Программирование микроконтроллера Atmega 8535 на ассемблере

Часть 1

Методические указания
к проведению лабораторных работ

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

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

Модуль «Микроконтроллер» 5
ЛАБОРАТОРНЫЕ РАБОТЫ 7
Работа № 1. Знакомство со средой программирования и отладки микроконтроллеров AVR STUDIO 7
Работа № 2. Порты ввода/вывода микроконтроллера Atmega8535 15
Работа № 3. Специальный регистр состояния SREG 22
Работа № 4. Стек. Реализация программной задержки 28
Работа № 5. Таймеры T0 и T2 в режиме подсчета временных интервалов 36
Работа № 6. Аналого-цифровой преобразователь микроконтроллера 46
ПРИЛОЖЕНИЕ А 57
ПРИЛОЖЕНИЕ Б 65

Модуль «Микроконтроллер»

Методические указания написаны для проведения лабораторных работ с использованием модуля «Микроконтроллер», входящего в состав лабораторного стенда.

Модуль «Микроконтроллер» предназначен для программирования и изучения функций микроконтроллера ATmega8535 семейства AVR, выпускаемого фирмой Atmel. Внешний вид модуля приведен на рис. 1.

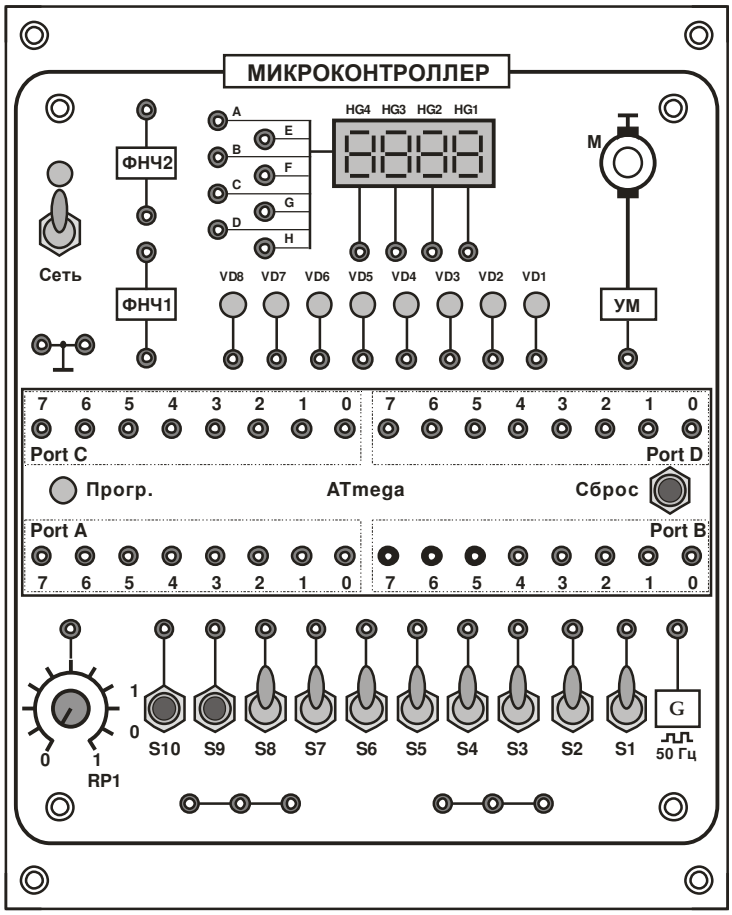


Рис. 1. Внешний вид модуля «Микроконтроллер»

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 М 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.

С тыльной стороны модуля располагается разъем для подачи напряжения ~220В 50 Гц на модуль, а также разъем для подключения модуля к персональному компьютеру по интерфейсу USB.

Табл. 1. Краткая характеристика микроконтроллера ATmega8535

Параметр	**Значение**
Частота установленного кварцевого резонатора	8 МГц
Напряжение электропитания	2,7 – 5,5 В
Объем внутренней Flash – памяти	8 кБайт
Объем энергонезависимой памяти	512 Байт
Объем внутренней ОЗУ	512 Байт
32 программируемых входа/выхода	32 на 4 портах
JTAG – интерфейс	нет
8-битные таймеры/счетчики с ШИМ	2 шт.
16-битный таймер/счетчик с ШИМ	1 шт.
10-разрядный аналогово-цифровой преобразователь	есть
Количество каналов АЦП	8
Аналоговый компаратор	есть
Источники внешних прерываний	3 шт.
Универсальный приемопередатчик USART	есть
SPI – интерфейс	есть
TWI – интерфейс	есть

ЛАБОРАТОРНЫЕ РАБОТЫ

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

Цель работы

Ознакомиться с программной средой программирования и отладки микроконтроллеров фирмы Atmel AVR Studio. 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).

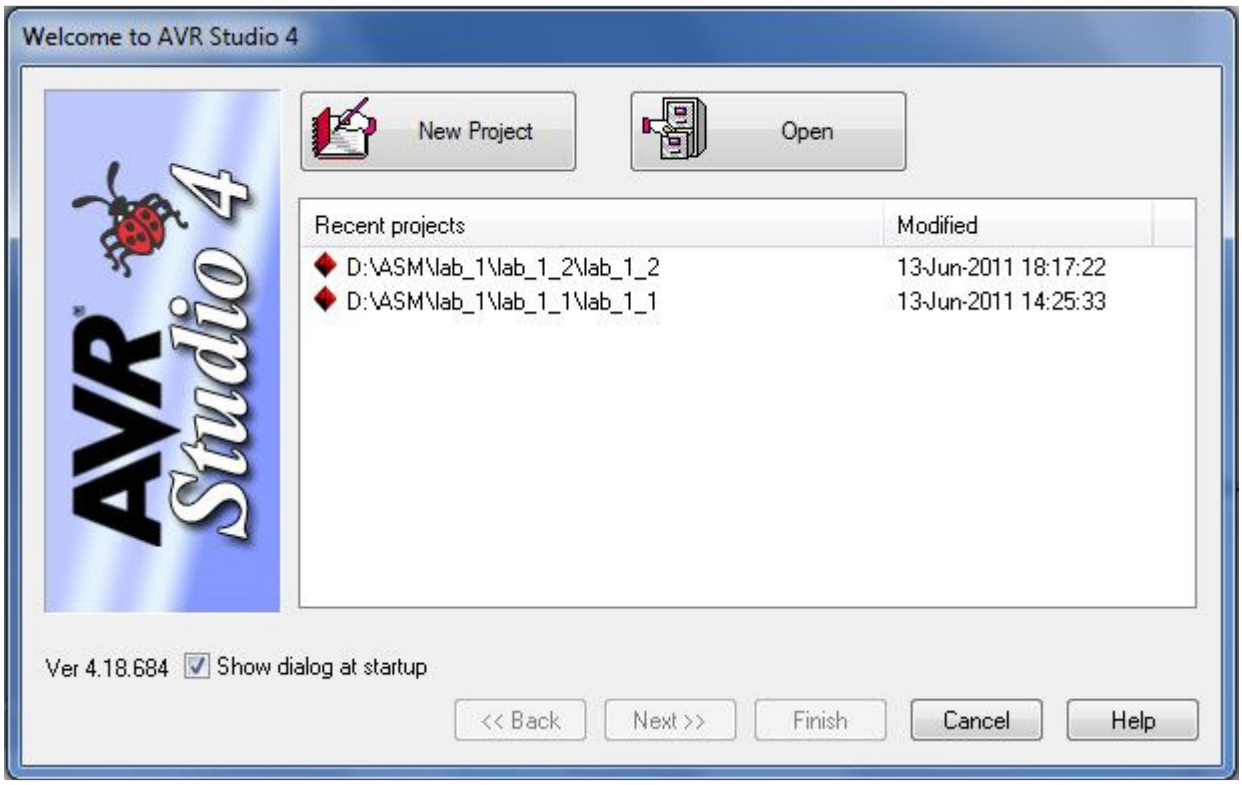


Рис. 1. Окно приветствия AVR Studio

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

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).

