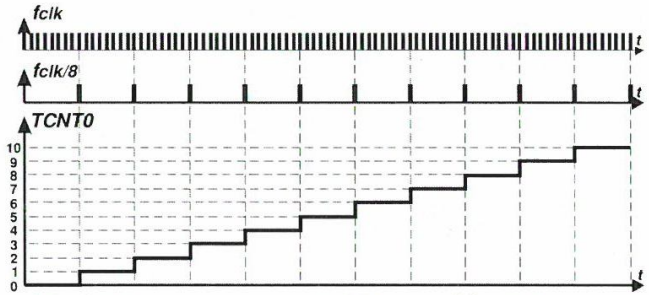



Рис. 1. Работа предделителя таймера T0 при коэффициенте делителя 8

Каждый таймер содержит регистр сравнения. Таймер T0 содержит 8-разрядный регистр OCR0 (Output Compare Register). В это регистр записывается уставка, то есть число от 0 до 255, при достижении которого счетным регистром TCNT0 формируется флаг прерывания по совпадению таймера OCF0 (Output Compare Flag).

В микроконтроллере Atmega 8535 таймеры T0 и T2 работают в двух режимах широтно-импульсной модуляции: быстрый ШИМ и фазово-корректный ШИМ.

В режиме *быстрого ШИМ* счетный регистр TCNT0(2) таймера производит формирование пилообразной развертки (рис. 2, а), инкрементируя свое значение по каждому импульсу с предделителя, который устанавливается в регистре управления **TCCR0(2)** таймера T0(T2). При достижении счетным регистром значения 255 происходит его автоматическое обнуление.

В регистрах сравнения OCR0(2) таймеров T0 и T2 может быть записано любое число от 0 до 255. С этим числом сравнивается значение счетного регистра TCNT0(2) и при их равенстве происходит переключение вывода таймера OC0 или OC2 в соответствии с настройками регистра управления TCCR0(2) таймера. В

случае, если при совпадении TCNT0(2)=OCR0(2) вывод OC0(2) таймера обнуляется, то ШИМ получается неинвертирующим (рис. 2, б), а если при совпадении вывод устанавливается в состояние логической «1», то ШИМ – инвертирующий (рис. 2, в).

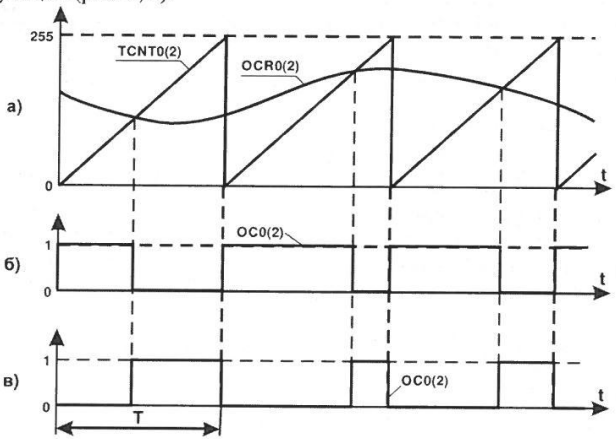



Рис. 2. Принцип работы таймеров T0 и T2 в режиме «Быстрый ШИМ»

*В режиме фазового ШИМ* счетный регистр TCNT0(2) формирует пилообразный сигнал развертки с нарастающим и спадающим фронтами. Сначала регистр инкрементирует свое значение от 0 до 255, а затем происходит его декремент от 255 до 0 (рис. 3, а).

Если при превышении значением, содержащимся в счетном регистре TCNT0(2), значения, содержащегося в регистре сравнения OCR0(2) происходит обнуление вывода OC0(2) таймера, то ШИМ является неинвертирующим (рис. 3, б). В противном случае ШИМ – инвертирующий (рис. 3, в).

Несмотря на то, что в режиме фазового ШИМ частота ШИМ-сигнала меньше, чем в режиме быстрого ШИМ, первый режим обладает большей разрешающей способностью и используется для достижения большей точности модуляции.

**Регистры таймера T0 в режиме ШИМ**

Режим широтно-импульсной модуляции, как и все остальные режимы работы таймера T0, инициализируется в регистре управления TCCR0 (табл. 2).

Таблица 2. Регистр управления TCCR0 таймера T0

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |

За установку режима работы таймера отвечают биты WGM01:WGM00. При установке WGM01=0, WGM00=1 активируется режим фазового ШИМ, при установке WGM01=1, WGM00=1 активируется режим быстрого ШИМ.

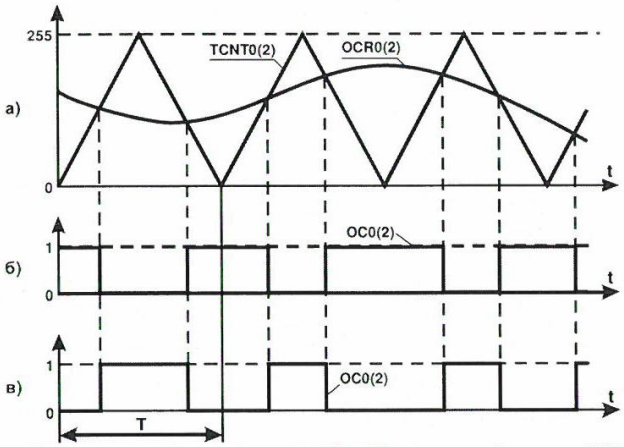



Рис. 3. Принцип работы таймеров Т0 и Т2 в режиме фазового ШИМ

При активации режима широтно-импульсной модуляции биты COM01 и COM00 определяют поведение вывода таймера Т0, в качестве которого выступает вывод PB3 микроконтроллера. Поведение вывода в соответствии с этими битами представлено в табл. 3 и табл. 4.

Таблица 3. Внешний выход OC0 таймера Т0 в режиме быстрого ШИМ

| COM01 | COM00 | Описание |
|---|---|---|
| 0 | 0 | Нормальная функция порта, вывод OC0 отключен |
| 0 | 1 | Комбинация бит зарезервирована |
| 1 | 0 | Очистка OC0 при совпадении. Неинвертирующий ШИМ |
| 1 | 1 | Установка OC0 при совпадении. Инвертирующий ШИМ |

Таблица 4. Внешний выход OC0 таймера Т0 в режиме фазового ШИМ

| COM01 | COM00 | Описание |
|---|---|---|
| 0 | 0 | Нормальная функция порта, вывод OC0 отключен |
| 0 | 1 | Комбинация бит зарезервирована |
| 1 | 0 | Очистка OC0 при совпадении при счете вверх. Неинвертирующий ШИМ |
| 1 | 1 | Установка OC0 при совпадении при счете вверх. Инвертирующий ШИМ |

Частота ШИМ-сигнала устанавливается битами CS02…CS00. Эти биты определяют источник задания частоты и коэффициент делителя (табл. 5).

Таблица 5. Установка источника задания частоты таймера Т0

| CS02 | CS01 | CS00 | Описание |
|---|---|---|---|
| 0 | 0 | 0 | Таймер остановлен |
| 0 | 0 | 1 | $f_{T0} = f_{CLK}/1$ |

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | $f_{T0} = f_{CLK}/8$ |
| 0 | 1 | 1 | $f_{T0} = f_{CLK}/64$ |
| 1 | 0 | 0 | $f_{T0} = f_{CLK}/256$ |
| 1 | 0 | 1 | $f_{T0} = f_{CLK}/1024$ |
| 1 | 1 | 0 | Внешний сигнал на входе T0. Спадающий фронт сигнала. |
| 1 | 1 | 1 | Внешний сигнал на входе T0. Нарастающий фронт сигнала. |

В соответствии с табл. 5, частота ШИМ рассчитывается следующим образом:

– для режима быстрого ШИМ: $f = f_{T0}/256$;

– для режима фазового ШИМ: $f = f_{T0}/512$.

**Регистры таймера T2 в режиме ШИМ**

Режим широтно-импульсной модуляции, как и все остальные режимы работы таймера T2, инициализируется в регистре управления TCCR2 (табл. 6).

Таблица 6. Регистр управления TCCR2 таймера T2

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | FOC2 | WGM20 | COM21 | COM20 | WGM21 | CS22 | CS21 | CS20 |

Описание и назначение управляющих бит регистра TCCR2 таймера T2 аналогичны описанным битам регистра TCCR0 таймера T0, поэтому отдельно не комментируются. Частота ШИМ-сигнала устанавливается битами CS22…CS20. Эти биты определяют источник задания частоты и коэффициент делителя (табл. 7).

Таблица 7. Установка источника задания частоты таймера T2

| CS22 | CS21 | CS20 | Описание |
|---|---|---|---|
| 0 | 0 | 0 | Таймер остановлен |
| 0 | 0 | 1 | $f_{T0} = f_{CLK}/1$ |
| 0 | 1 | 0 | $f_{T0} = f_{CLK}/8$ |
| 0 | 1 | 1 | $f_{T0} = f_{CLK}/32$ |
| 1 | 0 | 0 | $f_{T0} = f_{CLK}/64$ |
| 1 | 0 | 1 | $f_{T0} = f_{CLK}/128$ |
| 1 | 1 | 0 | $f_{T0} = f_{CLK}/256$ |
| 1 | 1 | 1 | $f_{T0} = f_{CLK}/1024$ |

Необходимо отметить, что в режиме ШИМ необязательно устанавливать маски и разрешать прерывания по совпадению и переполнению таймера T0 и T2. Эти прерывания следует активировать только при необходимости их использования в программе.

Для запуска режима ШИМ на таймере T0 или T2 необходимо:

– остановить таймер обнулением регистра управления TCCR0(2);

– обнулить регистр сравнения OCR0(2) и счетный регистр TCNT0(2);

– установить в регистре управления TCCR0(2) режим ШИМ и выбрать частоту его работы.

В процессе работы ШИМ в любой момент можно изменять содержимое регистра сравнения OCR0(2) таймеров, при этом среднее значение ШИМ-сигнала на выводе OC0 и OC2 микроконтроллера будет пропорционально изменяться.

**Пример.** Таймер T0 работает в режиме быстрого ШИМ. В зависимости от состояния битов порта A регулируется яркость светодиода, подключенного к выходу ШИМ таймера T0. Логическим сигналом с 0 бита порта D ШИМ изменяется с инвертирующего на неинвертирующий.

```
;---------------------------------------------------------------------
;ШИМ на таймере T0
;входы:
;     PORTA0...PORTA7      - задание уставки таймера
;     PORTD0               - инвертирующий (1)/неинвертирующий (0) ШИМ
;выходы:
;     PORTB3               - ШИМ на таймере T0

.include"m8535def.inc"   ;подключение      стандартной      библиотеки
ATmega8535
.cseg                    ;начало сегмента кода
.org$0                   ;по адресу 0
rjmp reset               ;и переход на reset

reset:
   ldi r16,0x01      ;инициализация портов ввода вывода.
   out PORTD,r16     ;PORTD0 - на ввод информации
   clr r16
   out DDRD,r16
   ldi r16,0xFF
   out PORTA,r16     ;PORTA - на ввод информации
   clr r16
   out DDRA,r16
   out PORTB,r16
   ldi r16,0x08
   out DDRB,r16      ;PORTB3 - на вывод информации
   clr r16
   out TCCR0,r16     ;сброс регистра управления таймера T0
   out OCR0,r16      ;сброс регистра сравнения таймера T0
   out TCNT0,r16     ;сброс регистра счета таймера T0
   ldi r16,0x69      ;установка режима быстрого неинвертирующего
   out TCCR0,r16     ;ШИМ таймера T0
main:                ;начало основной программы
   in r16,PINA       ;считывание значений порта ввода/вывода PORTA
   out OCR0,r16      ;и его отправка в регистр сравнения таймера T0
   in r16,PIND       ;считывание значений порта ввода/вывода PORTD
   andi r16,0x01     ;и выделение бита PORTD0
   cpi r16,0x01      ;Если PORTD0=1
   breq met1         ;то переход на метку met1
   ldi r17,0x69      ;иначе запись в r17 значения для неинверт. ШИМ
   rjmp met2         ;и переход на метку met2
met1:                ;По метке met1
```

```
   ldi r17,0x79      ;запись в r17 значения для инвертирующего ШИМ
met2:                ;По метке met2
   out TCCR0,r17     ;установка заданного режима работы таймера T0
   rjmp main         ;и возврат на main
;---------------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. Инициализации портов ввода/вывода в соответствии с поставленным заданием:

```
   ldi r16,0x01
   out PORTD,r16
   clr r16
   out DDRD,r16
   ldi r16,0xFF
   out PORTA,r16
   clr r16
   out DDRA,r16
   out PORTB,r16
   ldi r16,0x08
   out DDRB,r16
```

2. Инициализация таймера T0 на заданный режим работы:

```
   clr r16
   out TCCR0,r16
   out OCR0,r16
   out TCNT0,r16
   ldi r16,0x69
   out TCCR0,r16
```

Сначала очищаются все регистры таймера (`clr r16; out TCCR0,r16; out OCR0,r16; out TCNT0,r16`). После этого в регистр управления TCCR0, в соответствии с табл. 1 записываются необходимые комбинации управляющих бит. Установкой WGM00=1, WGM00=1 выбирается режим быстрого ШИМ; установкой COM01=1, COM00=0 выбирается режим неинвертирующего ШИМ; биты CS02:CS01:CS00=001 определяют работу таймера без предделителя частоты процессора clk/1.

3. В начале цикла происходит считывание данных с порта A и их запись в регистр сравнения таймера T0:

```
main:
     in r16,PINA
     out OCR0,r16
```

4. После этого производится опрос порта D и выделение младшего значащего бита:

```
     in r16,PIND
     andi r16,0x01
```

5. Сравнение PORTD0 с логическими уровнями 0 и 1:

```
     cpi r16,0x01
     breq met1
     ldi r17,0x69
     rjmp met2
met1:
     ldi r17,0x79
```

Если в результате сравнения (`cpi r16,0x01`) PORTD0=1, то происходит переход на метку met1, по которой в регистр r17 записывается значение, которое необходимо записать в регистр управления TCCR0 таймера T0 для реализации инвертирующего ШИМ (`ldi r17,0x79`). Иначе в r17 производится запись значения TCCR0 для неинвертирующего ШИМ (`ldi r17,0x69`).

6. При переходе на метку met2 происходит запись полученного в r17 значения в регистр управления TCCR0 таймера T0 и зацикливание программы:

```
met2:
out TCCR0,r17
rjmp main
```

Подробное рассмотрение программы показывает, что при использовании в таймере режима ШИМ пользователь может не использовать прерывания по совпадению и переполнению таймера, так как в режиме ШИМ все действия таймер делает автоматически. Это существенно облегчает организацию программы и уменьшает ее размер.

**Задание на выполнение**

1. Составить программу для своего варианта, которая реализует вывод сигнала с ШИМ с изменением скважности для заданных таймеров по коду входных сигналов (биты задания). В отчете привести:

– исходное задание;
– функциональную схему;
– представить расчет скважности, настройку таймера
– запись данных (скважности) в ОЗУ или ПЗУ;
– листинг программы;
– дизассемблированную программу;
– алгоритм;
– представить таблицу значений стека во время исполнения программы.

Варианты задания

| № вар | Таймер (ы), канал (ы) | Режим | Диапазон частот ШИМ | Входы (биты) | Биты задания, количество дискретных значений, диапазон изменения скважности |
|---|---|---|---|---|---|
| 1 | T2 | Б / П | [1000, 10000] | | PC5…PC7, 8 положений, 0…0,5 |
| 2 | Все 4 канала | Ф / И | [1000, 10000] | PA0- включает попарно | PA4…PA7, 16 положений, 0,25…0,5 |
| 3 | T1, A | Б / И | [200, 1000] | | Порт A, 256 положений, 0…1 |
| 4 | T0 и T2 | Ф / П и И | [200, 1000] | Одновременно работают 2 канала | PC0…PC3, 16 положений, 0…0,5 |
| 5 | T1, B | Б / И | <200 | | PC0…PC4, 32 положения, 0…0,5 |
| 6 | T1, A и B | Ф / П и И | >10000 | PA0-выбирает канал | PB0…PB2, 8 положений, 0…1 |
| 7 | Все 4 канала | Б / И | >10000 | PC0, PC1 выбирают номер канала | PA0…PA3, 16 положений, 0…1 |

| | | | | | |
|---|---|---|---|---|---|
| 8 | T0 | Ф / И | <200 | | Порт С, 256 положений, 0…1 |
| 9 | T1, A | Б / П | [1000, 10000] | | PA2…PA3, 4 положения, 0…0,5 |
| 10 | T0 и T2 | Ф / П и И | [1000, 10000] | PC7-выбирает канал | PC0…PC5, 64 положения, 0…1 |
| 11 | Все 4 канала | Б / И | [200, 1000] | PA0-T0, PA1-P2, PA2-T1A, PA3-T1B | PC6…PC7, 4 положения, 0…1 |
| 12 | T1, A и B | Ф / П и И | [200, 1000] | Одновременно работают 2 канала | Порт А, 256 положения, 0…0,5 |
| 13 | T2 | Б / И | <200 | | PA2…PA5, 16 положений, 0…1 |
| 14 | Все 4 канала | Ф / П | >10000 | PA0, PA1 выбирают номер канала | PC3…PC5, 8 положений, 0…0,75 |
| 15 | T1, A | Б / И | >10000 | | PB0…PB3, 16 положений, 0…0,5 |
| 16 | T0 и T2 | Ф / П и И | <200 | Одновременно работают 2 канала | PA3…PA7, 32 положения, 0…1 |
| 17 | T1, B | Б / И | [1000, 10000] | | PB0…PA2, 8 положений, 0…1 |
| 18 | T1, A и B | Ф / П и И | [1000, 10000] | PA7-выбирает канал | PA0…PA3, 16 положений, 0…0,5 |
| 19 | Все 4 канала | Б / П | [200, 1000] | PA4-T0, PA5-P2, PA6-T1A, PA7-T1B | PC1…PC3, 8 положений, 0,25…0,75 |
| 20 | T0 | Ф / И | [200, 1000] | | PA4…PA7, 16 положений, 0,25…0,5 |
| 21 | T1, A | Б / П | <200 | | PB0…PB4, 32 положений, 0…1 |
| 22 | T0 и T2 | Ф / П и И | >10000 | PA3-выбирает канал | PA3…PA7, 32 положений, 0…0,5 |
| 23 | T1, B | Б / И | >10000 | | PC, 256 положений, 0…1 |
| 24 | T1, A и B | Ф / П и И | <200 | Одновременно работают 2 канала | PA5…PB7, 8 положений, 0…0,5 |
| 25 | Все 4 канала | Ф / П, И, П, И | [1000, 10000] | Одновременно работают 4 канала | PB0…PB2, 8 положений, 0…0,75 |
| 26 | T0 | Б / И | [200, 1000] | | PA0…PA6, 128 положений, 0…1 |
| 27 | T1, A | Ф / И | [200, 1000] | | PC0…PC5, 64 положения, 0…0,5 |
| 28 | T0 и T2 | Б / П и И | [1000, 10000] | PC1-выбирает канал | PB6…PB7, 4 положения, 0…1 |
| 29 | T1, B | Ф / П | >10000 | | PA3…PA7, 32 положений, 0…0,75 |
| 30 | T1, A и B | Б / П и И | >10000 | PC6-выбирает канал | PC2…PC5, 16 положения, 0…0,5 |

Условные обозначения: Б – быстрый ШИМ, Ф – фазовый ШИМ, П – прямой ШИМ, И - инверсный

**Контрольные вопросы**

1. Укажите разрядность таймера T0? T1? T2?

2. Какие функции в программе могут выполнять таймеры?

3. Перечислите регистры ввода/вывода, управляющие работой таймера T0?

4. В каких режимах с ШИМ могут работать таймеры микроконтроллера ATmega8535?

5. Как настроить таймеры T0/T2 для работы в режиме быстрого ШИМ?

6. Какую функцию выполняет счетный регистр TCNT0 в режиме ШИМ?

7. Для каких целей используется регистр сравнения OCR0?

8. Какие значения необходимо записать в регистр OCR0 для получения скважности 0, 0,5 и 0,75?

## Работа № 6. 8-ми разрядные таймеры в режиме создания временных интервалов

### Цель работы

Освоить теоретический и практический материал по работе 8-разрядных таймеров микроконтроллеров. Применить приобретенные навыки при написании программы.

### Программа работы

1. Изучить необходимый теоретический материал о таймере T0.
2. Изучить необходимый теоретический материал о таймере T2.
2. Изучить представленные в лабораторной работе примеры программ по использованию таймера.
3. Написать и отладить собственную программу с использованием таймеров.

### Пояснения к работе

При использовании микроконтроллеров очень часто требуется с высокой точностью вести подсчет временных интервалов, оценивать длительность импульсов какого-либо внешнего сигнала, знать его частоту и скважность.

Все эти задачи решаются с помощью использования таймеров/счетчиков.

Для того, чтобы понять, что такое таймер микроконтроллера AVR, необходимо разобраться в принципе подсчета временных интервалов.

По сути, таймеры микроконтроллеров AVR представляют собой счетчики, которые ведут подсчет импульсов какого-либо внешнего сигнала.

Если период колебаний внешнего сигнала известен и стабилен во времени, то простым подсчетом количества периодов такого сигнала можно с высокой точностью вести подсчет временных интервалов.

Обычно в качестве источника стабильного периодического сигнала используется частота тактовых колебаний процессора, которая определяется используемым источником частоты – внутренним RC-резонатором или внешним кварцевым резонатором.

### 8-разрядный таймер/счетчик T0

Таймер/счетчик T0 служит для подсчета временных интервалов, регистрации внешних событий и работы в режиме ШИМ для вывода цифрового периодического сигнала с регулируемой скважностью и частотой.

Таймер синхронизируется от источника тактового сигнала процессора, либо от источника внешнего сигнала, подаваемого на цифровой вход микроконтроллера.

Рассмотрим работу таймера при его работе от источника тактового сигнала процессора (рис. 1), в качестве которого в лабораторном стенде выступает кварцевый резонатор с тактовой частотой 8 МГц.

Счет импульсов источника частоты ведется в 8-разрядном счетном регистре таймера **TCNT0** (Timer Counter T0). Перед тем, как попасть в схему счета импульсов, тактовый сигнал поступает на схему деления частоты, которая, в

соответствии с параметрами, установленными при инициализации таймера, производит деление частоты тактового сигнала $f_{CLK}$ в следующем соотношении:

– без делителя частоты тактового сигнала $f_{CLK}$;
– с делителем частоты тактового сигнала $f_{CLK}/8$;
– с делителем частоты тактового сигнала $f_{CLK}/64$;
– с делителем частоты тактового сигнала $f_{CLK}/256$;
– с делителем частоты тактового сигнала $f_{CLK}/1024$.

Каждый импульс с предделителя частоты считается и инкрементирует значение, содержащееся в счетном регистре **TCNT0** (рис. 1).

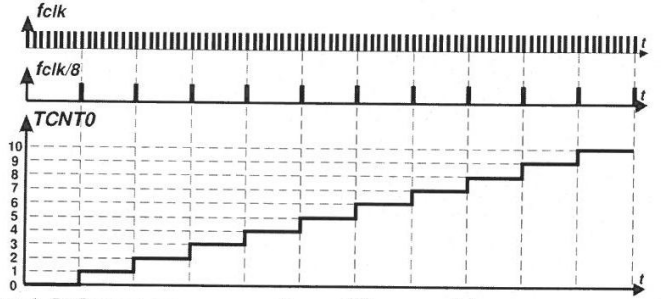



Рис. 1. Работа предделителя таймера T0 при коэффициенте делителя 8

Каждый таймер содержит регистр сравнения. Таймер T0 содержит 8-разрядный регистр **OCR0** (Output Compare register). В это регистр записывается уставка, то есть число от 0 до 255, при достижении которого счетным регистром **TCNT0** формируется флаг прерывания по совпадению таймера **OCF0** (Output Compare Flag).

Когда значение, записанное в регистре **TCNT0**, переполняет максимальное значение, которое можно записать в 8-разрядный регистр сравнения **OCR0**, формируется флаг прерывания по переполнению таймера **TOV0** (Timer Overflow Flag) (рис. 2).

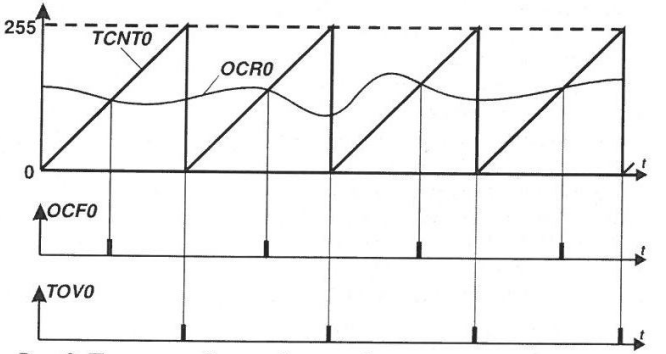



Рис. 2. Принцип работы таймера и формирования прерываний

Настройки таймера определяет регистр **TCCR0** (Timer/Counter Control register).

Табл. 1. Управляющий регистр TCCR0 таймера T0

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS2 | CS1 | CS0 |

**Бит 7 – FOC0** – бит принудительной установки в единичное значение выходного сигнала таймера в режиме ШИМ (в данной работе не рассматривается).

**Бит 6 и бит 3 – WGM00:WGM01 –** биты управления режимом работы таймера. Эти биты определяют режим работы таймера: нормальный, ШИМ-режимы или сброс при совпадении (обратите внимание 6 бит регистра устанавливает младший разряд режима работы, 3 бит – старший разряд). Виды режимов работы представлены в табл. 2.

Табл. 2. Режимы работы таймера/Счетчика T0

| WGM01 | WGM00 | Режим |
|---|---|---|
| 0 | 0 | Нормальный |
| 0 | 1 | Фазовый ШИМ |
| 1 | 0 | Очистка при совпадении |
| 1 | 1 | Быстрый ШИМ |

Таким образом при нулевых значениях битов WGM счетчик T0 работает в обычном счетном режиме, ШИМ-режимы работы будут рассмотрены в следующей работе.

Рассмотрим работу счетчика в режиме «*очистка при совпадении*».

В нормальном режиме работы счетный регистр **TCNT0** изменяется от 0 до 255 и далее продолжает счет (рис. 1, б).

В режиме «*очистка при совпадении*» значение уставки записывается в регистр **OCR0**. Когда значение счетного регистра **TCNT0** доходит до уставки **OCR0**, регистр **TCNT0** автоматически очищается, а таймер – перезапускается. Этот режим очень удобен при необходимости периодического выполнения каких-либо операций (рис. 3).

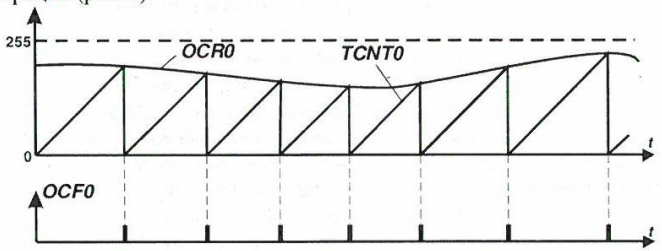



Рис. 3. Режим работы CTC (clear to compare-очистка при совпадении) таймера T0

**Биты 5 и 4 – COM01:COM00** – биты управления выводом OC0. Таймер T0 может выводить сигнал на внешние контакты микросхемы, этот вывод обозначается OC0 и определяется как альтернативная функция вывода PB3 порта ввода/вывода B. Управление выводом OC0 выполняется комбинацией бит COM01:COM00 (табл. 3) и зависит также от режима работы таймера (биты WGM01:WGM00).

Табл. 3. Функции вывода OC0 таймера T0 в нормальном режиме работы

| COM01 | COM00 | Пояснение |
|---|---|---|
| 0 | 0 | Вывод OC0 отключен |
| 0 | 1 | Изменение OC0 на противоположное при совпадении OCF0 |
| 1 | 0 | Очистка OC0 при совпадении OCF0 |
| 1 | 1 | Установка OC0 при совпадении OCF0 |

**Биты 3...0 – CS02...CS00** – биты предделителя таймера и источника задания частоты таймера (табл. 4).

Табл. 4. Функции бит CS02, CS01, CS00 таймера T0

| CS02 | CS01 | CS00 | Пояснение |
|---|---|---|---|
| 0 | 0 | 0 | Нет источника. Таймер остановлен. |
| 0 | 0 | 1 | Предделитель Кд=1 (частота $f_{CLK}$) |
| 0 | 1 | 0 | Предделитель Кд=8 (частота $f_{CLK}/8$) |
| 0 | 1 | 1 | Предделитель Кд=64 (частота $f_{CLK}/64$) |
| 1 | 0 | 0 | Предделитель Кд=256 (частота $f_{CLK}/256$) |
| 1 | 0 | 1 | Предделитель Кд=1024 (частота $f_{CLK}/1024$) |
| 1 | 1 | 0 | Внешний сигнал на выводе T0 (по спадающему фронту) |
| 1 | 1 | 1 | Внешний сигнал на T0 (по нарастающему фронту) |

Помимо управляющего регистра **TCCR0**, счетного регистра **TCNT0** и регистра сравнения **OCR0**, в микроконтроллере содержатся регистр масок прерываний таймеров **TIMSK** и регистр флагов прерываний таймеров и **TIFR**. Регистры позволяют отслеживать то или иное событие (совпадение, переполнение, захват), происходящее во всех таймерах микроконтроллера.

Регистры **TIMSK** и **TIFR** общие для всех таймеров, поэтому они содержат установки для всех таймеров (T0, T1 и T2) микроконтроллера.

Регистр маски прерываний таймеров **TIMSK** (табл. 5) разрешает то или иное прерывание того или иного таймера. Так, например:

– 0 разряд TOIE0 регистра установлен в единицу, это означает, что разрешается прерывание «Переполнение таймера T0»;

– 1 разряд OCIE0 регистра установлен в единицу, это означает, что разрешается прерывание «Совпадение таймера T0» и т.д.

Регистр флагов прерываний таймеров **TIFR** (табл. 6) показывает какое прерывание прозошло.

Например, для таймера T0 в регистре **TIFR** используются два бита:

– бит 1 – OCF0 – флаг прерывания по совпадению таймера;

– бит 0 – TOV0 – флаг прерывания по переполнению таймера.

Табл. 5. Регистр масок прерываний таймеров **TIMSK**

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
| Наименование бита разрешения прерывания | Совпадение таймера T2 | Переполнение таймера T2 | Захват таймера T1 | Совпадение A таймера T1 | Совпадение B таймера T1 | Переполнение таймера T1 | Совпадение таймера T0 | Переполнение таймера T0 |

Табл. 6. Регистр флагов прерываний таймеров **TIFR**

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 |
| Наименование флага прерывания | Флаг совпадения таймера T2 | Флаг переполнения таймера T2 | Флаг захвата таймера T1 | Флаг совпадения A таймера T1 | Флаг совпадения B таймера T1 | Флаг переполнения таймера T1 | Флаг совпадения таймера T0 | Флаг переполнения таймера T0 |

Значение этих регистров следующее. Когда возникает переполнение или совпадение таймера T0, автоматически формируется флаг прерывания TOV0 или OCF0 в регистре **TIFR**. Если в регистре масок прерываний таймеров **TIMSK** на соответствующее прерывание наложена маска, то вызывается процедура обработки прерывания. Если же маска прерывания не наложена, то при совпадении или переполнении таймера ничего не происходит.

**8-разрядный таймер/счетчик T2**

Принцип и режимы работы таймера T2 практически не отличаются от рассмотренного ранее таймера T0, за исключением следующего:

а) изменены коэффициенты предделителя и источник задания частоты;

б) в таймере T2 предусмотрен и асинхронный режим работы. Этот режим заключается в том, что если таймер T0 получает тактовый сигнал либо от источника частоты микроконтроллера $f_{CLK}$ , либо от внешнего источника тактирующего сигнала, подающегося на вывод T0 (PB0), то таймер T2 может работать в асинхронном режиме работы, будучи запитанным от отдельного источника частоты (асинхронный режим). В этом режиме тактовый сигнал поступает на выводы TOSC1, TOSC0 микроконтроллера. Асинхронный режим таймера T2 при проведении лабораторных работ не используется, теоретические аспекты его использования могут быть изучены студентами самостоятельно.

Таймер T2, как и таймер T0, содержит 8-разрядный управляющий регистр TCCR2, регистр счета TCNT2 и регистр сравнения TCNT2. Назначение этих регистров подобно назначению соответствующих регистров таймера T0.

Табл. 7. Управляющий регистр TCCR2

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | FOC2 | WGM20 | COM21 | COM20 | WGM21 | CS22 | CS21 | CS20 |

**Бит 7 – FOC2** – бит режима принудительной установки выходного сигнала таймера в режиме ШИМ (в данной работе не рассматривается).

**Бит 6 и бит 3 –WGM20:WGM21** –биты режима работы таймера Т2 (табл.8).

Табл. 8. Режимы работы таймера Т2

| WGM21 | WGM20 | Режим |
|---|---|---|
| 0 | 0 | Нормальный |
| 0 | 1 | Фазовый ШИМ |
| 1 | 0 | Очистка при совпадении |
| 1 | 1 | Быстрый ШИМ |

**Биты 5 и 4 – COM21:COM20** – режим подключения вывода OC2 таймера Т2 в качестве альтернативной функции вывода PD7 порта ввода/вывода D. Таймер Т2 может управлять этим выводом в соответствии с выбранным режимом работы, определяемым битами WGM21:WGM20, а также в соответствии с комбинацией разрядов COM21:COM20 (табл. 9)

Табл. 9. Функции вывода OC2 таймера Т2 в нормальном режиме работы

| COM21 | COM20 | Пояснение |
|---|---|---|
| 0 | 0 | Вывод OC2 отключен |
| 0 | 1 | Изменение OC2 на противоположное при совпадении OCF2 |
| 1 | 0 | Очистка OC2 при совпадении OCF2 |
| 1 | 1 | Установка OC2 при совпадении OCF2 |

**Биты 3…0 – CS22…CS20** – определяют источник задания частоты таймера Т2 и предделитель таймера (табл. 10).

Табл. 10. Функции бит CS22, CS21, CS20 таймера Т2

| CS02 | CS01 | CS00 | Пояснение |
|---|---|---|---|
| 0 | 0 | 0 | Нет источника. Таймер остановлен. |
| 0 | 0 | 1 | Предделитель Кд=1 (частота $f_{CLK}$) |
| 0 | 1 | 0 | Предделитель Кд=8 (частота $f_{CLK}/8$) |
| 0 | 1 | 1 | Предделитель Кд=32 (частота $f_{CLK}/32$) |
| 1 | 0 | 0 | Предделитель Кд=64 (частота $f_{CLK}/64$) |
| 1 | 0 | 1 | Предделитель Кд=128 (частота $f_{CLK}/128$) |
| 1 | 1 | 0 | Предделитель Кд=256 (частота $f_{CLK}/256$) |
| 1 | 1 | 1 | Предделитель Кд=1024 (частота $f_{CLK}/1024$) |

Источник тактового сигнал выбирается в регистре ASSR (Asinchronous Status Register).

Табл. 11. Регистр асинхронного режима таймера Т2

| Бит | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Название | – | – | – | – | AS2 | TCN2UB | OCR2UB | TCR2UB |

**Бит 3 – AS2** определяет, в каком режиме работает таймер. Если бит AS2=0, то таймер работает в синхронном режиме, используя в качестве тактового сигнала источник частоты микропроцессора. В случае, когда AS2=1, таймер получает

тактовый сигнал от внешнего кварцевого резонатора, подключенного к выводам TOSC1, TOSC0 микроконтроллера.

**Биты 2, 1, 0** – используются в асинхронном режиме и в данной работе не рассматриваются.

В регистре флагов таймеров **TIFR** (табл. 5) таймер T2 имеет два бита: бит 7 OCF2 (флаг прерывания сравнения таймера T2); бит 6 TOV2 (флаг прерывания по переполнению таймера T2).

В регистре масок прерываний таймеров **TIMSK** (табл. 6) таймер T2 имеет два бита: бит 7 OCIE2 (маска прерывания сравнения таймера T2); бит 6 TOIE2 (маска прерывания по переполнению таймера T2).

**Пример.** Написать программу, осуществляющую инкремент какого-либо регистра до значения, заданного на входе порта D, при этом содержимое регистра выводить с интервалом 500 мс в порт C.