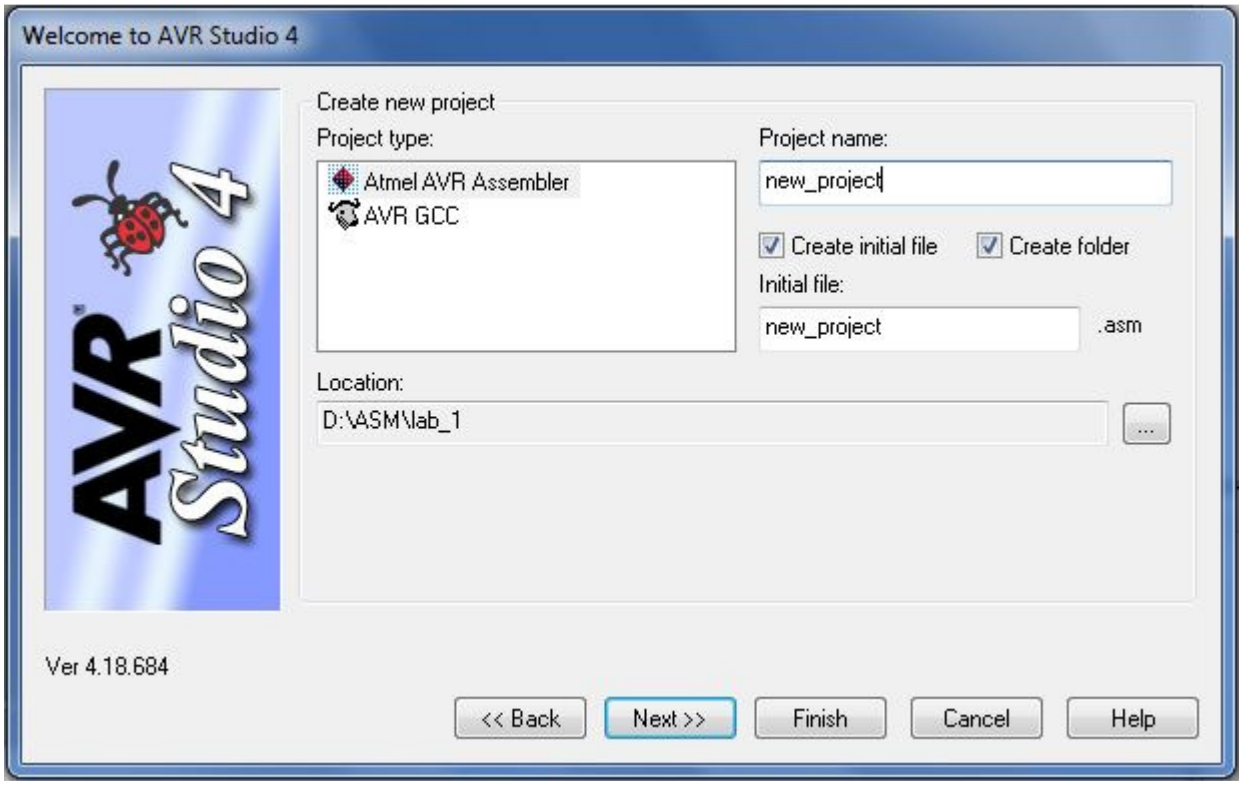


Рис. 2. Выбор языка программы

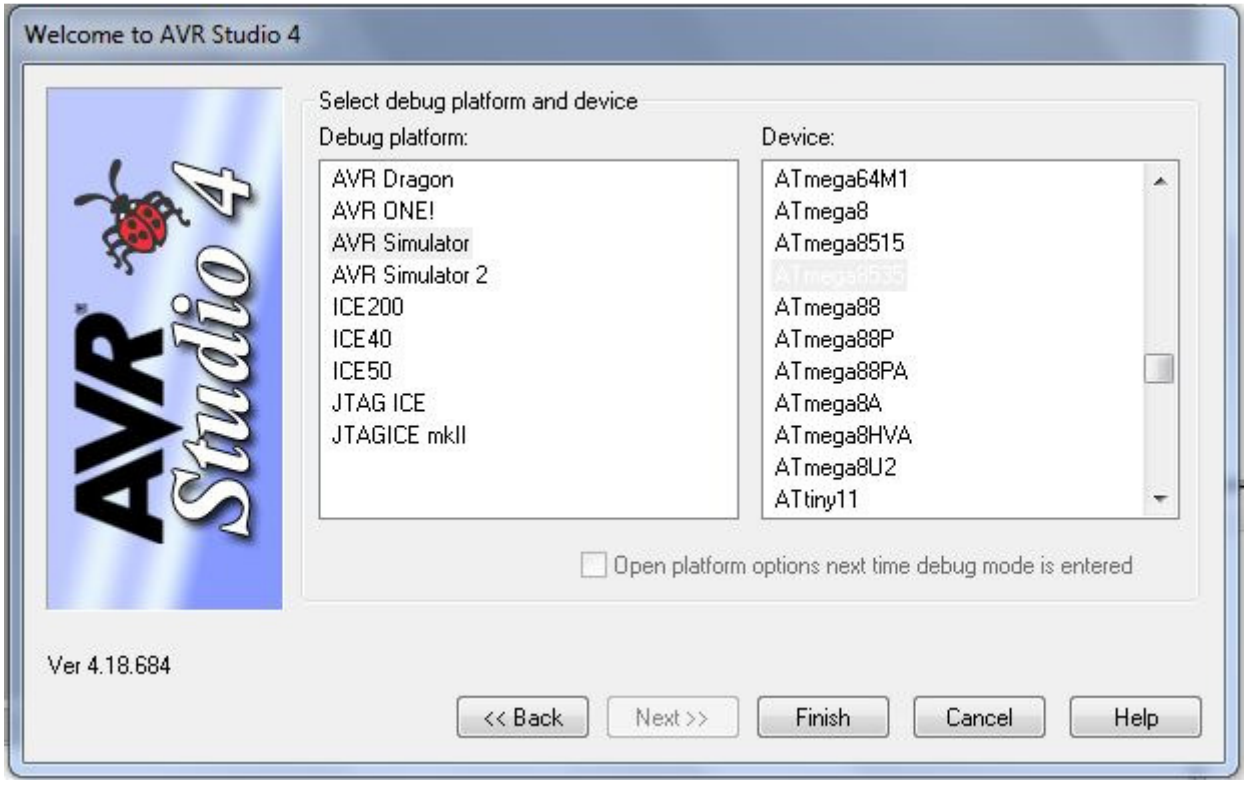


Рис. 3. Выбор платформы и типа устройства

4. При нажатии на кнопку «Finish» в программе открываются несколько областей (рис. 4). Это:

– 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 (рис. 4, окно «2»);

– окно текущих сообщений «built» программы (рис. 4, окно «3»);

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

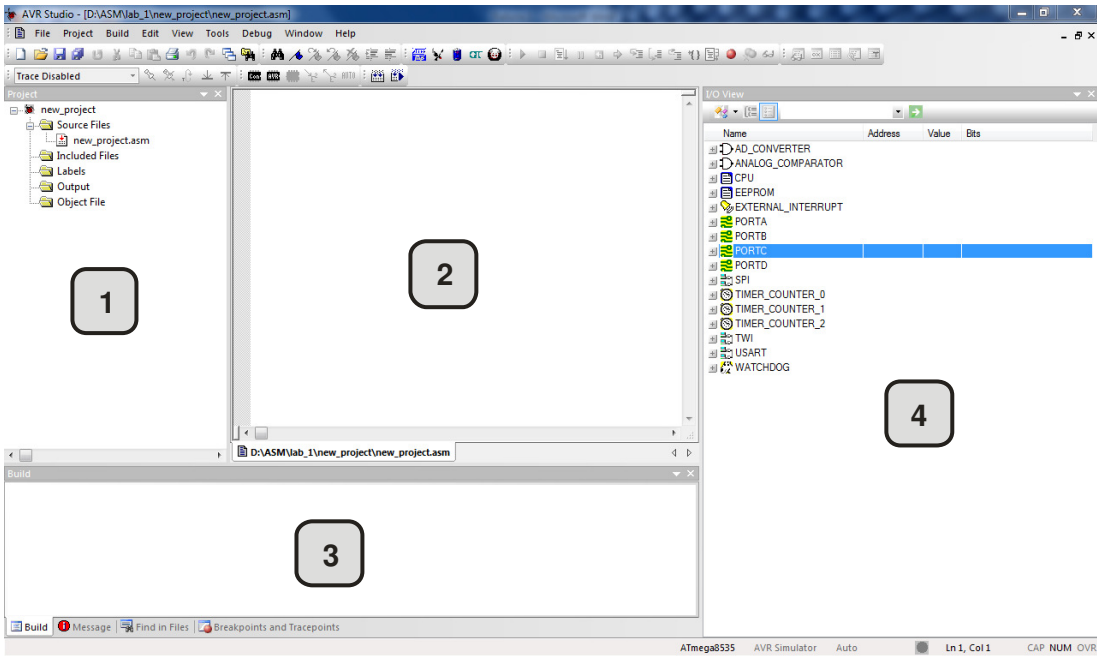


Рис. 4. Рабочее окно программы AVR Studio

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 стандартной библиотеки ATmega8535
.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 – логический «0», то
rjmp PINB0_is_0          ;переход на "PINB0_is_0"
ldi r16,0xAA             ;иначе вывод на PORTC
out PORTC,r16            ;числа 0xAA (1010 1010)
rjmp main                ;далее – возврат на main
PINB0_is_0:              ;если на PINB0 – логическая "1", то
ldi r16,0x55             ;вывод на PORTC
out PORTC,r16            ;числа 0x55 (0101 0101)
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.

Табл. 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.

Название	Сочетание клавиш	Назначение
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.
Toggle breakpoint (Точка остановки)	F9	Комадна добавляет пользовательскую точку остановки, при достижении которой программа будет приостановлена.
Remove all program breakpoints (удалить все точки остановки)	–	Команда удаляет все активные точки остановки программы.

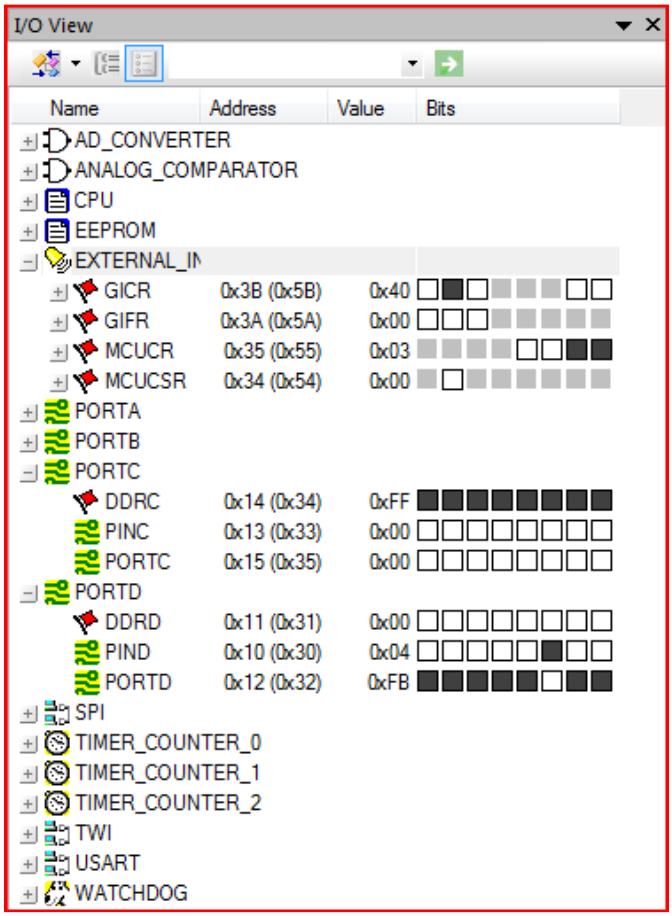


Рис. 5. Контроль состояния регистров микроконтроллера при симуляции

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 микроконтроллер

После успешной отладки программы необходимо проверить ее работу на микроконтроллере. Для этого необходимо:

– 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);

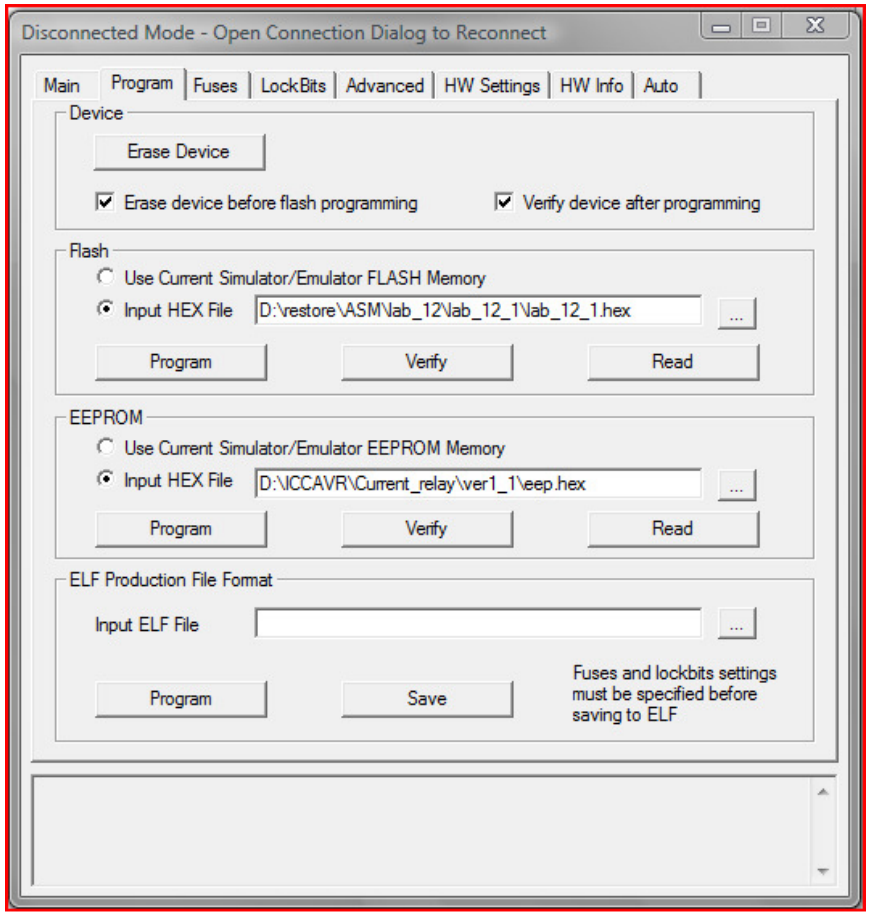


Рис. 6. Окно программирования микроконтроллера

– 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 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.

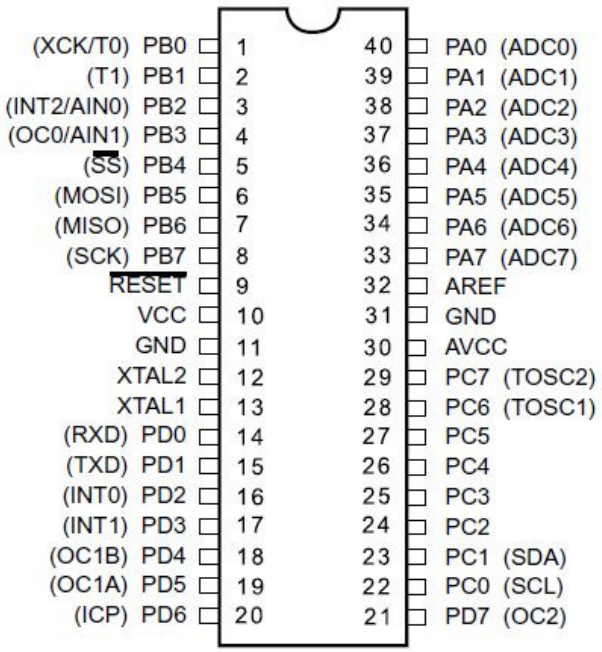


Рис. 1. Расположение выводов микросхемы контроллера ATmega8535 (DIP-40)

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).

Табл. 1. Регистры портов ввода/вывода

Наименование порта	**Регистры**		
	Регистр данных	**Регистр направления**	**Регистр состояния**
Порт A	PORTA	DDRA	PINA
Порт B	PORTB	DDRB	PINB
Порт C	PORTC	DDRC	PINC
Порт D	PORTD	DDRC	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 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».

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.

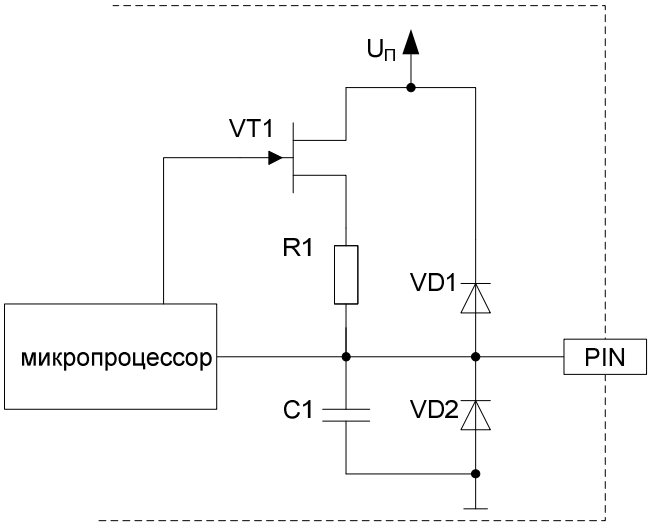


Рис. 2. Структура реализации вывода портов микроконтроллера

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).

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 А на ввод информации:

```
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 значение PINA записывается 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 информации:

```
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 0011 0101
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 чисел PC7:PC6 и PC1:PC0 с последующим
выводом результата на PORTD
;Входы: PINC7:PINC6 и PINC1:PINC0
;Выходы: PORTD
.include "m8535def.inc"  ;Подключение стандартной библиотеки ATmega8535
.cseg                    ;Начало сегмента кода
.org 0
reset:                   ;Инициализация портов ввода/вывода
     ldi r16,0xFF
     out DDRD,r16        ;Установка PORTD на вывод информации
     clr r16
     out DDRC,r16        ;Установка PORTC на ввод информации
main:
     in r16,PINC         ;Опрос PINC с занесением результата в r16
     mov r17,r16         ;Копирование результата в r17
     andi r16,0x03       ;Обнуление всех бит, кроме 0 и 1
     andi r17,0xC0       ;Обнуление всех вит, кроме 6 и 7
     clr r18             ;Очистка r18
shift:
     lsr r17             ;Сдвиг r17 влево
     inc r18             ;Инкремент r18
     cpi r18,0x06        ;Если r18≠6,
     brne shift          ;то возврат на shift
     add r16,r17         ;Иначе – сложение r16 и r17
     out PORTD,r16       ;Вывод результата на PORTD
     rjmp main           ;возврат на main
;-----------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. В начале программы подключается стандартная библиотека контроллера Atmega8535, которая содержит всю необходимую информацию о контроллере, а именно список регистров с их адресами, объемы памяти, список периферийных устройств и другие особенности:

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

2. Далее устанавливается начало сегмента кода на нулевой адрес:

```
.cseg
.org 0
```

3. После этого необходимо произвести инициализацию портов ввода/вывода. Порт C инициализируется на ввод, а порт D – на вывод информации:

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

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 (`ldi r16,0xFF`), а затем содержимое этого регистра переписывается в регистр DDRD (`out DDRD,r16`). Аналогичная операция производится с портом C.

4. В основном цикле программы необходимо опросить состояние регистра PINC и выделить два младших и два старших бита (согласно заданию):

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

Для этого содержимое регистра PINC переносится в регистр общего назначения (РОН) r16 (`in r16,PINC`) и копируется в регистр r17 (`mov r17,r16`). Далее осуществляется побитовое умножение содержимого регистров r16 и r17 с константами 0x03 (0000 0011) и 0xC0 (1100 0000) соответственно. Таким образом, в регистре r16 остаются только два младших бита PINC, а в регистре r17 – два старших бита PINC.

5. Для проведения операции сложения r16 и r17 необходимо преобразовать число XX00 0000, содержащееся в r17, в число вида 0000 00XX. Для этого в программе использован циклический сдвиг содержимого r17 на 6 бит:

```
clr r18
shift:
     lsr r17
     inc r18
     cpi r18,0x06
     brne shift
```

РОН r18 очищается (`clr r18`), происходит сдвиг r17 на один бит вправо (`lsr r17`), регистр r18 инкрементируется (`inc r18`), затем его содержимое сравнивается с числом 6 (`cpi r18,0x06`). Если r18 меньше 6, происходит возврат на метку «shift», иначе – выполнение программы далее (`brne shift`).

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 необходимо вернуться на метку «main» для создания циклического опроса портов:

```
rjmp main
```

Пример 2. Реализовать на микроконтроллере расчет логического уравнения $D = A \cdot (\overline{A} \cdot B + C)$, где A, B, C логические сигналы, поступающие на входы PINA0,

PINA1, PINA2, а D – результат решения логического уравнения, который выводится на PORTD0.

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

.include "m8535def.inc"  ;Подключение стандартной 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 бита числа A
     andi B,0x02         ;Выделение бита числа B
     andi C,0x04         ;Выделение 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 портов ввода/вывода в данной программе производится аналогично примеру 1:

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

```
reset:
     ldi r16,0x01
     out DDRD,r16
     clr r16
     out DDRA,r16
```

Здесь для удобства пользователя регистрам r20 … r23 присваиваются имена переменных A, B, C, D (`.def A=r20` и т.д.).

2. В основном цикле программы сначала в регистр r16 переписывается комбинация сигналов, находящаяся на выводах PINA2…PINA0. Затем производится выделение бита A, B и C:

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

3. В младших битах регистров r20…r22 формируются биты A, B, C. Для этого регистр r21 сдвигается на 1 бит, а r22 – на 2 бита вправо. В случае единичных сдвигов это целесообразно выполнить командой `lsl`:

```
lsr B
lsr C
lsr C
```

4. Для формирования числа $\overline{A}$ значение регистра A копируется в РОН r16, в котором оно инвертируется:

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

5. Далее производятся логические операции над числами A, B, C, NA согласно логическому уравнению $D = A \cdot (\overline{A} \cdot B + C)$, вывод результата на индикацию и зацикливание программы:

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

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 (далее SREG) 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. 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. 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      ;сравнение r16 и r17 проведением операции
                ;вычитания r16-r17. При равенстве выставляется
                ;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. 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 (флаг C) не происходит:

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

Результат:

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

– поскольку не произошло переполнение разрядов регистра, флаг переноса C не возникает.

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:

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

Результат:

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

– поскольку произошло переполнение разрядов регистра, возникает флаг переноса C.

Пример. 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. 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. 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 работе №4.

Пример. Произвести вычитание двух чисел. Первое число задается младшим полубайтом порта A и имеет формат A=0_0_0_0_PA3_PA2_PA1_PA0, второе число задается старшим полубайтом порта A и имеет формат B=0_0_0_0_PA7_PA6_PA5_PA4. Результат – модуль разницы A-B выводится на PORTC. Знак результата выводится на PORTC7.

```
//-------------------------------------------------------------------
.include "m8535def.inc" ;подключение стандартной библиотеки ATmega8535
.cseg                   ;начало сегмента кода
.org 0

ldi r16,0xFF            ;иициализация 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:                      ;метка 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. Сначала инициализируется тип используемого контроллера, а также указывается адрес начала сегмента кода:

```
.include "m8535def.inc"
.cseg
.org 0
```

2. Далее инициализируются порты ввода/вывода. Порт A инициализируется на ввод, порт C – на вывод информации:

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

3. В начале основного цикла производится опрос регистра PINA, 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
```

4. После приведения чисел A и B к одному виду осуществляется операция вычитания, которая устанавливает необходимые флаги в регистре SREG:

```
sub r16,r17
```

5. 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.

6. 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.

Работа № 4. Стек. Реализация программной задержки

Цель работы

Освоить теоретический и практический материал по использованию стека. 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.

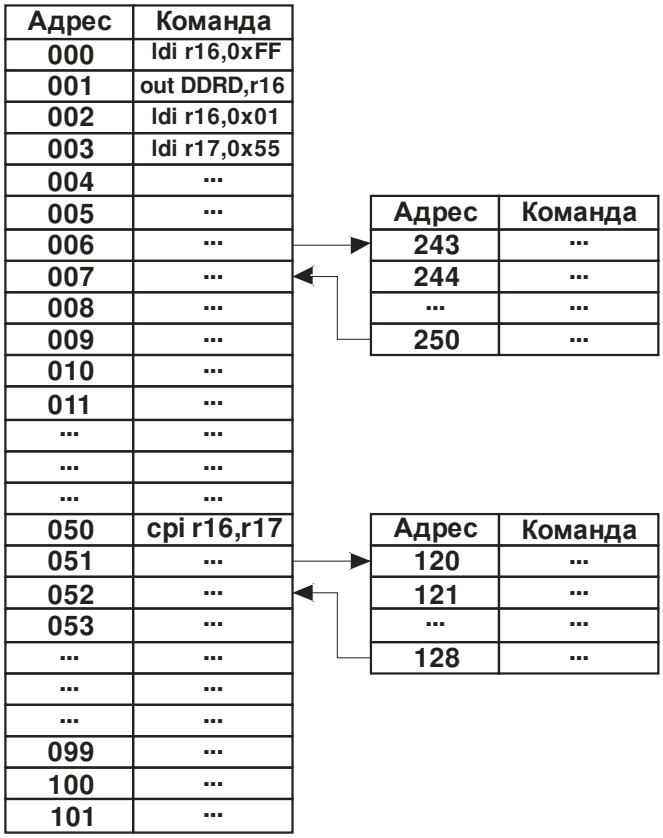


Рис. 1. Принцип работы указателя стека

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.