```
;---------------------------------------------------------------------
;Программа вывода в PORTC с периодом 500 мс увеличивающегося значения
;двоичного кода до значения, заданного на входах PIND.
;Входы:
;PD0…PD7 - задание максимального значения числа
;PC0…PC7 - индикация результата
.include"m8535def.inc"   ;подключение библиотеки контроллера
.cseg                    ;начало сегмента кода
.org $0                  ;по адресу 0
   rjmp reset            ;переход на метку reset
.org $13                 ;адрес обработки прерывания по совпадению T0
   rjmp T0_compare       ;и переход по этому адресу в случае
                         ;возникновения прерывания
         ; инициализация стека
reset:
   ldi r16,low(RAMEND)   ;запись в указатель стека адреса конца
   ldi r17,high(RAMEND)  ;памяти данных
   out spl,r16
   out sph,r17
         ; инициализация портов
   ldi r16,0xFF          ;инициализация портов ввода/вывода:
   out PORTD,r16         ;порт D - на ввод
   out DDRC,r16          ;порт C - на вывод
   clr r16
   out DDRD,r16
   out PORTC,r16
         ; инициализация и запуск таймера T0
   ldi r16,0x00          ;обнуление регистров таймера T0:
   out TCCR0,r16         ;регистра управления TCCR0
   out TCNT0,r16         ;регистра счета TCNT0
   ldi r16,0x4E          ;запись в регистр сравнения OCR0 числа 0x4E
   out OCR0,r16          ;соответствующего уставке 10 мс
   ldi r16,0x05          ;запуск таймера T0 с предделителем clk/1024
   out TCCR0,r16         ;в нормальном режиме работы
         ; разрешение работы прерывания
   ldi r16,0x02          ;для работы прерывания по совпадению T0
   out TIMSK,r16         ;накладывается маска OCIE0 в регистре TIMSK
```

```
   clr r16                ;необходимые в программе регистры
   clr r17                ;предварительно очищаются
   clr r18
   sei                    ;глобальное разрешение прерываний
main:                     ;основная часть программы
   rjmp main              ;пустой цикл
        ; обработчик прерывания
T0_compare:               ;при совершении прерывания по совпадению T0
   cli                    ;глобальный запрет прерываний
   in r24,SREG            ;сохранение регистра SREG
   clr r16                ;обнуление счетного регистра TCNT0
   out TCNT0,r16
   inc r17                ;инкремент РОН r17
   cpi r17,0x32           ;сравнение r17 с уставкой 0x32 (50)
   brne quit              ;если r17 не равен уставке, то переход на
   quit
   clr r17                ;иначе очистка r17
   in r16,PIND            ;ввод данных, содержащихся на PIND
   cp r16,r18             ;и сравнение РОН r16 с регистром счета r18
   brpl met_1             ;если r16>r18, то переход на met_1
   clr r18                ;иначе очистка регистра счета r18
met_1:
   out PORTC,r18          ;далее - вывод на индикацию r18
   inc r18                ;и инкремент регистра счета
quit:
   out SREG,r24           ;восстановление регистра SREG
   sei                    ;глобальное разрешение прерываний
   reti                   ;выход из подпрограммы прерываний
;---------------------------------------------------------------------
```

Рассмотрим особенности программы.

**1.** Строками программы