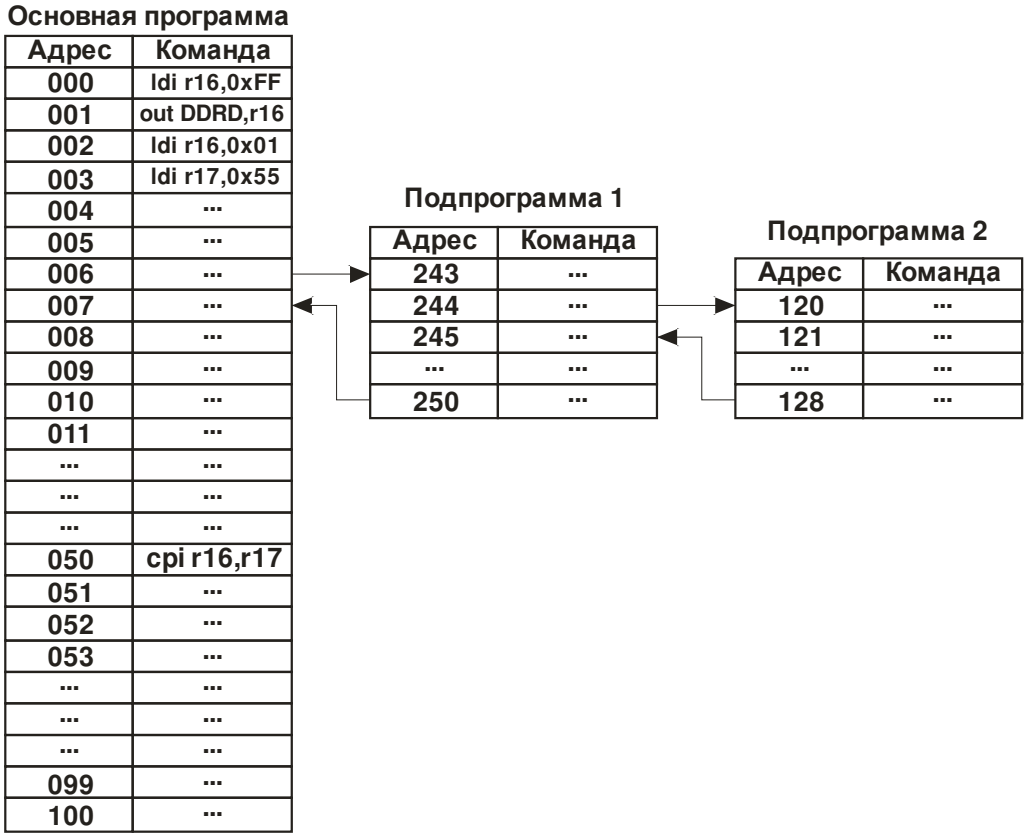


Рис. 2. Принцип заполнения стека

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 большим объемом памяти микроконтроллера.

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.

Рассчитаем примерное время программной задержки, реализованной следующим способом:

```
//-------------------------------------------------------------
clr r16
met_1:
inc r16
cpi r16,0xFF
brne met_1
//-------------------------------------------------------------
```

Здесь значение регистра РОН r16 увеличивается на 1, пока значение в r16 не достигнет значения 0xFF, или 255. При этом постоянно идет возврат на метку met_1, и только после того, как значение, записанное в r16, достигает значения 0xFF (или 255), разрешается выполнение последующих команд программы. Таким образом, программа как бы «зависает» на некоторое время. **Это «зависание» и является программной задержкой**.

Зная время выполнения команд (Приложение А) можно рассчитать время данной программной задержки. Примененные команды имеют следующее время расчета:

- `clr r16`: 1 такт процессора
- `inc r16`: 1 такт процессора
- `cpi r16,0xFF`: 1 такт процессора
- `brne met_1`: 1 такт процессора, если условие не выполняется и 2 такта процессора, если условие выполняется.

При попадании в цикл суммарное количество тактов процессора будет N=4, то есть при подсчете чисел от 1 до 255 суммарное количество тактов будет равняться N=4·255=1020 циклов. После этого необходимо вычесть один цикл, который остался неучтенным при выполнении ложного условия `brne met_1` в конце подсчета r16 и прибавить один такт на выполнение команды `clr r16`. Итого, суммарное количество тактов N=4·255–1+1=1020 циклов.

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

$$T_З = N \cdot \frac{1}{f_{OSC}} = 1020 \cdot \frac{1}{8 \cdot 10^6} = 127{,}5 \text{ мкс.}$$

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,low(RAMEND)      ;размещение вершины стека по адресу
ldi r17,high(RAMEND)     ;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 delay         ;вызов подпрограммы задержки
out SREG,r17        ;восстановление SREG после возврата из delay
rol r16             ;цикличный поворот r16 вправо
rjmp main           ;возврат на main
delay:                   ;подпрограмма №1
clr r18             ;очистка регистра r18
met1:
rcall delay1        ;переход на delay1
inc r18             ;инкремент r18
cpi r18,0x0F        ;сравнение значения r18 с числом 15
brne met1           ;если значение в r18≠15, то переход на met1
ret                 ;иначе – возврат в основную программу
delay1:             ;подпрограмма №2
clr r19             ;очистка регистра r19
met2:
rcall delay2        ;переход на delay2
inc r19             ;инкремент r19
cpi r19,0xFF        ;сравнение значения r19 с числом 255
brne met2           ;если значение в r19≠255, то переход на met2
ret                 ;иначе – возврат в подпрограмму №1
delay2:             ;подпрограмма №3
clr r20             ;очистка регистра r20
met3:
inc r20             ;инкремент r20
cpi r20,0xFF        ;сравнение значения r20 с числом 255
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 подключается библиотека контроллера Atmega8535, инициализируеся начало сегмента кода, а вершина стека помещается по адресу последней ячейки ОЗУ:

```
.include"m8535def.inc"
.cseg
.org 0
ldi r16,low(RAMEND)
```

```
ldi r17,high(RAMEND)
out spl,r16
out sph,r17
```
2 Инициализируется порт ввода/вывода C, 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 (`out PORTC,r16`), 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:
```
delay:
clr r18
met1:
rcall delay1
inc r18
cpi r18,0x0F
brne met1
ret
delay1:
clr r19
met2:
rcall delay2
inc r19
cpi r19,0xFF
brne met2
ret
delay2:
clr r20
met3:
inc r20
cpi r20,0xFF
brne met3
ret
```
Так, при входе в первый цикл, ограниченный меткой `met1` и условием `brne met1`, перед инкрементом регистра r18 происходит вызов подпрограммы `delay1`, в которой данная операция повторяется и осуществляется переход на подпрограмму

`delay2`. Таким образом, сначала происходит выполнение цикла подпрограммы `delay2`, потом – инкремент r19 в подпрограмме `delay1`, после чего снова осуществляется переход на `delay2`, 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 случает 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 дисассемблер программы, представленный в следующем виде:

Адрес	**Команда**	**SPH:SPL**
+00000000	ldi r16,low(RAMEND)	00 00
+00000001	ldi r17,high(RAMEND)	00 00
+00000002	out spl,r16	00 5F
+00000003	out sph,r17	02 5F
+00000004	ldi r16,0xFF	02 5F
+00000005	out DDRC,r16	02 5F
+00000006	ldi r16,0x01	02 5F
	main:	
+00000007	out PORTC,r16	02 5F
+00000008	in r17,SREG	02 5F
+00000009	rcall delay	02 5D
+0000000A	out SREG,r17	02 5F
+0000000B	rol r16	02 5F
+0000000C	rjmp main	02 5F
	delay:	
+0000000D	clr r18	02 5D
	met1:	
+0000000E	rcall delay1	02 5B
+0000000F	inc r18	02 5D
+00000010	cpi r18,0x0F	02 5D
+00000011	brne met1	02 5D
+00000012	ret	02 5F
	delay1:	
+00000013	clr r19	02 5B
	met2:	

Адрес	**Команда**	**SPH:SPL**
+00000014	rcall delay2	02 59
+00000015	inc r19	02 5B
+00000016	cpi r19,0xFF	02 5B
+00000017	brne met2	02 5B
+00000018	ret	02 5D
	delay2:	
+00000019	clr r20	02 59
+0000001A	inc r20	02 59
+0000001B	cpi r20,0xFF	02 59
+0000001C	brne met3	02 59
+0000001D	ret	02 5B

Таблица стека (Toggle memory window, Data Memory):

Адрес	**Значение**
02 58	0xFF
02 59	0xFF
02 5A	0x00
02 5B	0x15
02 5C	0x00
02 5D	0x0F
02 5E	0x00
02 5F	0x0A

Работа № 5. Таймеры T0 и T2 в режиме подсчета 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).

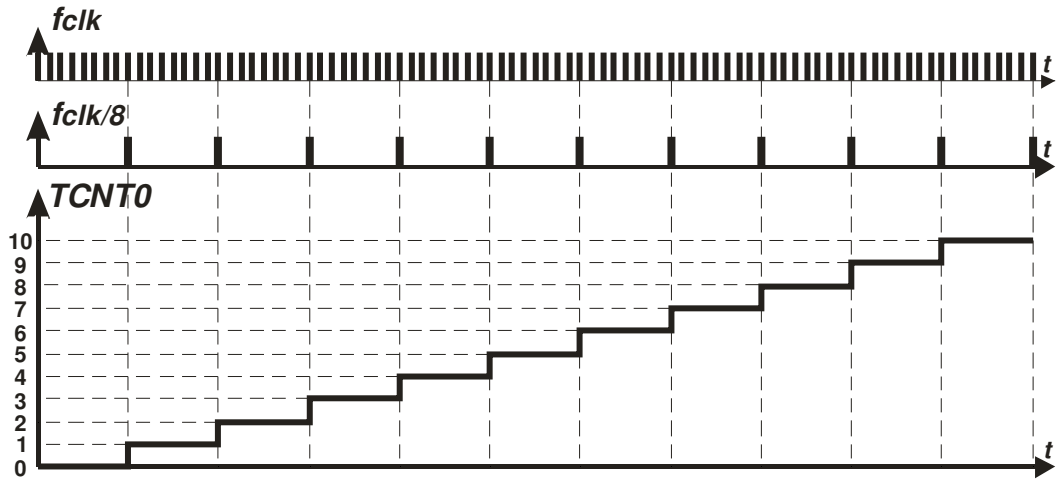


Рис. 1. Работа предделителя таймера T0 при коэффициенте делителя clk/8

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).