```
.org$13
rjmp T0_compare
```

выполняется инициализация адреса памяти, по которому начинается подпрограмма обработки прерывания по совпадению таймера T0. Адреса подпрограмм обработки перываний для каждого микроконтроллера заданы жестко и изменению не подлежат. Таблица адресов приведена в прил. 3.

**2.** В программе используется стек. Это связано с использованием в программе прерывания и необходимости сохранения в стеке адреса возврата из прерывания в основную программу. Поэтому вначале программы инициализируются регистры указателя стека SPH:SPL.

**3.** Для использования таймера T0 вначале необходимо выполнить его инициализацию. Для этого выполняются:

– обнуление счетного регистра TCNT0 и регистра управления TCCR0;

– устанавливается значение уставки в регистр сравнения OCR0.

Выполним расчет уставки. В случае данной программы получить уставку 500 мс на таймере не получается, так как даже при предделителе clk/1024 максимальная уставка таймера равняется:

$$T_{УСТ} = \left(\frac{8000000}{1024}\right)^{-1} \cdot 256 = 32{,}768 \text{ мс.}$$

Как реализовать на таймере T0 уставку большую значения 32,768 мс? Один из вариантов считать количество прерываний таймера. Например, для получения уставки 500 мс можно запрограммировать таймер на уставку 10 мс и производить подсчет срабатываний таймера – каждое 50-е срабатывание таймера будет соответствовать времени $T_{УСТ}$ = 50*10мс= 500 мс. Рассчитаем значение уставка таймера в 10мс при предделителе clk/1024:

$$OCR0 = 10 \cdot 10^{-3} \cdot \frac{8000000}{1024} = 78{,}125.$$

Таким образом, кругленное значение, записываемое в регистр сравнения OCR0, равно 78 (0x4E).

– запуск таймера. Выполняется установкой коэффициента предделителя равного 1024, после этого таймер запускается в работу в нормальном режиме.

**4.** Разрешение работы прерывания по совпадению таймера T0. Для этого:

– в регистре масок прерываний таймеров TIMSK уставливается в единичное значение 1 бит для разрешения прерывания по совпадению таймера;

– разрешается работа всех прерываний (инструкция `sei`) путем записи логической «1» в старший бит регистра SREG и основная программа.

**5.** Основной цикл `main` пустой, т.к. не содержит никаких операторов.

**6.** Выполнение обработчика прерывания. При наступлении прерывания по совпадению T0 осуществляется переход из любого места основной программы (в данном случае только из строки `rjmp main`) по адресу обработки прерывания, указанному в строке `rjmp T0_compare`. После этого:

– запрещаются все прерывания (`cli`); которые разрешаются только по окончании обработки прерывания;

– сохраняется значение регистра SREG (`in r24,SREG`), который при выходе из подпрограммы восстанавливается (`out SREG,r24`);

– обнуление счетного регистра таймера T0;

– счет количества срабатываний прерывания таймера. Регистр r17 постоянно инкрементируется и если его значение не совпадает с уставкой 0x32 (десятичное значение равно 50) (инструкция `cpi r17,0x32`) происходит выход из подпрограммы обработки прерывания (`brne quit`);

– при совпадении значения регистра r17 с уставкой 0x32, т.е. каждые 500 мс счетчика прерываний обнуляется (`clr r17`), считываются данные с порта D (`in r16,PIND`), выполняется сравнение с регистром счета r18 (`cp r16,r18`).

– если уставка, заданная на PIND, меньше, чем текущее значение r18, то r18 обнуляется. После этого значение r18 выводится на PORTC (`out PORTC,r18`) и значение этого регистра инкрементируется;

– в конце подпрограммы вновь разрешаются прерывания (инструкция `sei`) и завершается обработка прерывания (инструкция `reti`) и выполняется возвращение в основную программу.

### Задание на выполнение

1. С использованием таймера Т0 составить программу «бегущий огонь» по вариантам с включением заданных выходов портов, периодами их включения и изменениями в тактах работы.

Варианты программы «бегущий огонь»:

| № вар. | 1 такт | 2 такт | 3 такт | 4 такт |
|---|---|---|---|---|
| 1 | PD0, PD1 – 2 с | PD1, PD2 -1 с | PD2, PD3 -0,5 с | – |
| 2 | PA0…PA3 -0,1 с | PA4…PA7 – 0,2 с | PC0…PC3 -0,3 с | PC4…PC7 – 0,4 с |
| 3 | PC0 – 0,5 с | PC2 – 0,5 с | PC4 – 1,5 с | PC6 – 1,5 с |
| 4 | PB0 – 1 с | Нет – 0,5 с | PB1 – 1 с | Нет – 0,5 с |
| 5 | PC7 – 5 с | PC6, PC7 – 5 с | PC5…PC7 – 5 с | PC4…PC7 – 5 с |
| 6 | PA0 – 2 с | PA1 – 4 с | – | – |
| 7 | PC0…PC3 - 10 с | PC1…PC4 -10 с | PC2…PC5 - 10 с | – |
| 8 | PD7, PC7 -0,5 с | Нет – 2 с | PD0, PC0 - 0,5 с | Нет - 2 с |
| 9 | PC0 – 0,2 с | PC1 – 0,2 с | PC2 – 0,2 с | Нет – 1 с |
| 10 | PA0…PA3 – 0,1 с | Нет – 0,2 с | PA4…PA7 – 0,1 с | Нет – 0,2 с |
| 11 | Порт D – 5 с | Нет – 1 с | Порт C – 5 с | – |
| 12 | PA3 – 2 с | Нет – 4 с | – | – |
| 13 | PC7 – 1 с | Нет – 5 с | PC6 – 1 с | Нет – 5 с |
| 14 | PA0…PA3 – 2 с | Нет – 1 с | PD0…PD3 – 2 с | |
| 15 | PC0…PC3 – 0,1 с | PC1…PC3 -0,1 с | PC2…PC3 – 0,1 с | PC3 – 0,5 с |
| 16 | Порт D – 2 с | Порт A – 2 с | Нет – 4 с | |
| 17 | PC0… PC2 - 2 с | PC1…PC3 - 2 с | PC2…PC4 - 2 с | Нет – 5 с |
| 18 | Порт A -0,1 с | Нет – 1 с | Порт C -0,1 с | Нет – 1 с |
| 19 | PC0 – 0,1 с | PC1 – 0,1 с | PC2 – 0,1 с | Нет – 0,5 с |
| 20 | PB0 – 5 с | PB1 – 2,5 с | PB2 – 1,25 с | Нет – 5 с |
| 21 | PD7 – 1 с | PD6 – 1 с | PD0…5 – 5 с | – |
| 22 | PA6 – 1 с | PA7 – 4 с | – | – |
| 23 | PC0, PC3 – 2 с | PC1, PC4 – 2 с | Нет – 5 с | – |
| 24 | PD4…PD7 – 2,5 с | Нет – 5 с | PD0…PD3 – 2,5 с | Нет – 5 с |
| 25 | Нет – 1,5 с | Порт C – 0,5 с | Нет – 1,5 с | Порт A – 1,5 с |
| 26 | PB0 – 5 с | Нет – 10 с | – | – |
| 27 | PC6,7 – 2 с | PC4, PC5 – 2 с | PC2…PC3 – 2 с | PC0…PC1 – 4 с |
| 28 | PA0…PA3 – 2 с | PA4…PA7 – 2 с | PA0…PA7 – 2 с | Нет – 6 с |
| 29 | PD0,PD2,PD4 – 5 с | PD1, PD3,PD5 – 5 с | Нет – 10 с | – |
| 30 | PC0 – 1,5 с | PC1 – 3 с | PC2 – 4,5 с | – |

*Примечание*: символ '–' обозначает что такт отсутствует, например, в 1 варианте программа выполняется только за 3 такта.

В отчете привести:

– функциональную схему;

– листинг программы;

– дизассемблированную программу;

– блок-схему алгоритма;

– представить настройку таймера и расчет времени задержки, реализованной на таймере;

– представить таблицу значений стека во время исполнения программы.

**Контрольные вопросы**

1. Сколько таймеров содержит микроконтроллер ATmega8535? Какие из них 8-ми разрядные, 16-ти разрядные?

2.Какие регистры определяют работу таймера T0?

3. Поясните назначение регистра управления в целом?

4. Какие биты изменяют режим работы таймера T0? Режим работы вывода?

5. Как установить коэффициент предделителя таймера T0 равный 256? 6. Объясните назначение регистров TIMSK и TIFR.

8. Как в примере реализуется задержка на 500 мс?

9. Что изменится в программе, если установить задержку по времени равную 1 с? Равную 0,1 с?

## ЛИТЕРАТУРА

1. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR. – СПб.: Наука и техника, 2008.– 544 с.

2. Бродин В. Б. Системы на микроконтроллерах и БИС программируемой логики / В. Б. Бродин, А. В. Калинин.-М.: ЭКОМ, 2002.- (Современная микропроцессорная техника).- 398с.: ил. (Микроконтроллеры MCS-51 - Микроконтроллеры ADUC812 - Микроконтроллеры Atmel).

3. Баранов В. Н. Применение микроконтроллеров AVR: Схемы, алгоритмы, программы / В. Н. Баранов; Фирма "Atmel".-М.: Додэка: Додэка- XXI, 2004.- (Мировая электроника).- 287 с.: ил.

4. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы Atmel. – М.: Издательский дом «Додэка».– 2004.

5. Мортон Дж. Микроконтроллеры AVR. Вводный курс. / Пер. с англ. – М. : «Додэка-XXI», 2006.–272 с.

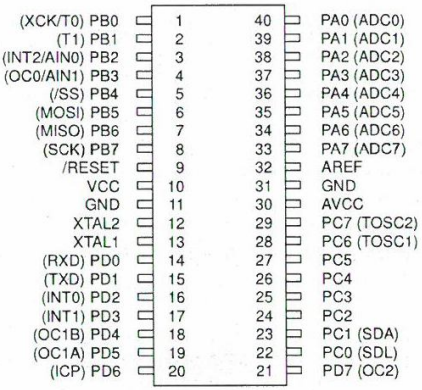
6. Предко М. Руководство по микроконтроллерам: В 2 т./М. Предко; Пер. с англ. Под ред. И. Шагурина, С. Б. Лужанского.- М. : Постмаркет, 2001-.- (Библиотека современной электроники). Т. 1, 2001.- 415 с.: ил.

7. Предко М. Руководство по микроконтроллерам: В 2 т. / М. Предко; Пер. с англ. под ред. И. И. Шагурина, С. Б. Лужанского.- М. : Постмаркет, 2001. (Библиотека современной электроники). Т. 2, 2001.- 487 с.: ил. (Микроконтроллеры PICMICRO, AVR и Basic Stamp).

8. Трамперт В. AVR-RISC микроконтроллеры. / Пер. с нем.–К.: «МК-Пресс», 2006.– 464 с.

## ПРИЛОЖЕНИЕ 1

### Расположение выводов микроконтроллера ATmega8535




*Назначение выводов:*

RESET – сброс микроконтроллера
VCC – напряжение питания
GND – общий провод
XTAL1, XTAL2 – подключение кварцевого резонатора
AVCC – аналоговое питание для АЦП
AREF – внешний источник опорного напряжения для АЦП
PA0...PA7 – Выводы порта A
PB0...PB7 – Выводы порта B
PC0...PC7 – Выводы порта C
PD0...PD7 – Выводы порта D

*Альтернативные функции выводов:*

XCK – внешний тактовый вход интерфейса USART
T0, T1 – входы таймеров T0, T1
OC0, OC1A, OC1B, OC2 – выходы таймеров T0, T1, T2
ICP – вход захвата таймера T1
INT0, INT1, INT2 – входы внешних прерываний
AIN0, AIN1 – входы аналогового компаратора
SS – сетевой режим по интерфейсу SPI
MOSI – выход интерфейса SPI
MISO – вход интерфейса SPI
SCK – тактовый вход интерфейса SPI
RXD, TXD – вход и выход USART
SDA, SDL – линии последовательной передачи данных и тактовых импульсов по шине $I^2C$
TOSC2, TOSC1 – выводы подключение часового резонатора 32768 Гц
ADC0...ADC7 – каналы АЦП

# ПРИЛОЖЕНИЕ 2. Регистры ввода/вывода микроконтроллера ATmega8535

| Адрес | Обозначение | Наименование | Бит 7 | Бит 6 | Бит 5 | Бит 4 | Бит 3 | Бит 2 | Бит 1 | Бит 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $3F ($5F) | SREG | Регистр статуса | I<br>Общее разрешение прерываний | T<br>Хранение копируемого бита | H<br>Флаг полувинного переноса | S<br>Флаг знака | V<br>Флаг переполнения дополнит. кода | N<br>Флаг отриц. значения | Z<br>Флаг нуля | C<br>Флаг переноса |
| $3E ($5E) | SPH | Указатель стека, ст.байт | - | - | - | - | - | SP10 | SP9 | SP8 |
| $3D ($5D) | SPL | Указатель стека,мл.байт | SP7 | SP6 | SP5 | SP4 | SP3 | SP2 | SP1 | SP0 |
| $3C ($5C) | OCR0 | Регистр сравнения T0 | | | | | | | | |
| $3B ($5B) | GICR | Регистр Разрешения внешних прерываний | INT1<br>Разрешение внешнего прерывания INT1 | INT0<br>Разрешение внешнего прерывания INT0 | INT2<br>Разрешение внешнего прерывания INT2 | - | - | - | IVSEL<br>Размещение таблицы прерываний 0-начало Flash,1-загрузчик | IVCE<br>Разрешение изменения положения таблицы прерываний |
| $3A ($5A) | GIFR | Регистр флагов внешних прерываний | INTF1<br>Флаг внешнего прерыв. INT1 | INTF0<br>Флаг внешнего прерыв. INT0 | INTF2<br>Флаг внешнего прерыв. INT2 | - | - | - | - | - |
| $39 ($59) | TIMSK | Регистр маски прерываний таймеров | OCIE2<br>Флаг разрешения прерывания по совпадению T2 | TOIE2<br>Флаг разрешения прерывания по переполнению T2 | TICIE1<br>Флаг разрешения прерывания по захвату T1 | OCIE1A<br>Флаг разрешения прерывания по совпадению T1A | OCIE1B<br>Флаг разрешения прерывания по совпадению T1B | TOIE1<br>Флаг разрешения прерывания по переполнению T1 | OCIE0<br>Флаг разрешения прерывания по совпадению T0 | TOIE0<br>Флаг разрешения прерывания по переполнению T0 |
| $38 ($58) | TIFR | Регистр флагов прерываний таймеров | OCF2<br>Флаг прерывания по совпадению T2 | TOV2<br>Флаг прерыв. по переполнению T2 | ICF1<br>Флаг прерывания по захвату T1 | OCF1A<br>Флаг прерыв. по совпадению T1A | OCF1B<br>Флаг прерывания по совпадению T1B | TOV1<br>Флаг прерыв. по переполнению T1 | OCF0<br>Флаг прерывания по совпадению T0 | TOV0<br>Флаг прерыв. по переполнению T0 |
| $37 ($57) | SPMCR | Регистр управления памятью | SPMIE | RWWSB | – | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN |
| $36 ($56) | TWCR | Регистр управления интерфейсом TWI | TWINT<br>Флаг прерывания | TWEA<br>Бит подтверждения | TWSTA<br>Бит условия старта | TWSTO<br>Бит условия остановки | TWWC<br>Флаг коллизий записи | TWEN<br>Бит разрешения | – | TWIE<br>Разрешение прерывания |
| $35 ($55) | MCUCR | Регистр управления микроконтроллера | SM2<br>Режим SLEEP | SE<br>Разрешение режима SLEEP | SM1<br>Режим SLEEP (SM2:SM1:SM1): 000-Idle, 001- уменьшения шума ADC, 010-PowerDown, 011-PowerSave, 110-Standby, 111-Ext Standby | SM0 | ISC11<br>Условие генерации внешнего прерывания INT1: 00- по низкому уровню, 01- по любому фронту, 10- по отрицательному фронту, 11- по положительному фронту | ISC10 | ISC01<br>Условие генерации внешнего прерывания INT0: 00- по низкому уровню, 01- по любому фронту, 10- по отрицательному фронту, 11- по положительному фронту | ISC00 |
| $34 ($54) | MCUCSR | Регистр статуса и управления микроконтроллера | – | ISC2<br>Чувствительность прерывания INT2 | – | – | WDRF<br>Флаг сброса WatchDog | BORF<br>Флаг сброса Brown-out | EXTRF<br>Флаг внешнего сброса | PORF<br>Флаг сброса питания |
| $33 ($53) | TCCR0 | Регистр управления таймера T0 | FOC0<br>Принудительное изменение состояния выхода OC0 (Normal и CTC) | WGM00<br>Режим работы таймера (WGM01, WGM00): 00-Normal, 01-фазовый ШИМ | COM01<br>Режим работы блока сравнения: 00-таймер отключен от OC0, 01- состояние меняется на противоположное, 10- вывод сбрасыв. в 0,11- вывод сбрасыв. в 1 | COM00 | WGM01<br>Режим работы таймера (WGM01, WGM00): 10-CTC, 11- быстрый ШИМ | CS02 | CS01<br>Управление тактовым сигналом (делитель тактов): 000-таймер остановлен, 001-clk, 010- clk/8, 011-clk/64, 100-clk/256, 101-clk/1024, 110-вывод по отрицательному фронту, 111-вывод по положительному фронту | CS00 |