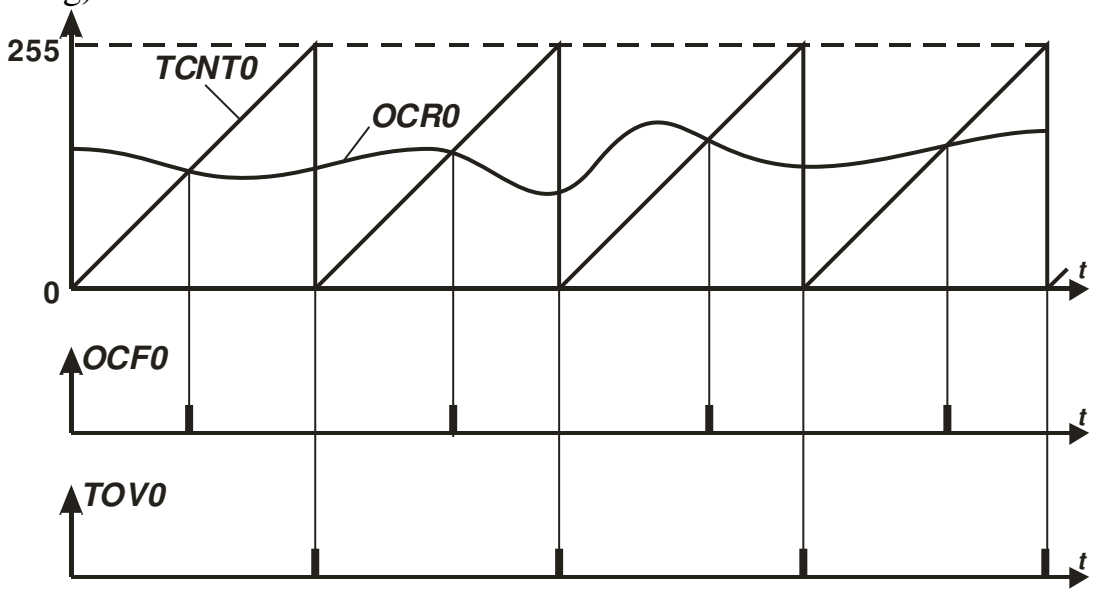


Рис. 2. Принцип работы таймера и формирования прерываний

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).

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. Этот бит определяет режим принудительной установки выходного сигнала таймера в режиме ШИМ (в данной работе не рассматривается).

Бит 6 и бит 3 – WGM00:WGM01. Эти биты определяют режим работы таймера/счетчика (табл. 2)

Табл. 2. Режимы работы таймера/Счетчика T0

WGM01	**WGM00**	**Режим**
0	0	Нормальный
0	1	Фазо-корректный ШИМ
1	0	Очистка при совпадении
1	1	Быстрый ШИМ

В нормальном режиме работы максимальное значение счетного регистра **TCNT0** ограничивается максимальным значением 255 (рис. 1, б).

В режиме очистки 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).

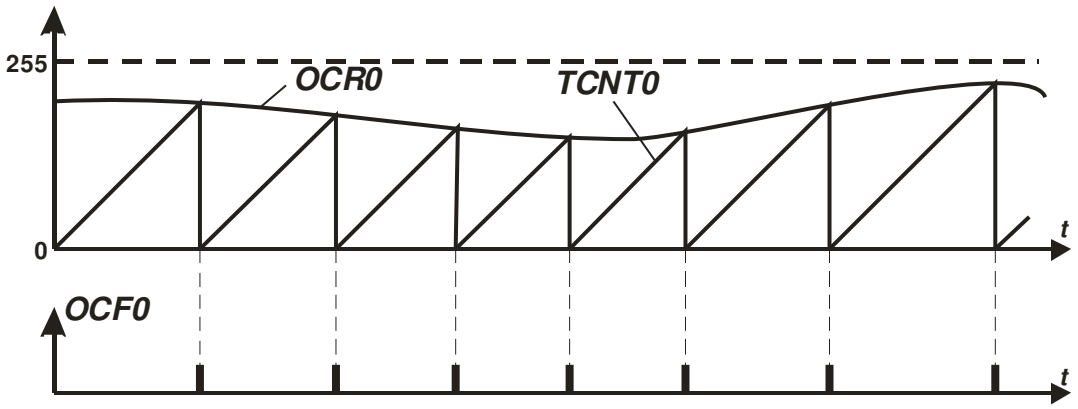


Рис. 3. Режим работы таймера T0 CTC (clear to compare) «Очистка при совпадении»

Биты 5 и 4 – COM01:COM00. Микросхема контроллера Atmega 8535 содержит вывод OC0 таймера T0 в качестве альтернативной функции вывода PB3 порта ввода/вывода B. Таймер T0 может управлять этим выводом в соответствии с

выбранным режимом работы, определяемым регистрами WGM01:WGM00, а также в соответствии с комбинацией бит COM01:COM00 (табл. 3)

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. Эти биты определяют источник задания частоты таймера T0 и предделитель 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 $f_{CLK}/1$
0	1	0	Предделитель $f_{CLK}/8$
0	1	1	Предделитель $f_{CLK}/64$
1	0	0	Предделитель $f_{CLK}/256$
1	0	1	Предделитель $f_{CLK}/1024$
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 регистры флагов и масок прерываний таймеров соответственно **TIFR** и **TIMSK**.

Значение этих регистров следующее. Когда возникает совпадение или переполнение таймера T0, автоматически формируется флаг прерывания OCF0 или TOV0 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.

Поскольку регистры **TIMSK** и **TIFR** – общие для всех таймеров, они содержат все флаги и маски таймеров микроконтроллера.

Табл. 5. Описание регистра TIFR

Бит	7	6	5	4	3	2	1	0
Название	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Табл. 6. Описание регистра TIMSK

Бит	7	6	5	4	3	2	1	0
Название	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

В регистре **TIFR** (табл. 5) бит 1 OCF0 – флаг прерывания по совпадению таймера T0; бит 0 TOV0 – флаг прерывания по переполнению таймера T0.

В регистре **TIMSK** (табл. 6) бит 1 OCIE0 – маска прерывания по совпадению таймера T0; бит 0 TOIE0 – маска прерывания по переполнению таймера T0.

8-разрядный таймер/счетчик T2

Принцип и режимы работы таймера T2 практически не отличаются от рассмотренного ранее таймера T0, однако 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. Этот бит определяет режим принудительной установки выходного сигнала таймера в режиме ШИМ (в данной работе не рассматривается).

Бит 6 и бит 3 – WGM20:WGM21. Эти биты определяют режим работы таймера/счетчика (табл. 8).

Табл. 8. Режимы работы таймера/Счетчика T0

WGM21	**WGM20**	**Режим**
0	0	Нормальный
0	1	Фазо-корректный ШИМ
1	0	Очистка при совпадении
1	1	Быстрый ШИМ

Биты 5 и 4 – COM21:COM20. Микросхема контроллера Atmega 8535 содержит вывод OC2 таймера T2 в качестве альтернативной функции вывода PD7 порта ввода/вывода B. Таймер T2 может управлять этим выводом в соответствии с выбранным режимом работы, определяемым регистрами WGM21:WGM20, а также в соответствии с комбинацией бит COM21:COM20 (табл. 9)

Табл. 9. Функции вывода OC2 таймера T2 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 T2 и предделитель таймера (табл. 10).

Табл. 10. Функции бит CS22, CS21, CS20 таймера T2

CS02	**CS01**	**CS00**	**Пояснение**
0	0	0	Нет источника. Таймер остановлен.
0	0	1	Предделитель $f_{CLK}/1$
0	1	0	Предделитель $f_{CLK}/8$
0	1	1	Предделитель $f_{CLK}/32$
1	0	0	Предделитель $f_{CLK}/64$
1	0	1	Предделитель $f_{CLK}/128$
1	1	0	Предделитель $f_{CLK}/256$
1	1	1	Предделитель $f_{CLK}/1024$

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

Табл. 11. Регистр асинхронного режима таймера T2

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 на PORTC.

```
;---------------------------------------------------------------------
;Программа вывода на 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 Atmega8535
.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 прерывания

reset:
cli                      ;глобальный запрет прерываний
ldi r16,low(RAMEND)      ;запись в указатель стека адреса конца
ldi r17,high(RAMEND)     ;памяти данных
out spl,r16
out sph,r17
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
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
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 выполнена
rjmp main                ;пустой.

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 программу более подробно.

1. Сначала подключается стандартная библиотека контроллера Atmega8535:

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

2. После этого производится инициализация адресов памяти, по которым начинается основная программа и подпрограмма обработки прерывания по совпадению таймера T0. Эти адреса для данного контроллера заданы жестко и изменению не подлежат. Таблица адресов приведена в приложении.

```
.cseg
.org$0
rjmp reset
.org$13
rjmp T0_compare
```

3. В основном цикле производится инициализация портов ввода/вывода и таймера T0. Также указывается начало указателя стека SPH:SPL и обнуляются необходимые регистры общего назначения:

```
cli
ldi r16,low(RAMEND)
ldi r17,high(RAMEND)
out spl,r16
out sph,r17
ldi r16,0xFF
out PORTD,r16
out DDRC,r16
clr r16
out DDRD,r16
out PORTC,r16
ldi r16,0x00
out TCCR0,r16
out TCNT0,r16
ldi r16,0x4E
out OCR0,r16
ldi r16,0x05
out TCCR0,r16
ldi r16,0x02
out TIMSK,r16
clr r16
clr r17
clr r18
main:
rjmp main
```

Для правильной инициализации таймера T0 сначала все его регистры обнуляются. После этого необходимо выбрать предделитель таймера и задать уставку таймера.
В случае данной программы получить уставку 500 мс на таймере не получается, так как даже при предделителе clk/1024 максимальная уставка таймера равняется:

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

Поэтому для получения уставки $T_{\text{УСТ}} = 500$ мс принято решение запрограммировать таймер на уставку 10 мс и производить подсчет срабатываний таймера. Каждое 50-е срабатывание будет соответствовать времени 500 мс.
Уставка таймера T0 $T_{\text{УСТ}} = 10$ мс при предделителе clk/1024 получается при уставке в регистре сравнения:

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

Таким образом, в регистр сравнения OCR0 записывается число 78 (0x4E). После этого таймер запускается в нормальном режиме с предделителем clk/1024. В регистре масок таймеров TIMSK ставится маска на прерывание по совпадению таймера T0 OCIE0.
После процесса инициализации разрешаются глобальные прерывания (`sei`) путем записи логической «1» в старший бит регистра SREG и основная программа замыкается.
4. При наступлении прерывания по совпадению T0 осуществляется переход из любого места основной программы по адресу обработки прерывания OCIE1. После этого запрещаются все прерывания (`cli`), которые разрешаются только по окончании обработки прерывания. При входе в программу обработки прерывания также сохраняется регистр SREG (`in r24,SREG`), который при выходе из подпрограммы (`reti`) восстанавливается (`out SREG,r24`).
Следует отметить, что так можно поступать, только если в программе задействовано одно прерывание. Иначе при наступлении прерывания необходимо сначала перейти на свободный от таблицы прерываний адрес памяти, а затем производить обработку прерывания.
В подпрограмме обработки прерывания после необходимых процедур, описанных выше, происходит обнуление счетного регистра таймера T0:

```
clr r16
out TCNT0,r16
```

После этого организуется счет количества срабатываний прерывания таймера:

```
inc r17
cpi r17,0x32
brne quit
```

Регистр r17 постоянно инкрементируется и при его совпадении с уставкой 0x32 (50) (`cpi r17,0x32`) происходит дальнейшая обработка данных. Иначе осуществляется переход на метку, по которой производится выход из подпрограммы обработки прерывания (`brne quit`).

```
clr r17
in r16,PIND
cp r16,r18
brpl met_1
clr r18
```

```
met_1:
out PORTC,r18
inc r18
```
Каждые 500 мс значение счетчика прерываний обнуляется (`clr r17`), после чего производится считывание данных с порта D (`in r16,PIND`) и их 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 РОН r18 инкрементируется.

Работа № 6. Аналого-цифровой преобразователь микроконтроллера

Цель работы

Освоить теоретический и практический материал по работе 10-разрядного аналого-цифрового преобразователя (АЦП) микроконтроллера Atmega 8535. Применить приобретенные навыки при написании программы.

Программа работы

1. Изучить необходимый теоретический материал о принципах работы и функционирования встроенного АЦП микроконтроллера Atmega8535 и освоить его инициализацию.
2. Разобраться в программе по использованию АЦП, представленной в лабораторной работе.
3. Написать и отладить собственную программу с использованием АЦП в соответствии с вариантом.

Пояснения к работе

При использовании микроконтроллера в качестве устройства управления каким-либо процессом часто возникает необходимость оценивать величины аналоговых сигналов, в качестве которых могут выступать сигналы с задающих потенциометров, датчиков обратных связей, термопар и др.

Однако, поскольку микроконтроллер является цифровым устройством, непосредственно оценить величину аналогового сигнала путем опроса ножки микроконтроллера, к которой он подключен, не представляется возможным.

Для преобразования аналогового сигнала в цифровой с целью его последующей обработки предназначен **аналого-цифровой преобразователь** (АЦП), встроенный в микроконтроллер Atmega8535 и другие контроллеры семейства AVR фирмы Atmel.

АЦП микроконтроллера выполнен 10-разрядным, то есть аналоговый сигнал, поступающий на его вход, может быть разложен на 2^{10}=1024 дискреты. Таким образом, фактически, разрядность АЦП отвечает за его разрешающую способность, или чувствительность преобразователя. **Разрешающая способность АЦП** – это минимальная разница аналогового сигнала при двух измерениях, которую еще различает преобразователь.

Для правильного функционирования АЦП ему необходим некий «эталон», то есть напряжение, которое будет приниматься за базу, относительно которой будет измеряться подаваемый на АЦП аналоговый сигнал. Это эталонное напряжение принято называть **опорным напряжением**. В качестве источника опорного напряжения в микроконтроллерах Atmega8535 может выступать напряжение питания контроллера, внутренний стабилизированный источник 2,56 В или внешний сигнал, подключаемый к выводу AREF микросхемы контроллера.

Минимальное напряжение, которое АЦП сможет распознать, можно рассчитать по формуле:

$$U_{min} = \frac{1}{2^n} \cdot U_{ref}.$$

При использовании в качестве источника опорного напряжения питания микроконтроллера 5 В:

$$U_{min} = \frac{1}{2^{10}} \cdot 5 = 4{,}88 \text{ мВ}.$$

Ввиду того, что в качестве АЦП в микроконтроллерах AVR используется **АЦП последовательного приближения**, процесс преобразования аналогового сигнала в пропорциональный ему цифровой код занимает некоторое время. Упрощенная структура АЦП последовательного приближения представлена на рис. 1.

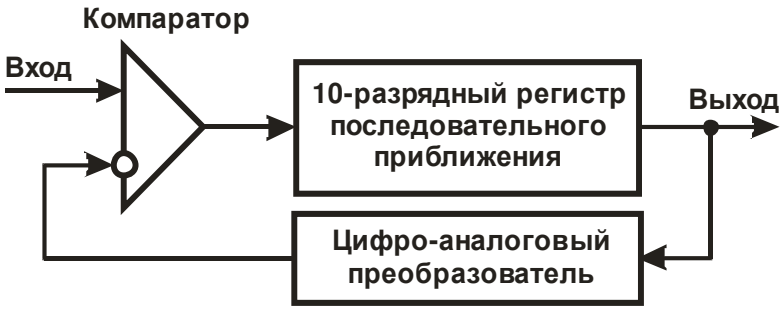


Рис. 1. Структура АЦП последовательного приближения

АЦП состоит из компаратора, регистра последовательного приближения и цифро-аналогового преобразователя. Работает АЦП следующим образом.

В начале преобразования все выходы регистра последовательного приближения устанавливаются в состояние логического «0», за исключением старшего разряда: 10 0000 0000. При этом на выходе внутреннего цифро-аналогового преобразователя формируется аналоговый сигнал, равный половине диапазона АЦП (рис. 2, интервал t0..t1). Компаратор при этом измеряет разницу между входным сигналом и сигналом с выхода ЦАП.

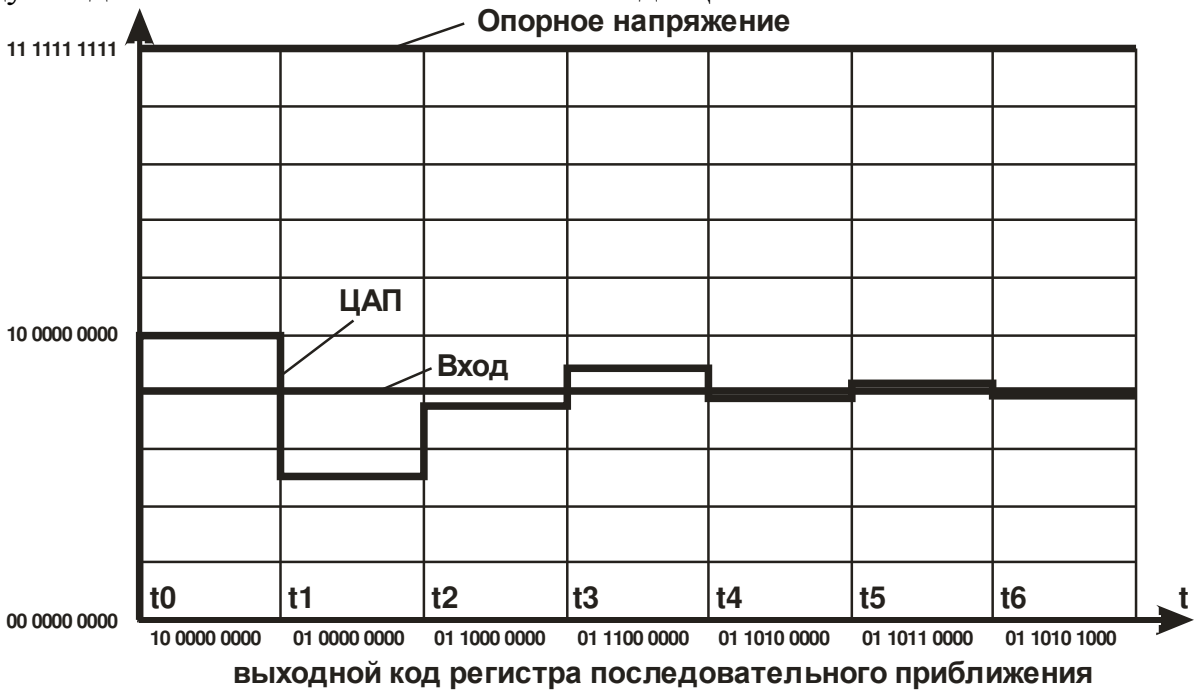


Рис. 2. Принцип работы АЦП последовательного приближения

Если напряжение на входе АЦП оказывается больше, чем установленное на выходе ЦАП, то регистр последовательного приближения сохраняет принятое изначально состояние 10 0000 0000. Если же измеряемое напряжение оказывается меньше подаваемого с ЦАП, регистр устанавливается в состояние 01 000 0000 (рис. 2, интервал t1-t2).

Если принятое состояние 01 0000 0000 оказывается меньше измеряемого сигнала, 8-й бит кода закрепляется и 7-й бит кода регистра последовательного приближения устанавливается в единичное состояние: 01 1000 0000 (рис. 2, интервал t2-t3). Поскольку и при прибавлении этого бита сигнал на выходе ЦАП оказался меньше входного (рис. 2), 7-й бит закрепляется, а к коду прибавляется 6-й бит: 01 1100 0000 (рис. 2, интервал t3-t4).

В этом случае выходной сигнал ЦАП оказывается больше сигнала на входе, поэтому 6-й бит принимается равным «0», а к коду прибавляется 5-й бит: 01 1010 0000 (рис. 2, интервал t4-t5). Далее процесс повторяется до установки последнего младшего бита кода регистра последовательного приближения.

По окончании процесса преобразования результат передается в два 8-разрядных регистра **ADCH** и **ADCL**.

В микроконтроллерах AVR время преобразования АЦП можно регулировать. Для этого в структуру преобразователя встроен предделитель, подобный тому, который встроен в таймеры T0…T2 микроконтроллера. Необходимо отметить, что с уменьшением времени преобразования уменьшается точность получаемого результата. Рекомендуемая частота работы АЦП находится в пределах 50…200 кГц. При работе преобразователя в этом диапазоне обеспечивается минимальная погрешность при получении результата.

Для того, чтобы правильно рассчитать время преобразования АЦП, необходимо помнить, что процесс преобразования занимает 13 тактов АЦП при его непрерывной работе и 25 циклов при его первом запуске. Аналого-цифровой преобразователь микроконтроллера Atmega8535 содержит 8 входов, подключенных к выводам порта ввода/вывода A. Поскольку преобразователь одновременно может работать только с одним входом, они коммутируются специальным коммутатором (мультиплексором), управляемым программно с помощью регистра **ADMUX** (табл. 1).

Табл. 1 Регистр управления мультиплексором АЦП **ADMUX**

Бит	7	6	5	4	3	2	1	0
Название	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Биты 6 и 7 – REFS1, REFS0. С помощью этих бит определяется источник опорного напряжения АЦП (табл. 2).

Табл. 2.Задание источника опорного напряжения АЦП

REFS1	**REFS0**	**Источник опорного напряжения**
0	0	Внешний источник, подключенный к выводу AREF.
0	1	Напряжение питания АЦП, подаваемое на вывод AVCC. К выводу AREF должен быть подключен конденсатор.
1	0	Комбинация не используется.
1	1	Внутренний стабилизированный источник 2,56 В. К выводу AREF должен быть подключен конденсатор.

Бит 5 – ADLAR. Этот бит отвечает за «левое» выравнивание результата. Поскольку для хранения 10-разрядного результата преобразования АЦП используются два 8-разрядных регистра ADCH и ADCL, некоторые биты регистра ADCH остаются неиспользуемыми. Если ADLAR=1, то результат преобразования АЦП сохраняется «левым» выравниванием по регистру ADCH. Это особенно удобно, если пользователю не нужны два младших бита результата преобразования АЦП (табл. 3).

Табл. 3. Представление результата преобразования АЦП в зависимости от управляющего бита ADLAR

ADLAR=0								
ADCH	–	–	–	–	–	–	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
ADLAR=1								
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	–	–	–	–	–	–

Биты 4…0 – MUX4…MUX0. С помощью этих бит определяется, какой из входов АЦП подключен к преобразователю. Здесь различают два режима работы входов АЦП – одиночный и дифференциальный (табл. 4).

Табл. 4. Определение логики работы входов АЦП в соответствии с битами MUX4…MUX0

MUX4…MUX0	**Одиночные входы**	**Положительный дифференциальный вход**	**Отрицательный дифференциальный вход**	**Коэффициент усиления**
00000	ADC0	–		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	–	ADC0	ADC0	10
01001		ADC1	ADC01	10
01010		ADC0	ADC0	200
01011		ADC1	ADC0	200
01100		ADC2	ADC2	10
01101		ADC3	ADC2	10
01110		ADC2	ADC2	200
01111		ADC3	ADC2	200
10000		ADC0	ADC1	1
10001		ADC1	ADC1	1
10010		ADC2	ADC1	1
10011		ADC3	ADC1	1
10100		ADC4	ADC1	1
10101		ADC5	ADC1	1

MUX4…MUX0	Одиночные входы	Положительный дифференциальный вход	Отрицательный дифференциальный вход	Коэффициент усиления
10110		ADC6	ADC1	1
10111		ADC7	ADC1	1
11000		ADC0	ADC2	1
11001		ADC1	ADC2	1
11010		ADC2	ADC2	1
11011	–	ADC3	ADC2	1
11100		ADC4	ADC2	1
11101		ADC5	ADC2	1
11110	1,22 В	–		
11111	0 В			

При решении типовых задач обычно используется одиночное включение входов (табл. 4), однако в некоторых случаях (например, для исключения воздействия на входы микроконтроллера сигнала помехи) возникает необходимость использования дифференциальных входов. В этом случае АЦП измеряет сигнал не между конкретным аналоговым входом ADC0…ADC7 и общей точкой, а между положительным и отрицательным дифференциальными входами (табл. 4).

В некоторых случаях, когда необходимо выловить разницу в очень близких сигналах, полезно перед аналого-цифровым преобразованием эти сигналы усилить. С этой целью перед подачей на АЦП микроконтроллер может усилить дифференциальный сигнал в 1…10…200 раз (табл. 4).

Для калибровки преобразователя можно также ко всем входам подключить источник стабильного напряжения 1,22 В (MUX4…0=11110) или общую точку (MUX4…0=11111).

Настройка аналого-цифрового преобразователя и управление им осуществляется с помощью регистра управления АЦП **ADCSRA** (табл. 5).

Табл. 5. Регистр управления АЦП ADCSRA

Бит	7	6	5	4	3	2	1	0
Название	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Бит 7 – ADEN. Этот бит разрешает использование АЦП. Записью логического «0» в ADEN АЦП будет немедленно выключено.

Бит 6 – ADSC. Запись логической «1» в этот бит разрешает преобразование АЦП. Необходимо записывать в ADCS логическую «1» для начала каждого преобразования. По окончании преобразования бит будет автоматически сброшен в состояние логического «0».

Бит 5 – ADATE. Записью логической «1» в этот бит разрешается автоматический запуск АЦП по событию. Событие выбирается комбинацией битов ADTS в регистре **SFIOR** микроконтроллера.

Бит 4 – ADIF. Этот бит – флаг готовности результата преобразования АЦП.Устанавливается автоматически по окончании процесса преобразования.

Бит 3 – ADIE. Этот бит – маска прерывания готовности результата преобразования АЦП. Если ADIE=1, то при появлении флага готовности результата ADIF будет сгенерировано прерывание.

Биты 2…0 – ADPS2…ADPS0. Комбинация этих бит устанавливает делитель тактовой частоты процессора для тактирования АЦП (табл. 6).

Табл. 6. Предделитель АЦП микроконтроллера Atmega8535

ADPS2	**ADPS1**	**ADPS0**	**Предделитель**
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

В регистре специальных функций битами ADTS2…ADTS0 задается источник автозапуска АЦП при активации этой функции в регистре **ADCSRA**.

Табл. 7. Регистр специальных функций контроллера SFIOR

Бит	7	6	5	4	3	2	1	0
Название	ADTS2	ADTS1	ADTS0	–	–	–	–	–

Табл. 8. Источник автозапуска АЦП (биты ADTS2…ADTS0)

ADTS2	**ADTS1**	**ADTS0**	**Источник автозапуска**
0	0	0	Непрерывное преобразование АЦП
0	0	1	Аналоговый компаратор
0	1	0	Внешнее прерывание INT0
0	1	1	Прерывание по совпадению таймера T0
1	0	0	Прерывание по переполнению таймера T0
1	0	1	Прерывание по совпадению канала B таймера T1
1	1	0	Прерывание по переполнению таймера T1
1	1	1	Прерывание по захвату сигнала таймера T1

Пример 1. Подать аналоговый сигнал на вход ADC2 АЦП и вывести 8-разрядный результат на порт ввода/вывода C. Не использовать прерывание готовности результата преобразования АЦП.