| $32 ($52) | TCNT0 | Счетный регистр T0 | | | | | | | | |
| $31 ($51) | OSCCAL | Регистр калибровки clk | | | | | | | | |
| $30 ($50) | SFIOR | Регистр специальных функций | ADTS2<br>Источник запуска АЦП: 000- непрерывное преобразования, 001-прерывание от компаратора, 010- INT0, 011- совпадение T0, 100- переполнение T0, 101- совпадение T1, 110- переполнение T1, 111- захват T1 | ADTS1 | ADTS0 | ADHSM<br>Увеличение скорости преобразования АЦП | ACME<br>Доступ к мультиплексору АЦП при работе компаратора (инверсный вход) | PUD<br>Pull-up недоступен (подтягивающие резисторы не подключаются) | PSR2<br>Сброс предделителя таймера T2 | PSR10<br>Сброс предделителя таймеров T0, T1 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $2F ($4F) | TCCR1A | Регистр управления А таймера Т1 | COM1A1 | COM1A0 | COM1B1 | COM1B0 | FOC1A | FOC1B | WGM11 | WGM10 |
| | | | Режим работы блока сравнения А при "Совпадении": 00- таймер отключен от вывода OC1A, 01- состояние меняется на противоположное, 10- вывод сбрасыв. в 0, 11- вывод сбрасыв. в 1 | | Режим работы блока сравнения при "Совпадении": 00- таймер отключен от вывода OC1B, 01- состояние меняется на противоположное, 10- вывод сбрасыв. в 0, 11- вывод сбрасыв. в 1 | | Принудительное изменение состояния выхода OC1A (Normal и CTC) | Принудительное изменение состояния выхода OC1B (Normal и CTC) | Режим работы таймера (вместе с битами WGM13 и WGM12 регистра TCCR1B), режимы Normal, CTC (сброс при совпадении), быстрый и фазовый ШИМ | |
| $2E ($4E) | TCCR1B | Регистр управления В таймера Т1 | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 |
| | | | Включение схемы подавление помех (4 выборки) | Выбор фронта захвата (0- по отриц. Фронту, 1- по полож.) | | Режим работы таймера (вместе с битами WGM11 и WGM10 регистра TCCR1A режимы Normal, CTC (сброс при совпадении), PWM (ШИМ) | | Управление тактовым сигналом (делитель тактов): 000- таймер остановлен, 001-clk, 010- clk/8, 011-clk/64, 100-clk/256 101-clk/1024, 110-вывод по отриц.фронту, 111-вывод по полож. фронту | | |
| $2D ($4D) | TCNT1H | Счетный регистр Т1 | Старший байт | | | | | | | |
| $2C ($4C) | TCNT1L | Счетный регистр Т1 | Младший байт | | | | | | | |
| $2B ($4B) | OCR1AH | Регистр сравнения А Т1 | Старший байт | | | | | | | |
| $2A ($4A) | OCR1AL | Регистр сравнения А Т1 | Младший байт | | | | | | | |
| $29 ($49) | OCR1BH | Регистр сравнения В Т1 | Старший байт | | | | | | | |
| $28 ($48) | OCR1BL | Регистр сравнения В Т1 | Младший байт | | | | | | | |
| $27 ($47) | ICR1H | Регистр захвата Т1 | Старший байт | | | | | | | |
| $26 ($46) | ICR1L | Регистр захвата Т1 | Младший байт | | | | | | | |
| $25 ($45) | TCCR2 | Регистр управления Т2 | FOC2 | WGM20 | COM21 | COM20 | WGM21 | CS22 | CS21 | CS20 |
| | | | Принудительное изменение состояния выхода OC2 (Normal и CTC) | Режим работы таймера (WGM21, WGM20): 00-Normal, 01-фазовый ШИМ | Режим работы блока сравнения при "Совпадении": 00- таймер отключен от вывода OC0, 01- состояние меняется на противоположное, 10- вывод сбрасывается в 0, 11- вывод сбрасывается в 1 | | Режим работы таймера (WGM21, WGM20): 10-CTC, 11- быстрый ШИМ | Управление тактовым сигналом (делитель тактов): 000- таймер остановлен, 001-clk, 010- clk/8, 011-clk/32, 100-clk/64, 101-clk/128, 110-clk/256, 111-clk/1024 | | |
| $24 ($44) | TCNT2 | Счетный регистр Т2 | | | | | | | | |
| $23 ($43) | OCR2 | Регистр сравнения Т2 | | | | | | | | |
| $22 ($42) | ASSR | Регистр состояния асинхронного режима | – | – | – | – | AS2 | TCN2UB | OCR2UB | TCR2UB |
| | | | | | | | Переключение режима работы: 0- внутренний такт, 1- со кварца TOSC1/TOSC2 | Состояние обновления TCNT2: 0- готов к записи, 1- есть значение | Состояние обновления OCR2: 0- готов к записи, 1- есть значение | Состояние обновления TCCR2: 0- готов к записи, 1- есть значение |
| $21 ($41) | WDTCR | Регистр управления сторожевым таймером | – | – | – | WDTOE | WDE | WDP2 | WDP1 | WDP0 |
| | | | | | | Разрешение изменения сторожевого таймера | Сторожевой таймер включен | Коэффициент деления предделителя: 000-16K, 001-32K, 010- 64K, 011- 128K, 100-256K, 101-512K, 110-1024K, 111-2048K | | |
| $20 ($40) | UBRRH | Регистр скорости передачи данных USART | URSEL | – | – | – | UBRR11 | UBRR10 | UBRR9 | UBRR8 |
| | UCSRC | Регистр В состояния и управления USART | URSEL | UMSEL | UPM1 | UPM0 | USBS | UCSZ1 | UCSZ0 | UCPOL |
| | | | Выбор регистра: 0- UBRRH, 1- UCSRC | Режим работы: 0- асинхронный, 1- синхронный | Контроли и формирование четности 00- выключен, 10- проверка на четность (even), 11- проверка на нечетность (odd) | | Кол-во стоп-битов: 0- 1 стоп-бит, 1- 2 стоп-бита | Формат посылок (вместе с UCSZ2 регистра UCSRB): 00- 5 разрядов, 01-6 разр., 10- 7 разрядов, 11- 8 разрядов | | Полярность тактового сигнала в синхронном режиме |
| $1F ($3F) | EEARH | Регистр адреса EEPROM | – | – | – | – | – | – | – | EEAR8 |
| $1E ($3E) | EEARL | Регистр адреса EEPROM | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 |
| $1D ($3D) | EEDR | Регистр данных EEPROM | | | | | | | | |
| $1C ($3C) | EECR | Регистр управления EEPROM | – | – | – | – | EERIE | EEMWE | EEWE | EERE |
| | | | | | | | Разрешение прерывания EEPROM | Разрешение записи в EPROM | Запись в EEPROM | Чтение из EEPROM |
| $1B ($3B) | PORTA | Регистр данных порта А | PORTA7 | PORTA6 | PORTA5 | PORTA4 | PORTA3 | PORTA2 | PORTA1 | PORTA0 |
| $1A ($3A) | DDRA | Регистр напрев. порта А | DDA7 | DDA6 | DDA5 | DDA4 | DDA3 | DDA2 | DDA1 | DDA0 |
| $19 ($39) | PINA | Выводы порта А | PINA7 | PINA6 | PINA5 | PINA4 | PINA3 | PINA2 | PINA1 | PINA0 |
| $18 ($38) | PORTB | Регистр данных порта В | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $17 ($37) | DDRB | Регистр направ. порта B | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 |
| $16 ($36) | PINB | Выводы порта B | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |
| $15 ($35) | PORTC | Регистр данных порта C | PORTC7 | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 |
| $14 ($34) | DDRC | Регистр направ. порта C | DDC7 | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 |
| $13 ($33) | PINC | Выводы порта C | PINC7 | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 |
| $12 ($32) | PORTD | Регистр данных порта D | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 |
| $11 ($31) | DDRD | Регистр направ. порта D | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 |
| $10 ($30) | PIND | Выводы порта D | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 |
| $0F ($2F) | SPDR | Регистр данных SPI | | | | | | | | |
| $0E ($2E) | SPSR | Регистр состояния SPI | SPIF<br>Флаг прерывания от SPI | WCOL<br>Флаг конфликта записи | – | – | – | – | – | SPI2X<br>Удвоение скорости SPI |
| $0D ($2D) | SPCR | Регистр управления SPI | SPIE<br>Разрешение прерывания от SPI | SPE<br>Включение SPI | DORD<br>Порядок передачи данных | MSTR<br>Выбор режима: 0-Slave, 1-Master | CPOL<br>Полярность тактового сигнала: 0-генер «+», 1- генер «-» | CPHA<br>Фронт тактового сигнала: 0- передний SCK, 1-задний | SPR1 | SPR0<br>Скорость передачи: 00-clk/4, 01- clk/16, 10- clk/64, 11- clk/128 |
| $0C ($2C) | UDR | Регистр данных USART | | | | | | | | |
| $0B ($2B) | UCSRA | Регистр A состояния и управления USART | RXC<br>Флаг завершения приема | TXC<br>Флаг завершения передачи | UDRE<br>Регистр передатчика пуст | FE<br>Флаг ошибки кадрирования | DOR<br>Флаг переполнения приемника | PE<br>Флаг ошибки контроля четности | U2X<br>Флаг удвоения скорости передачи | MPCM<br>Мультиплексорный обмен |
| $0A ($2A) | UCSRB | Регистр B состояния и управления USART | RXCIE<br>Разрешение прерывания по завершению приема | TXCIE<br>Разрешение прерывания по завершению передачи | UDRIE<br>Разрешение прерывания при очистке регистра данных | RXEN<br>Разрешение приема | TXEN<br>Разрешение передачи | UCSZ2<br>Формат посылок: 0- 5...8 разрядов, 1- 9-разрядов | RXB8<br>8- разряд принимаемых данных | TXB8<br>8- разряд передаваемых данных |
| $09 ($29) | UBRRL | Регистр скорости передачи данных USART | Младший байт | | | | | | | |
| $08 ($28) | ACSR | Регистр состояния и управления аналогового компаратора | ACD<br>Выключение компаратора | ACBG<br>Подключение к неинвертирующему входу внутр. ИОН | ACO<br>Выход компаратора | ACI<br>Флаг прерывания от компаратора | ACIE<br>Разрешение прерывания от компаратора | ACIC<br>Подключение компаратора к схеме захвата таймера T1 | ACIS1 | ACIS0<br>Условие возникновение прерывания от компаратора: 00- любое изменение выхода, 10- изм. с 1 на 0, 11- изм. с 0 на 1 |
| $07 ($27) | ADMUX | Регистр управления мультиплексором АЦП | REFS1 | REFS0<br>Выбор источника опорного напряжения: 00- внешний, подключенный к AREF, 01 – внешний, подключенный к AVCC, 11– внутренний 2,56В с внешним конденсатором | ADLAR<br>Выравнивание результата преобразования (1- левое) | MUX4 | MUX3<br>Канал АЦП: 00000...00111 – каналы ADC0...ADC7 с K=1; 01001 – (ADC1-ADC0)*10; 01011 – (ADC1-ADC0)*200; 01101 – (ADC3-ADC2)*10; 01111 – (ADC3-ADC2)*200; 10000...10111 - (ADCn-ADC1); 11000...11101 - (ADCn-ADC1); 11110 – внутренний источник 1,22В; 11111 – 0В | MUX2 | MUX1 | MUX0 |
| $06 ($26) | ADCSRA | Регистр состояния и управления АЦП | ADEN<br>Разрешение работы | ADSC<br>Запуск преобразования | ADATE<br>Режим работы: 0- одиночное преобразование, 1- автозапуск в зависимости от ADTS2...0 (SFIOR) | ADIF<br>Флаг прерывания от компаратора | ADIE<br>Разрешение прерывания от компаратора | ADPS2 | ADPS1 | ADPS0<br>Выбор коэффициента деления предделителя АЦП: 000 - 2, 001-2, 010-4, 011-8, 100-16, 101-32, 110 -64, 111-128 |
| $05 ($25) | ADCH | Регистр данных АЦП | Старший байт | | | | | | | |
| $04 ($24) | ADCL | Регистр данных АЦП | Младший байт | | | | | | | |
| $03 ($23) | TWDR | Регистр данных TWI | | | | | | | | |
| $02 ($22) | TWAR | Регистр адреса TWI | TWA6 | TWA5 | TWA4 | TWA3 | TWA2 | TWA1 | TWA0 | TWGCE |
| $01 ($21) | TWSR | Регистр состояния TWI | TWS7 | TWS6 | TWS5<br>Статус TWI | TWS4 | TWS3 | – | TWPS1 | TWPS0<br>Предделитель TWI: 00 - K=1, 01 – K=4, 10 – K=16, 11 – K=64 |
| $00 ($20) | TWBR | Регистр скорости TWI | TWBR7 | TWBR6 | TWBR5 | TWBR4 | TWBR3 | TWBR2 | TWBR1 | TWBR0 |

## ПРИЛОЖЕНИЕ 3

## ТАБЛИЦА ВЕКТОРОВ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРА ATMega8535

| № вектора прерываний | Адрес | Источник | Примечание |
|---|---|---|---|
| 1 | $000 | RESET | Сброс по выводу RESET и сторожевому таймеру (Hardware Pin, Power-On Reset and Watchdog Reset) |
| 2 | $001 | INTO | Запрос внешнего прерывания 0 (External Interrupt Reguest 0) |
| 3 | $002 | INT1 | Запрос внешнего прерывания 1 (External Interrupt Reguest 1) |
| 4 | $003 | TIMER2 COMP | Совпадение при сравнении таймера/счетчика 2 (Timer/Conter2 Compare Match) |
| 5 | $004 | TIMER2 OVF | Переполнение таймера/счетчика2 (Timer/Conter2 Overflow) |
| 6 | $005 | TIMER1 CAPT | Захват таймера/счетчика1 (Timer/Conter1 Capture Event) |
| 7 | $006 | TIMER1 COMPA | Совпадение А при сравнении таймера/счетчика 1 (Timer/Conter1 Compare Match A) |
| 8 | $007 | TIMER1 COMPB | Совпадение В при сравнении таймера/счетчика 1 (Timer/Conter1 Compare Match B) |
| 9 | $008 | TIMER1 OVF | Переполнение таймера/счетчика1 (Timer/Conter1 Overflow) |
| 10 | $009 | TIMER0 OVF | Переполнение таймера/счетчика0 (Timer/Conter0 Overflow) |
| 11 | $00A | SPI, STC | Завершение пересылки SPI (SPI Serial Transfer Complete) |
| 12 | $00B | USART, RX | Завершение приема USART (UART, Rx Complete) |
| 13 | $00C | USART, UDRE | Регистр данных USART пуст (UART Data Register Empty) |
| 14 | $00D | USART, TX | Завершение передачи USART (USART, Tx Complete) |
| 15 | $00E | ADC | Завершение ADC преобразования (ADC Conversion Complete) |
| 16 | $00F | EE_RDY | Готовность EEPROM (EEPROM Ready) |
| 17 | $010 | ANA_COMP | Срабатывание аналогового компаратора (Analog Comparator) |
| 18 | $011 | TWI | Последовательный двухпроводной интерфейс Two-wire Serial Interface |
| 19 | $012 | INT2 | Внешнее прерывание External Interrupt Request 2 |
| 20 | $013 | TIMER0 COMP | Совпадение Р при сравнении таймера/счетчика Т0 Timer/Counter0 Compare Match |
| 21 | $014 | SPM_RDY | Готовность Store Program. Memory Ready |

# ПРИЛОЖЕНИЕ 4
# АССЕМБЛЕР МИКРОКОНТРОЛЛЕРОВ AVR

## П4.1. Общая информация

Здесь представлена информация по ассемблеру всей серии AVR, т.к. все микроконтроллеры этой серии программно совместимы.

Ассемблер – это инструмент, с помощью которого создаётся программа для микроконтроллера. Ассемблер транслирует ассемблируемый исходный код программы в объектный код, который может использоваться в симуляторах или эмуляторах AVR. Также ассемблер генерирует код, который может быть непосредственно введен в программную память микроконтроллера.

При работе с ассемблером нет никакой необходимости в непосредственном соединении с микроконтроллером.

Исходный файл, с которым работает ассемблер, должен содержать мнемоники, директивы и метки.

Перед каждой строкой программы можно ставить метку, которая является алфавитно-цифровой строкой, заканчивающейся двоеточием. Метки используются как указания для безусловного перехода и команд условного перехода.

Строка программы может быть в одной из четырёх форм:

```
[ Метка:] директива [операнды] [Комментарий]
[ Метка:] команда   [операнды] [Комментарий]
Комментарий
Пустая строка
```

*Примечание:* Символы квадратной скобки `[…]` означают, что использование элемента необязательно.

Комментарий имеет следующую форму:

```
; [Текст]
```

Таким образом любой текст после символа " ; " игнорируется ассемблером и имеет значение только для пользователя.

Операнды можно задавать в различных форматах:

- десятичный: 10, 255
- шестнадцатеричный (два способа): 0x1a или $1a
- двоичный: 0b00001010, 0b11111111
- восьмеричный: 010, 077

## П4.2. Система команд

Система команд микроконтроллеров ATMEL семейства AVR очень большая и в то же время эффективная. Одной из отличительных особенностей микроконтроллеров AVR является то, что почти все команды выполняются за 1 тактовый цикл. Исключение составляют команды перехода. Это существенно увеличивает производительность микроконтроллера даже при относительно невысокой тактовой частоте.

Все команды можно классифицировать на 5 типов:

1. Арифметические команды.

2. Логические команды.
3. Команды перехода.
4. Команды передачи данных.
5. Побитовые команды и команды тестирования битов.

Система команд приведена в прил. 5.

### П4.3. Директивы ассемблера

Ассемблер поддерживает множество директив. Директивы не транслируются непосредственно в коды операции. Напротив, они используются, чтобы корректировать местоположение программы в памяти, определять макрокоманды, инициализировать память и так далее. То есть это указания самому ассемблеру, а не команды микроконтроллера.

Все директивы ассемблера приведены в табл. 1.

Таблица 1. Директивы ассемблера

| Директива | Описание |
|---|---|
| BYTE | Зарезервировать байт под переменную |
| CSEG | Сегмент кодов |
| DB | Задать постоянным(и) байт(ы) в памяти |
| DEF | Задать символическое имя регистру |
| DEVICE | Задать для какого типа микроконтроллера компилировать |
| DSEG | Сегмент данных |
| DW | Задать постоянное(ые) слово(а) в памяти |
| EQU | Установите символ равный выражению |
| ESEG | Сегмент EEPROM |
| EXIT | Выход из файла |
| INCLUDE | Включить исходный код из другого файла |
| LIST | Включить генерацию .lst - файла |
| NOLIST | Выключить генерацию .lst - файла |
| ORG | Начальный адрес программы |
| SET | Установите символ равный выражению |

Синтаксис всех директив следующий:

```
.<директива>
```

То есть перед директивой должна стоять точка. Иначе ассемблер воспринимает это как метку.

Поясним наиболее важные директивы ассемблера.

***CSEG** - Code segment*

Директива CSEG указывает на начало сегмента кодов. Ассемблируемый файл может иметь несколько кодовых сегментов, которые будут объединены в один при ассемблировании.

Синтаксис:

```
.CSEG
```

Пример:

```
.DSEG                ; Начало сегмента данных
vartab: .BYTE 4      ; Резервируется 4 байта в СОЗУ
.CSEG                ; Начало сегмента кодов
const:  .DW 2        ; Записать 0x0002  в программной памяти
mov r1,r0            ; Что-то делать
```

***DSEG** - Data Segment*

Директива DSEG указывает на начало сегмента данных. Ассемблируемый файл может содержать несколько сегментов данных, которые потом будут собраны в один при ассемблировании. Обычно сегмент данных состоит лишь из директив BYTE и меток.

Синтаксис:

```
.DSEG
```

Пример:

```
.DSEG                    ; Начало сегмента данных
var1: .BYTE 1            ; Резервировать 1 байт под переменную
table: .BYTE tab_size    ; Резервировать tab_size байтов.
.CSEG
ldi r30,low(var1)
ldi r31,high(var1)
ld r1,Z
```

***ESEG** - EEPROM Segment*

Директива ESEG указывает на начало сегмента EEPROM памяти. Ассемблируемый файл может содержать несколько EEPROM сегментов, которые будут собраны в один сегмент при ассемблировании. Обычно сегмент EEPROM состоит из DB и DW директив (и меток). Сегмент EEPROM памяти имеет свой собственный счетчик. Директива ORG может использоваться для размещения переменных в нужной области EEPROM.

Синтаксис:

```
.ESEG
```

Пример:

```
.DSEG                    ; Начало сегмента данных
var1:.BYTE 1             ; Резервировать 1 байт под переменную
table: .BYTE tab_size    ; Зарезервировать  tab_size байт.
.ESEG
eevar1: .DW 0xffff       ; Записать 1 слово в EEPROM
```

***ORG** - Установить адрес начала программы*

Директива ORG присваивает значения локальным счетчикам. Используется только совместно с директивами .CSEG, .DSEG, .ESEG.

Синтаксис:

```
.ORG  <адрес>
```

Пример:

```
.DSEG              ; Начало сегмента данных
.ORG 0x37          ; Установить адрес СОЗУ  на 37h
```

```
variable: .BYTE 1    ; Зарезервировать байт СОЗУ по адресу 37h
.CSEG
.ORG 0x10            ; Установить счетчик команд на адрес 10h
mov r0,r1            ; Чего-нибудь делать
```

***DB** - определить байт(ы) в программной памяти или в EEPROM*

Директива DB резервирует ресурсы памяти в программной памяти или в EEPROM. Директиве должна предшествовать метка. DB задает список выражений, и должна содержать по крайней мере одно выражение. Размещать директиву следует в сегменте кодов или в EEPROM сегменте.

Список выражений представляет собой последовательность выражений, разделенных запятыми. Каждое выражение должно быть величиной между –128 и 255.

Если директива указывается в сегменте кодов и список выражений содержит более двух величин, то выражения будут записаны так, что 2 байта будут размещаться в каждом слове Flash-памяти.

Синтаксис:

```
LABEL:  .DB <список выражений>
```

Пример:

```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
const2: .DB 1, 2, 3
```

***DW** – Определить слово(а) в программной памяти или в EEPROM*

Директива DW резервирует ресурсы памяти в программной памяти или в EEPROM. Директиве должна предшествовать метка. DW задает список выражений, и должна содержать по крайней мере одно выражение. Размещать директиву следует в сегменте кодов или в EEPROM сегменте.

Список выражений представляет собой последовательность выражений, разделенных запятыми. Каждое выражение должно быть величиной между –32768 и 65535.

Синтаксис:

```
LABEL: .DW список выражений
```

Пример:

```
.CSEG
varlist:  .DW 0, 0xffff, 0b1001110001010101, -32768, 65535
.ESEG
eevarlst: .DW 0,0xffff,10
```

***DEF** – Присвоить имя регистру*

Директива DEF позволяет присвоить символическое имя регистру. Регистр может иметь несколько символических имен.

Синтаксис:

```
.DEF <Имя>=<Регистр>
```

Пример:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
ldi temp,0xf0        ; Загрузить 0xf0 в регистр temp
in ior,0x3f          ; Прочитать SREG в регистр ior
eor temp, ior        ; Выполнить операцию
```

***EQU** – Присвоить имя выражению*

Директива EQU присваивает значение метке. Эта метка может быть использована в других выражениях. Значение этой метки нельзя изменить или переопределить.

Синтаксис:

```
.EQU <метка>=<выражение>
```

Пример:

```
.EQU io_offset = 0x23
.EQU porta     = io_offset + 2
.CSEG                          ; Начало сегмента кодов
clr r2                         ; Очистить регистр r2
out porta,r2                   ; Записать в порт A
```

***INCLUDE** –вставить другой файл*

Директива INCLUDE говорит Ассемблеру начать читать из другого файла. Ассемблер будет ассемблировать этот файл до конца файла или до директивы EXIT. Включаемый файл может сам включать директивы INCLUDE.

Синтаксис:

```
.INCLUDE "<имя файла>"
```

Пример:

```
; iodefs.asm:
.EQU sreg = 0x3f     ; Регистр статуса
.EQU sphigh = 0x3e   ; Старший байт указателя стека.
.EQU splow  = 0x3d   ; Младший байт указателя стека.
; incdemo.asm
.INCLUDE "iodefs.asm"; Включить файл «iodefs.asm»
in r0,sreg           ; Прочитать регистр статуса
```

***EXIT** – выйти из файла*

Директива EXIT позволяет ассемблеру остановить ассемблирование текущего файла. Обычно ассемблер работает до конца файла. Если он встретит директиву EXIT, то продолжит ассемблировать со строки, следующей за директивой INCLUDE.

Синтаксис:

```
.EXIT
```

Пример:

```
.EXIT                ; выйти из этого файла
```

***DEVICE*** *– Указать для какого микроконтроллера ассемблировать*

Директива позволяет пользователю сообщить ассемблеру, для какого типа устройства пишется программа. Если ассемблер встретит команду, которая не поддерживается указанным типом микроконтроллера, то будет выдано сообщение. Также сообщение появится в случае, если размер программы превысит объем имеющейся в этом устройстве памяти.

Синтаксис:

```
.DEVICE AT90S1200 |AT90S2313 | AT90S2323 | AT90S2333 | AT90S2343 |
AT90S4414 | AT90S4433 | AT90S4434 | AT90S8515 | AT90S8534 |
AT90S8535 | ATtiny11 | ATtiny12 | ATtiny22 | ATmega603 |
ATmega103| Atmega8535
```

Пример:

```
.DEVICE ATmega8535  ;использовать ATmega8535
.CSEG
.ORG 0000
jmp label1          ;При ассемблировании появиться сообщение, что
                    ;ATmega8535 не поддерживает команду jmp
```

П4.4. Некоторые особенности программирования

Память данных почти полностью доступна программе пользователя и большинство команд ассемблера предназначено для обмена данными с ней. Команды пересылки данных предоставляют возможность непосредственной и косвенной адресации ячеек СОЗУ, непосредственной адресации регистров ввода/вывода и регистров общего назначения. Так как каждому регистру сопоставлена ячейка памяти, то обращаться к ним можно не только командами адресации регистров, но и командами адресации ячеек СОЗУ.

Например, команда:

```
MOV R10,R15     ; скопировать регистр R15 в регистр R10 делает
                ; абсолютно то же самое, что и команда:
LDS R10,$0015   ; загрузить в регистр R10 содержимое ячейки с
                ; адресом $0015
```

То же самое относится и к регистрам ввода/вывода. Для них предусмотрены специальные команды:

```
IN Rd,P         ; загрузить данные из порта I/O с номером P
                ; в регистр Rd
OUT P,Rd        ; записать данные из регистра Rd в порт I/O
                ; с номером P.
```

При использовании этих команд номер порта указывается в диапазоне 0<P<63. При использовании команд адресации ячеек памяти для работы с регистрами ввода/вывода указывается адрес регистра в памяти данных $0020-$005F.

Пример применения разных команд:

```
LDI R16,$FF
OUT $12,R16     ; записать в PORTD число 255
STS $0032,R16   ; записать непосредственно в ячейку $0032
```

Адрес регистра ввода/вывода в СОЗУ получается прибавлением к номеру порта числа $20.

Памятью программ является ПЗУ и изменяется только при программировании кристалла. Константы можно располагать в памяти программ в виде слов.