```
;-----------------------------------------------------------------
Входы:
     PA2 – аналоговый вход АЦП
Выходы:
     PC0…PC7 – индикация кода результата преобразования.

.include"m8535def.inc"   ;Подключение стандартной библиотеки Atmega8535
.cseg                    ;Начало сегмента кода
.org$0                   ;по адресу 0

ldi r16,low(RAMEND)      ;Запись адреса вершины стека
ldi r17,high(RAMEND)     ;В конце памяти данных
```

```
out spl,r16
out sph,r17
ldi r16,0xFB            ;Инициализация неиспользуемых ножек порта A
out PORTA,r16           ;на ввод информации.
clr r16             ;Используемый вход АЦП не инициализируется
out DDRA,r16
out PORTC,r16       ;Инициализация порта C на вывод информации
ser r16
out DDRC,r16
ldi r16,0x62        ;Инициализация мультиплексора АЦП
out ADMUX,r16
ldi r16,0xC7
out ADCSRA,r16      ;Инициализация АЦП и его запуск

main:               ;Начало основного цикла
in r16,ADCSRA       ;Опрос регистра ADCSRA
SBRC r16,4          ;Если бит 4 чист, то следующая команда игнориуется
rcall ADC_ready     ;Иначе вызов ADC_ready
rjmp main           ;Возврат на main

ADC_ready:          ;По метке ADC_ready
in r16,SREG         ;сохранение значения регистра SREG
ldi r17,0x97        ;Обнуление флага ADIF записью в него логической «1»
out ADCSRA,r17
in r17,ADCH         ;Считывание результата преобразования из ADCH
out PORTC,r17       ;и вывод результата на PORTC
ldi r17,0xC7        ;Перезапуск АЦП
out ADCSRA,r17
out SREG,r16        ;Восстановление регистра SREG
ret                 ;возврат по адресу, хранящемуся в стеке
;---------------------------------------------------------------------
```

Рассмотрим программу более подробно.

1. Сначала происходит подключение стандартной библиотеки контроллера Atmega8535. Также указывание адрес начала сегмента кода:

```
.include"m8535def.inc"
.cseg
.org$0
```

2. Далее осуществляется инициализация стека, вершина стека располагается в конце памяти данных.

```
ldi r16,low(RAMEND)
ldi r17,high(RAMEND)
out spl,r16
out sph,r17
```

3. Производится инициализация портов ввода/вывода. В программе используются порты ввода/вывода A и C. При этом необходимо запомнить, что, поскольку АЦП использует альтернативные функции порта A, то выводы порта, используемые АЦП, инициализировать не нужно. Остальные выводы рекомендуется «притянуть» к уровню логического «0» или «1» во-избежание наведения на них помех. Порт C инициализируется на вывод информации:

```
ldi r16,0xFB
out PORTA,r16
clr r16
```

```
out DDRA,r16
out PORTC,r16
ser r16
out DDRC,r16
```

4. После инициализации портов производится инициализация АЦП установкой необходимых управляющих бит в регистрах ADMUX и ADCSRA:

```
ldi r16,0x62
out ADMUX,r16
ldi r16,0xC7
out ADCSRA,r16
```

В качестве источника сигнала опорного напряжения выбирается напряжение электропитания процессора, поданное на вывод AVCC. Также выбирается «левое» выравнивание результата преобразования ADLAR. Выбирается второй канал АЦП (`ldi r16,0x62;out ADMUX,r16`). В регистра ADCSRA устанавливаются биты ADEN (включение АЦП), ADSC (запуск преобразования АЦП), а также выбирается предделитель clk/128 для повышения точности преобразования.

5. В основном цикле ведется опрос флага окончания преобразования АЦП, расположенного в регистре ADCSRA:

```
main:
in r16,ADCSRA
SBRC r16,4
rcall ADC_ready
rjmp main
```

Сначала считывание значение регистра ADCSRA в РОН r16 (`in r16,ADCSRA`). После этого опрашивается 4-й бит этого регистра, соответствующий флагу ADIF (`SBRC r16,4`). Если бит чист, то пропускается следующая за директивой `SBRC` инструкция. Иначе происходит переход на метку ADC_ready (`rcall ADC_ready`).

6. По метке ADC_ready происходит обработка результата преобразования АЦП:

```
ADC_ready:
in r16,SREG
ldi r17,0x97
out ADCSRA,r17
in r17,ADCH
out PORTC,r17
ldi r17,0xC7
out ADCSRA,r17
out SREG,r16
ret
```

При переходе по метке ADC_ready сначала в РОН r16 необходимо сохранить значение регистра SREG (`in r16,SREG`). После этого необходимо вручную сбросить флаг готовности преобразования АЦП записью в него логической «1» (`ldi r17,0x97; out ADCSRA,r17`). Необходимо запомнить, что все флаги в микроконтроллерах AVR сбрасываются записью в них логической «1». Далее в РОН r17 считывание результат преобразования АЦП, хранящийся в ADCH (`in r17,ADCH`). Два младших бита, хранящиеся в ADCL, отбрасываются. Содержимое r17 передается в порт ввода/вывода C (`out PORTC,r17`), осуществляется перезапуск АЦП, восстановление регистра SREG (`out SREG,r16`) и выход из подпрограммы на адрес, записанный в стеке (`ret`).

Пример 2. Задача повторяет пример 1, но используется прерывание по готовности результата преобразования АЦП. Не использовать «левое» выравнивание результата.