Например:

```
.dw  $033f,$676d,$7653,$237e,$777f
```

Для работы с данными, расположенными в памяти программ, предусмотрена команда

LPM – загрузить байт памяти программ, на который указывает регистр Z в регистр R0.

Адрес байта константы определяется содержимым регистра Z. Старшие 15 битов определяют слово адреса (от 0 до 4к) состояние младшего бита определяет выбор младшего байта (0) или старшего байта (1).

При работе с портами ввода/вывода следует учитывать следующую особенность. Если вывод порта сконфигурирован как выход, то его переключение производится через регистр данных (PORTA, PORTB, PORTC, PORTD), если вывод сконфигурирован как вход, то его опрос следует производить через регистр выводов входа порта (PINA, PINB, PINC, PIND).

Особенностью использования арифметических и логических команд является то, что некоторые из них работают только с регистрами R16–R31.

Пример:

`CPI Rd, K` – сравнить регистр Rd с константой K, 16<d<31.

Команды CBI и SBI работают только с младшими 32-мя регистрами ввода/вывода.

При использовании подпрограмм нужно обязательно определять стек! Для этого нужно занести значения адреса вершины стека в регистры SPH и SPL.

П4.5. Написание программы

Программа, написанная на ассемблере, должна иметь определенную структуру. Предлагается следующий шаблон (для ATmega8535)

```
;-------------------------------------------------;
; название программы,                             ;
; краткое описание, необходимые пояснения         ;
:-------------------------------------------------;
;       подключаемые дополнительные файлы
.include "m8535def.inc"             ; файл описания ATmega8535
.include "<имя_файла1.расширение>" ; включение дополнительных
.include "<имя_файла2.расширение>" ; файлов
;       глобальные константы
.equ      <имя1> = <константа1>
.equ      <имя2> = <константа2>
;       глобальные регистровые переменные
.def      <имя1>= <регистр1>
.def      <имя2>= <регистр2>
;       сегмент данных
.dseg
.org <адрес>          ; адрес первого зарезервированного байта
label1:  .BYTE 1      ; резервировать 1 байт под переменную label1
label2:  .BYTE m      ; резервировать m байт под переменную label2
;       сегмент EEPROM (ЭСППЗУ)
```

```
.eseg
.org <адрес>          ; адрес первого зарезервированного байта
.db  выражение1,выражение2,…  ; записать список байтов в EEPROM
.dw  выражение1,выражение2,…  ; записать список слов в EEPROM
;       сегмент кодов
.cseg
.org $0000          ; адрес начала программы в программной памяти
;       вектора прерываний (если они используются)
rjmp reset          ;прерывание по сбросу
.org $0002
rjmp INT0           ;обработчик прерывания INT0
.org $0004
rjmp INT1           ;обработчик прерывания INT1
.org adrINTx        ;адрес следующего обработчика прерываний
rjmp  INTx          ;обработчик прерывания x
…….                 ;далее располагаются обработчики остальных
                    ;прерываний
;       начало основной программы
main:     <команда> xxxx
            …     …
;       подпрограмма 1
subr1: <команда> xxxx
……        ……….         ……
            ret
;       подпрограмма 2
subr2: <команда> xxxx
……        ……….         ……
            ret
;       программы обработчиков прерываний
INT0: <команда> xxxx
……        ……….         ……
            reti
……….
;       конец программы никак не обозначается
```

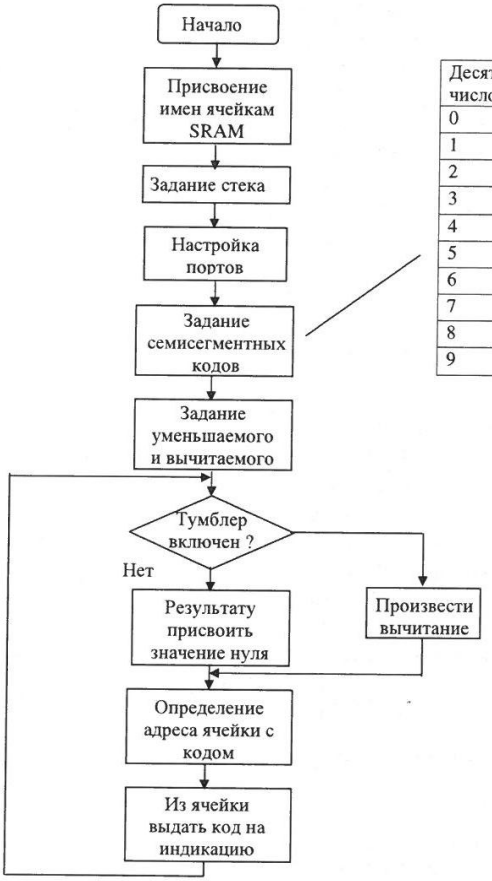
### П4.6. Пример программы

Ниже приводятся простейшая программа, демонстрирующая использование директив ассемблера. Программа вычитает из числа 5 число 3. Если подан сигнал разрешения («1» на вход PA4), то на индикацию (сегменты – биты PC0…PC7, индикатор – бит PB3) выдать результат вычитания. Если нет разрешения, то на индикацию вывести цифру ноль. На рис. 1 приведен алгоритм программы.

```
; Программа №1
.include "m8535def.inc"  ;включить файл - описание для ATmega8535
.dseg                    ;сегмент данных
.equ      cod0=$64       ;присвоение имен ячейкам SRAM
.equ      cod1=$65
.equ      cod2=$66
.equ      cod3=$67
.equ      cod4=$68
.equ      cod5=$69
.equ      cod6=$6a
.equ      cod7=$6b
```

```
.equ      cod8=$6c
.equ      cod9=$6d

.cseg
.org      0
rjmp      reset
.org  $30           ;начало программы
reset:
ldi   r16,$00       ;определение стека с вершиной по адресу $00ff
out   sph,r16
ldi   r16,$ff
out   spl,r16
```




| Десятичное число | Код |
|---|---|
| 0 | 3f |
| 1 | 06 |
| 2 | 5b |
| 3 | 4f |
| 4 | 66 |
| 5 | 6d |
| 6 | 7d |
| 7 | 07 |
| 8 | 7f |
| 9 | 6f |

Рис. 1. Алгоритм программы

```
ldi   zl,$64          ;задание адреса начала зарезервированных ячеек
ldi   zh,$00

ldi   r16,$ff         ;настроить порт C на выход
out   ddrc,r16
ldi   r16,00          ;настроить порт A на вход
out   ddra,r16
ldi   r16,$c          ;настроить порт B: биты 2 и 3 на выход, остальные на вход
out   ddrb,r16
ldi   r16,$f0         ;настроить порт D: биты 0...4 на вход, остальные на выход
out   ddrd,r16

sbi   portB,3         ;выдать 1 на разряд 3 порта B
ldi   r17,$3f         ;задание семисегментных кодов
sts   cod0,r17
ldi   r17,$06
sts   cod1,r17
ldi   r17,$5b
sts   cod2,r17
ldi   r17,$4f
sts   cod3,r17
ldi   r17,$66
sts   cod4,r17
ldi   r17,$6d
sts   cod5,r17
ldi   r17,$7d
sts   cod6,r17
ldi   r17,$07
sts   cod7,r17
ldi   r17,$7f
sts   cod8,r17
ldi   r17,$6f
sts   cod9,r17

ldi   r17,5           ;задание уменьшаемого
ldi   r18,3           ;задание вычитаемого
m1:   sbis pina,4     ;если включен тумблер SA1,то пропустить
rjmp      m2          ;следующую команду
mov   r20,r17         ; в r20 поместить уменьшаемое
sub   r20,r18         ; вычесть вычитаемое
rjmp      vv
m2:   ldi  r20,0
vv:   push zl         ;сохранить zl в стеке
add   zl,r20          ;сложить zl с результатом
ld    r0,z            ;семисегментный код результата переслать в r20
pop    zl             ;извлечь zl из стека
out   portc,r0        ;выдать результат на индикацию
rjmp      m1
```

## ПРИЛОЖЕНИЕ 5

## СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ AVR

Арифметические и логические команды

| Мнемо-ника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| ADD | Rd,Rr<br>0≤d≤31<br>0≤r≤31 | Сложить без переноса | Rd ←Rd + Rr | Z, C, N, V, H | 1 |
| ADC | Rd.Rr<br>0≤d≤31<br>0≤r≤31 | Сложить с переносом | Rd ← Rd + Rr+ C | Z, C, N, V, H | 1 |
| ADIW | Rd,K<br>dE{24,26,28, 30}<br>0≤K≤63 | Сложить непосредственное значение со словом | Rdh:Rdl←Rdh:Rdl+ K | Z, C, N, V | 2 |
| SUB | Rd,Rr<br>0≤d≤31<br>0≤r≤31 | Вычесть без заема | Rd ← Rd – Rr | Z, C, N, V, H | 1 |
| SUBI | Rd, K<br>16≤d≤31<br>0≤K≤255 | Вычесть непосредственное значение | Rd ← Rd – K | Z, C, N, V, H | 1 |
| SBC | Rd, Rr<br>0≤d≤31<br>0≤r≤31 | Вычесть с заемом | Rd←Rd-Rr-C | Z, C, N, V, H | 1 |
| SBCI | Rd, K<br>16≤d≤32<br>0≤K≤255 | Вычесть непосредственное значение с заемом | Rd←Rd-K-C | Z, C, N, V, H | 1 |
| SBIW | Rd, K<br>dE{24,26,28, 30} O≤K≤63 | Вычесть непосредственное значение из слова | Rdh:Rdl←Rdh:Rdl- K | Z, C, N, V | 2 |
| AND | Rd, Rr<br>0≤d≤31<br>0≤r≤31 | Выполнить логическое AND | Rd ← Rd • Rr | Z, N, V | 1 |
| ANDI | Rd, K<br>16<d<31<br>0<k≤255 | Выполнить логическое AND | Rd ← Rd • K | Z, N, V | 1 |
| OR | Rd, Rr<br>0≤d≥31<br>0≤r≤31 | Выполнить логическое OR | Rd ←Rd v Rr | Z, N, V | 1 |
| ORI | Rd, K<br>16≤d≤31<br>0≤K≤255 | Выполнить логическое OR с непосредственным значением | Rd ←Rd v K | Z, N, V | 1 |

| Мнемо-ника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| EOR | Rd, Rr<br>0≤d≤31<br>0≤r≤31 | Выполнить исключающее OR | Rd ←Rd Rr | Z, N, V | 1 |
| COM | Rd<br>0≤d≤31 | Выполнить дополнение до единицы | Rd←SFF-Rd | Z, C, N, V | 1 |
| NEG | Rd<br>0≤d≤31 | Выполнить дополнение до двух | Rd ← S00 - Rd | Z, C, N, V, H | 1 |
| SBR | Rd, K<br>16≤d≤31<br>0≤K≤255 | Установить биты в регистре | Rd ← Rd v K | Z, N,V | 1 |
| CBR | Rd, K<br>16≤d≤31<br>O≤K≤255 | Очистить биты в регистре | Rd ←Rd • (SFF - K) | Z, N, V | 1 |
| INC | Rd<br>0≤d≤31 | Инкрементировать | Rd ←Rd + 1 | Z, N,V | 1 |
| DEC | Rd<br>0≤d≤31 | Декрементировать | Rd ← Rd - 1 | Z, N,V | 1 |
| TST | Rd<br>0≤r≤31 | Проверить на ноль или минус | Rd←Rd.Rd | Z, N, V | 1 |
| CLR | Rd<br>0≤d≤31 | Очистить регистр | Rd ← Rd ⊕ Rd | Z, N, V | 1 |
| SER | Rd<br>16≤d≤31 | Установить все биты регистра | Rd ←SFF | нет | 1 |
| CP | Rd, Rr<br>0≤d≤31<br>0≤r≤31 | Сравнить регистры | Rd-Rr | Z, C, N, V, H | 1 |
| CPC | Rd, Rr<br>0≤d≤31<br>0≤r≤31 | Сравнить регистры с учетом переноса | Rd-Rr-C | Z, C, N, V, H | 1 |
| CPI | Rd, K<br>16≤d≤31<br>0≤K≤255 | Сравнить с константой | Rd-K | Z, C, N, V, H | 1 |

Команды сдвигов и операций с битами

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| LSL | Rd<br>0≤d≤31 | Логически сдвинуть влево | Rd(n+1)←Rd(n),<br>Rd(0)←0,C←Rd(7) | Z,C,N,V,H | 1 |
| LSR | Rd<br>0≤d≤31 | Логически сдвинуть вправо | Rd(n)←Rd(n+1),<br>Rd(7)←0,C←Rd(0) | Z,C,N,V | 1 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| ROL | Rd<br>0≤d≤31 | Сдвинуть влево через перенос | Rd(0) ←C,<br>Rd(n+1)←Rd(n),<br>C←Rd(7) | Z,C,N,V,H | 1 |
| ROR | Rd<br>0≤d≤31 | Сдвинуть вправо через перенос | Rd(7) ←C,<br>Rd(n)←Rd(n+1),<br>C ← Rd(0) | Z,C,N,V | 1 |
| ASR | Rd<br>0≤d≤31 | Арифметически сдвинуть вправо | Rd(n)←Rd(n+1),<br>n=0...6, Rd(0)←C | Z,C,N,V | 1 |
| SWAP | Rd 0≤d≤31 | Поменять нибблы местами | Rd(3...0) <–<br>>Rd(7...4) | Нет | 1 |
| BSET | s<br>0≤s≤7 | Установить флаг | SREG(s)← 1 | SREG(s) | 1 |
| BCLR | s<br>0≤s≤7 | Очистить флаг | SREG(s)← 0 | SREG(s) | 1 |
| SBI | P,b<br>O≤P≤31<br>0≤b≤7 | Установить бит в регистр I/O | I/0(P,b)← 1 | Нет | 2 |
| CBI | P,b<br>0≤P≤31<br>0≤b≤7 | Очистить бит в регистре I/O | I/0(P,b)← 0 | Нет | 2 |
| BST | Rd,b<br>0≤d≤31<br>0≤b≤7 | Переписать бит из регистра во флаг T | T← Rd(b) | T | 1 |
| BLD | Rd,b<br>0≤d≤31<br>0≤b≤7 | Загрузить T флаг в бит регистра | Rd(b) ← T | Нет | 1 |
| SEC | | Установить флаг переноса | C← 1 | C | 1 |
| CLC | | Очистить флаг переноса | C←0 | C | 1 |
| SEN | | Установить флаг отрицательного значения | M←1 | N | 1 |
| CLN | | Очистить флаг отрицательного значения | N←0 | N | 1 |
| SEZ | | Установить флаг нулевого значения | Z←1 | Z | 1 |
| CLZ | | Очистить флаг нулевого значения | Z←0 | Z | 1 |
| SEI | | Установить флаг глобального прерывания | I← 1 | I | 1 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| CLI | | Очистить флаг глобального прерывания | I←0 | I | 1 |
| SES | | Установить флаг знака | S← 1 | S | 1 |
| CLS | | Очистить флаг знака | S←0 | S | 1 |
| SEV | | Установить флаг переполнения | V← 1 | V | 1 |
| CLV | | Очистить флаг переполнения | V←0 | V | 1 |
| SET | | Установить флаг T | T←1 | T | 1 |
| CLT | | Очистить флаг T | T←0 | T | 1 |
| SEH | | Установить флаг полупереноса | H← 1 | H | 1 |
| CLH | | Очистить флаг полупереноса | H←0 | H | 1 |
| NOP | | Выполнить холостую команду | | Нет | 1 |
| SLEEP | | Установить режим SLEEP | | Нет | 1 |
| WDR | | Сбросить сторожевой таймер | | Нет | 1 |

Команды пересылки данных

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| ELPM | | Расширенная загрузка из памяти программ в регистр RO | R0← (Z+RAMPZ) | Нет | 3 |
| MOV | Rd,Rr<br>0≤d≤31<br>0≤r≤31 | Копировать регистр | Rd←Rr | Нет | 1 |
| LDI | Rd,k<br>16≤d≤31<br>0≤k≤255 | Загрузить непосредственное значение | Rd←K | Нет | 1 |
| LDS | Rd,k<br>0≤d≤31<br>0≤k≤65535 | Загрузить из ОЗУ | Rd←(k) | Нет | 3 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| LD | Rd, X<br>0≤d≤31 | Загрузить косвенно | Rd ← (X) | Нет | 2 |
| LD | Rd, X+<br>0≤d≤31 | Загрузить косвенно с постинкрементом | Rd ← (X),<br>X←X+1 | Нет | 2 |
| LD | Rd, X–<br>0≤d≤31 | Загрузить косвенно с преддекрементом | X←X-1,<br>Rd ← (X) | Нет | 2 |
| LD | Rd, Y<br>0≤d≤31 | Загрузить косвенно | Rd←(Y), | Нет | 2 |
| LD | Rd, Y+<br>0≤d≤31 | Загрузить косвенно с постинкрементом | Rd←(Y), Y←Y+1 | Нет | 2 |
| LD | Rd, Y<br>0≤d≤31 | Загрузить косвенно с преддекрементом | Y←Y-1, Rd ← (Y) | Нет | 2 |
| LDD | Rd, Y+q<br>0≤d≤31<br>0≤q≤63 | Загрузить косвенно со смещением | Rd←(Y+q) | Нет | 2 |
| LD | Rd, Z<br>0≤d≤31 | Загрузить косвенно | Rd ← (Z) | Нет | 2 |
| LD | Rd, Z+<br>0≤d≤31 | Загрузить косвенно с постинкрементом | Rd ← (Z), Z←Z+1 | Нет | 2 |
| LD | Rd, –Z<br>0≤d≤31 | Загрузить косвенно с преддекрементом | Z←Z-1, Rd←(Z) | Нет | 2 |
| LDD | Rd, Z+q<br>0≤d≤31<br>0≤q≤31 | Загрузить косвенно со смещением | Rd ← (Z+q) | Нет | 2 |
| STS | k, Rr<br>0≤d≤31<br>0≤k≤65535 | Загрузить непосредственно в ОЗУ | (k) ←Rr | Нет | 3 |
| ST | X, Rr<br>0≤r≤31 | Записать косвенно | (X) ← Rr | Нет | 2 |
| ST | X+, Rr<br>0≤r≤31 | Записать косвенно с постинкрементом | (X) ←Rr, X←X+ 1 | Нет | 2 |
| ST | –X, Rr<br>0≤r≤31 | Записать косвенно с преддекрементом | X←X-1, (X) ←Rr | Нет | 2 |
| ST | Y, Rr<br>0≤r≤31 | Записать косвенно | (Y) ←Rr | Нет | 2 |
| ST | Y+, Rr<br>0≤r≤31 | Записать косвенно с постинкрементом | (Y) ← Rr, Y←Y+ 1 | Нет | 2 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| ST | –Y, Rr<br>0≤r≤31 | Записать косвенно с преддекрементом | Y←Y-1, (Y) ← Rr | Нет | 2 |
| STD | Y+q, Rr<br>0≤r≤31<br>0≤q≤63 | Записать косвенно со смещением | (Y+q)←Rr | Нет | 2 |
| ST | Z, Rr<br>0≤r≤31 | Записать косвенно | (Z) ←Rr | Нет | 2 |
| ST | Z+, Rr<br>0≤r≤31 | Записать косвенно с постинкрементом | (Z) ← Rr, Z←Z+ 1 | Нет | 2 |
| ST | –Z, Rr<br>0≤r≤31 | Записать косвенно с преддекрементом | Z←Z-1, (Z) ← Rr | Нет | 2 |
| STD | Z+q, Rr<br>0≤r≤31<br>0≤q≤63 | Записать косвенно со смещением | (Z+q)←Rr | Нет | 2 |
| LPM | | Загрузить байт из памяти программ | R0 ← (Z) | Нет | 3 |
| IN | Rd, P<br>0≤d≤31<br>0≤P≤63 | Загрузить данные из порта I/O в регистр | Rd←P | Нет | 1 |
| OUT | P, Rr<br>0≤r≤31<br>0≤P≤63 | Записать данные из регистра в порт I/O | P←Rr | Нет | 1 |
| PUSH | Rr<br>0≤r≤31 | Сохранить регистр в стеке | STACK ←Rr | Нет | 2 |
| POP | Rr<br>0≤r≤31 | Загрузить в регистр из стека | Rr ← STACK | Нет | 2 |

Команды переходов

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| RJMP | k<br>-2K<k<2K | Перейти относительно | PC ←PC + k + 1 | Нет | 2 |
| LJMP | | Перейти косвенно | PC←Z | Нет | 2 |
| JMP | k<br>0<k<4M | Перейти | PC←k | Нет | 3 |
| RCALL | k<br>-2K≤k≤2K | Вызвать подпрограмму относительно | PC ← PC + k + 1 | Нет | 3 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| ICALL | | Вызвать подпрограмму косвенно | PC←Z | Нет | 3 |
| CALL | k<br>0≤k≤64K | Выполнить длинный вызов подпрограммы | PC←k | Нет | 4 |
| RET | | Вернуться из подпрограммы | PC ← STACK | Нет | 4 |
| RETI | | Вернуться из прерывания | PC ← STACK | I | 4 |
| CPSE | Rd, Rr<br>0≤d≤31,<br>0≤r≤31 | Сравнить и пропустить, если равно | If Rd=Rr then<br>PC ← PC + 2 (or 3) | Нет | ½/3 |
| SBRC | Rr, b<br>0≤r≤31<br>0≤b≤7 | Пропустить, если бит в регистре очищен | if Rr(b)=0 then<br>PC ← PC + 2 (or 3) | Нет | ½/3 |
| SBRS | Rr, b<br>0≤r≤31<br>0≤b≤7 | Пропустить, если бит в регистре установлен | If Rr(b)=1 then<br>PC ←PC + 2 (or 3) | Нет | ½/3 |
| SBIC | P, b<br>0≤P≤31<br>0≤b≤7 | Пропустить, если бит в регистре I/O очищен | if I/O P(b)=0 then<br>PC ← PC + 2 (or 3) | Нет | ½/3 |
| SBIS | P, b<br>0≤r≤31<br>0≤b≤7 | Пропустить, если бит в регистре I/O установлен | If I/O P(b)=1 then<br>PC ←PC + 2 (or 3) | Нет | ½/3 |
| BRBS | s, k<br>0≤s≤7<br>-64≤k≤+63 | Перейти, если бит в регистре статуса установлен | if SREG(s)=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRBC | s, k<br>0≤s≤7<br>-64≤k≤+63 | Перейти, если бит в регистре статуса очищен | if SREG(s)=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BREQ | k<br>-64≤k≤+63 | Перейти, если равно | if Rd=Rr (Z=1) then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRNE | k<br>-64≤k≤+63 | Перейти, если не равно | if Rd≠Rr(Z=0) then<br>PC<←PC+ k+ 1 | Нет | 1/2 |
| BRCS | k<br>-64≤k≤+63 | Перейти, если флаг переноса установлен | if C=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRCC | k<br>-64≤k≤+63 | Перейти, если флаг переноса очищен | if C=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRSH | k<br>-64≤k≤+63 | Перейти, если равно или больше (без знака) | if Rd<Rr(C=0) then<br>PC←PC+ k+ 1 | Нет | 1/2 |

| Мнемоника | Операнды | Описание | Операция | Флаги | Кол-во циклов |
|---|---|---|---|---|---|
| BRLO | k<br>-64≤k≤+63 | Перейти, если меньше (без знака) | if Rd<Rr (C=1) then<br>PC ←PC + k + 1 | Нет | 1/2 |
| BRMI | k<br>-64≤k≤+63 | Перейти, если минус | if N=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRPL | k<br>-64≤k≤+63 | Перейти, если плюс | if N=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRGE | k<br>-64≤k≤+63 | Перейти, если больше или равно (со знаком) | if Rd>Rr (N⊕V=0)<br>then PC←PC+k+1 | Нет | 1/2 |
| BRLT | k<br>-64≤k≤+63 | Перейти, если меньше чем (со знаком) | if Rd<Rr (N⊕V=1)<br>then PC←PC+k+1 | Нет | 1/2 |
| BRHS | k<br>-64≤k≤+63 | Перейти, если флаг полупереноса установлен | if H=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRHC | k<br>-64≤k≤+63 | Перейти, если флаг полупереноса очищен | if H=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRTS | k<br>-64≤k≤+63 | Перейти, если флаг T установлен | if T=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRTC | k<br>-64≤k≤+63 | Перейти, если флаг T очищен | if T=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRVS | k<br>-64≤k≤+63 | Перейти, если флаг переполнения установлен | if V=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRVC | k<br>-64≤k≤+63 | Перейти, если флаг переполнения очищен | if V=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRIE | k<br>-64≤k≤+63 | Перейти, если глобальное прерывание разрешено | if I=1 then<br>PC ← PC + k + 1 | Нет | 1/2 |
| BRID | k<br>-64k≤k≤+63 | Перейти, если глобальное прерывание запрещено | if I=0 then<br>PC ← PC + k + 1 | Нет | 1/2 |

## ПРИЛОЖЕНИЕ 6

## ОПИСАНИЕ ПРОГРАММЫ «USB ISP»

Программа «USB ISP» позволяет программировать микроконтроллер, считывать программу из микроконтроллера, записывать и читать внутреннее перепрограммируемое ПЗУ микроконтроллера, программировать биты защиты, менять режимы работы генератора тактовых импульсов и режимы старта микроконтроллера. Входным форматом данных для программатора является hex-файл стандарта фирмы Intel (Intel Standard или Intel Extended).

Запустите программу «USB ISP», на экране компьютера появится окно программы (рис. 1).

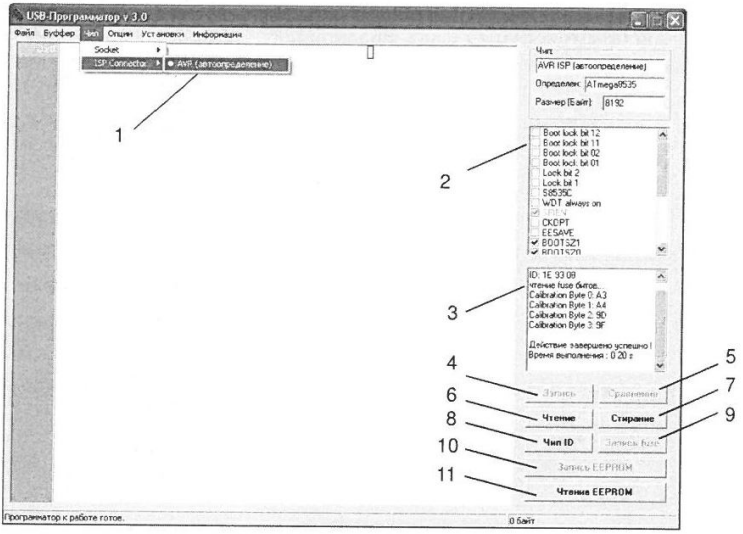



Рис. 1. Окно программатора

Убедитесь в появлении надписи «Программатор к работе готов».

Для начала работы с программой необходимо указать фирму - разработчик микроконтроллера («AVR» в поле 1).

Далее необходимо выбрать файл «*.hex» для прошивки в микроконтроллер (меню файл –> открыть файл).

После этого можно просто нажать кнопку записи (поле 4). При этом будут выполнены все указанные действия по порядку:

– проверка типа контроллера и входа в режим программирования (поле 8);
– стирание микросхемы (поле 7);
– запись программы в микроконтроллер;
– проверка записанной программы (поле 5);
– установка необходимых бит защиты памяти микроконтроллера (поле 9).

При выполнении этих действий будет гореть светодиод «Прогр» на модуле.

В нижней статусной строке будут высвечены результаты прошивки памяти микроконтроллера.

Если требуется более глубокая настройка параметров микроконтроллера при программировании, то в поле 2 имеются дополнительные настройки работы контроллера. Все указанные в этом поле режимы описаны в разделе программирования фирменного описания микроконтроллера ATMega8535 фирмы Atmel (www.atmel.com) и не требуют изменений при программировании микроконтроллера в составе лабораторной установки.

Общий выход из программы осуществляется при нажатии на системную кнопку окна  . При этом появится предупреждение о необходимости отключения программатора. Данное предупреждение предназначено для смены микросхемы при серийном программировании и для лабораторного стенда значения не имеет.