```
;---------------------------------------------------------------------
Входы:
     PA2 – аналоговый вход АЦП
Выходы:
     PC0…PC7 – индикация кода результата преобразования.
.include"m8535def.inc"   ;Подключение стандартной библиотеки Atmega8535
.cseg                    ;Начало сегмента кода.
.org$0                   ;По адресу $0
rjmp reset               ;осуществляется переход на reset.
.org$0E                  ;При переходе по адресу $0E
rjmp ADC_ready           ;осуществляется вызов функции обработки
                         ;прерывания готовности АЦП.
reset:                   ;По метке reset:
cli                      ;осуществляется запрет всех прерываний.
ldi r16,low(RAMEND)      ;Инициализируется стек
ldi r17,high(RAMEND)
out spl,r16
out sph,r17
ldi r16,0xFB             ;Производится инициализация портов
out PORTA,r16            ;ввода/вывода
clr r16
out DDRA,r16
out PORTC,r16
ser r16
out DDRC,r16
ldi r16,0x42
out ADMUX,r16            ;Инициализация АЦП
ldi r16,0xCF
out ADCSRA,r16
sei                      ;глобальное разрешение прерываний

main:                    ;Главный цикл программы
rjmp main                ;выполнен пустым

ADC_ready:               ;По метке ADC_ready:
cli                      ;Осуществляется глобальный запрет прерываний
in r16,SREG              ;Сохраняется регистр SREG
in r17,ADCL              ;Считываются 8 младших бит результата из ADCL
in r18,ADCH              ;и 2 старших бита из ADCH
lsr r17                  ;два младших бита результата отбрасываются
lsr r17
clr r19                  ;Два старших бита результата в РОН r18
met1:                    ;смещаются на 6 позиций влево.
lsl r18
inc r19
cpi r19,0x06
brne met1
or r17,r18               ;Происходит логическое сложение r17 и r18
out PORTC,r17            ;Результат выводится на PORTC.
```

```
ldi r17,0xCF
out ADCSRA,r17          ;Происходит перезапуск АЦП.
out SREG,r16            ;Восстанавливается регистр SREG.
sei                     ;Глобальное разрешение прерываний.
reti                    ;Выход из подпрограммы обработки прерывания.
;-----------------------------------------------------------------
```

Рассмотрим программу более подробно, показав отличия от программы, рассмотренной в примере 1.

1. В начале программы, помимо подключения библиотеки контроллера Atmega8535 указываются адреса прерываний: reset – нулевое прерывание по адресу $0, прерывание готовности результата преобразования АЦП по адресу $0E:

```
.include"m8535def.inc"
.cseg
.org$0
rjmp reset
.org$0E
rjmp ADC_ready
```

При попадании на эти векторы происходит переход на соответствующие метки reset и ADC_ready.

2. Порты ввода/вывода и указатель стека инициализируются, как и в примере 1. Однако инициализация АЦП производится по другому.

```
reset:
cli
ldi r16,low(RAMEND)
ldi r17,high(RAMEND)
out spl,r16
out sph,r17
ldi r16,0xFB
out PORTA,r16
clr r16
out DDRA,r16
out PORTC,r16
ser r16
out DDRC,r16
ldi r16,0x42
out ADMUX,r16
ldi r16,0xCF
out ADCSRA,r16
sei
```

В регистре управления мультиплексором ADMUX убирается бит «левого» выравнивания результата. Это не связано с использованием прерывания по готовности АЦП, а сделано для демонстрации работы с двумя регистрами хранения результата АЦП: ADCH и ADCL. В регистре управления АЦП ADCSRA ставится маска ADIE на прерывание готовности АЦП для его активации. После всех установок необходимо осуществить глобальное разрешение прерываний (`sei`).

3. В отличие от примера 1, основной цикл программы выполнен пустым:

```
main:
rjmp main
```

Это сделано по причине того, что при использовании нет необходимости занимать процессор постоянной проверкой флага прерывания готовности АЦП. При появлении флага прерывания основной цикл main будет автоматически прерван,

адрес возврата сохранен в стеке, а программа перейдет по метке ADC_ready, указанной в векторе $0E.

4. При переходе по метке ADC_ready осуществляется обработка прерывания по готовности АЦП:

```
ADC_ready:
cli
in r16,SREG
in r17,ADCL
in r18,ADCH
lsr r17
lsr r17
clr r19
met1:
lsl r18
inc r19
cpi r19,0x06
brne met1
or r17,r18
out PORTC,r17
ldi r17,0xCF
out ADCSRA,r17
out SREG,r16
sei
reti
```

При входе в подпрограмму сначала производится глобальный запрет всех прерываний (`cli`). При использовании одного прерывания этого можно не делать, однако правильно при входе в любую подпрограмму обработки прерываний остальные – запрещать.

После этого производится сохранение регистра SREG в РОН r16. Обнуление флага прерывания не производится, так как при использовании прерывания это делается автоматически.

Далее результат преобразования считывается из регистров ADCH (старший) и ADCL (младший). Поскольку результат преобразования АЦП – 10-разрядное число, в регистре ADCH (r18) будут использованы два младших бита: 0000 00XX, а в регистре ADCL (r17) – все биты: XXXX XXXX.

Для использования восьми старших битов результата необходимо сместить с потерей 0 и 1 битов регистр r17, приведя его к виду: 00XX XXXX (`lsr r17; lsr r17`), сместить два младших бита регистра r18 на 6 позиций влево, приведя его к виду: XX00 0000:

```
met1:
lsl r18
inc r19
cpi r19,0x06
brne met1
```

После этого регистры r17 и r18 складываются (`or r17,r18`), а результат выводится на порт С. Далее АЦП перезапускается, регистр SREG восстанавливается (`out SREG,r16`), производится глобальное разрешение всех прерываний (`sei`), а потом – выход из подпрограммы обработки прерывания (`reti`).

ПРИЛОЖЕНИЕ А
Система команд микроконтроллеров семейства AVR

Арифметические и логические команды

Мнемо-ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
ADD	Rd,Rr 0≤d≤31 0≤r≤31	Сложить без переноса	Rd ←Rd + Rr	Z, C, N, V, H	1
ADC	Rd.Rr 0≤d≤31 0≤r≤31	Сложить с переносом	Rd ← Rd + Rr+ C	Z, C, N, V, H	1
ADIW	Rd,K dE{24,26,28,30} 0≤K≤63	Сложить непосредственное значение со словом	Rdh:Rdl←Rdh:Rdl+K	Z, C, N, V	2
SUB	Rd,Rr 0≤d≤31 0≤r≤31	Вычесть без заема	Rd ← Rd – Rr	Z, C, N, V, H	1
SUBI	Rd, K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение	Rd ← Rd – K	Z, C, N, V, H	1
SBC	Rd, Rr 0≤d≤31 0≤r≤31	Вычесть с заемом	Rd←Rd-Rr-C	Z, C, N, V, H	1
SBCI	Rd, K 16≤d≤32 0≤K≤255	Вычесть непосредственное значение с заемом	Rd←Rd-K-C	Z, C, N, V, H	1
SBIW	Rd, K dE{24,26,28,30} O≤K≤б3	Вычесть непосредственное значение из слова	Rdh:Rdl←Rdh:Rdl-K	Z, C, N, V	2
AND	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить логическое AND	Rd ← Rd • Rr	Z, N, V	1
ANDI	Rd, K 16<d<31 0<k≤255	Выполнить логическое AND	Rd ← Rd • K	Z, N, V	1
OR	Rd, Rr 0≤d≥31 0≤r≤31	Выполнить логическое OR	Rd ←Rd v Rr	Z, N, V	1
ORI	Rd, K 16≤d≤31 0≤K≤255	Выполнить логическое OR с непосредственным значением	Rd ←Rd v K	Z, N, V	1
EOR	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить исключающее OR	Rd ←Rd Rr	Z, N, V	1

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

Команды сдвигов и операций с битами

Мнемо ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
LSL	Rd 0≤d≤31	Логически сдвинуть влево	Rd(n+1)←Rd(n), Rd(0)←0, C←Rd(7)	Z,C,N,V,H	1
LSR	Rd 0≤d≤31	Логически сдвинуть вправо	Rd(n)←Rd(n+1), Rd(7) ←0, C←Rd(0)	Z,C,N,V	1
ROL	Rd 0≤d≤31	Сдвинуть влево через перенос	Rd(0) ←C, Rd(n+1)←Rd(n), C←Rd(7)	Z,C,N,V,H	1

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
ROR	Rd 0≤d≤31	Сдвинуть вправо через перенос	Rd(7) ←C, Rd(n)←Rd(n+1), C ← Rd(0)	Z,C,N,V	1
ASR	Rd 0≤d≤31	Арифметически сдвинуть вправо	Rd(n)←Rd(n+1), n=0...6, Rd(0)←C	Z,C,N,V	1
SWAP	Rd 0≤d≤31	Поменять нибблы местами	Rd(3...0) <–>Rd(7...4)	Нет	1
BSET	s 0≤s≤7	Установить флаг	SREG(s)← 1	SREG(s)	1
BCLR	s 0≤s≤7	Очистить флаг	SREG(s)← 0	SREG(s)	1
SBI	P,b O≤P≤31 0≤b≤7	Установить бит в регистр I/O	l/0(P,b)← 1	Нет	2
CBI	P,b 0≤P≤31 0≤b≤7	Очистить бит в регистре I/O	l/0(P,b)← 0	Нет	2
BST	Rd,b 0≤d≤31 0≤b≤7	Переписать бит из регистра во флаг T	T← Rd(b)	T	1
BLD	Rd,b 0≤d≤31 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
CLI		Очистить флаг глобального прерывания	I←0	I	1
SES		Установить флаг знака	S← 1	S	1

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
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 полу переноса	H← 1	H	1
CLH		Очистить флаг полу переноса	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 0≤d≤31 0≤r≤31	Копировать регистр	Rd←Rr	Нет	1
LDI	Rd,k 16≤d≤31 0≤k≤255	Загрузить непосредственное значение	Rd←K	Нет	1
LDS	Rd,k 0≤d≤31 0≤k≤65535	Загрузить из ОЗУ	Rd←(k)	Нет	3
LD	Rd,X 0≤d≤31	Загрузить косвенно	Rd ← (X)	Нет	2
LD	Rd,X+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (X), X←X+1	Нет	2
LD	Rd,X- 0≤d≤31	Загрузить косвенно с преддекрементом	X←X-1, Rd ← (X)	Нет	2

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

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
ST	-Z,Rr 0≤r≤31	Записать косвенно с преддекрементом	Z←Z-1, (Z) ← Rr	Нет	2
STD	Z+q,Rr 0≤r≤31 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 0≤d≤31 0≤P≤63	Загрузить данные из порта I/O в регистр	Rd←P	Нет	1
OUT	P,Rr 0≤r≤31 0≤P≤63	Записать данные из регистра в порт I/O	P←Rr	Нет	1
PUSH	Rr 0≤r≤31	Сохранить регистр в стеке	STACK ←Rr	Нет	2

Команды переходов

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
RJMP	k -2K<k<2K	Перейти относительно	PC ←PC + k + 1	Нет	2
LJMP		Перейти косвенно	PC←Z	Нет	2
JMP	k 0<k<4M	Перейти	PC←k	Нет	3
RCALL	k -2K≤k≤2K	Вызвать подпрограмму относительно	PC ← PC + k + 1	Нет	3
ICALL		Вызвать подпрограмму косвенно	PC←Z	Нет	3
CALL	k 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 0≤d≤31, 0≤r≤31	Сравнить и пропустить, если равно	If Rd=Rr then PC ← PC + 2 (or 3)	Нет	½/3
SBRC	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре очищен	if Rr(b)=0 then PC ← PC + 2 (or 3)	Нет	½/3

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
SBRS	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре установлен	If Rr(b)=1 then PC ←PC + 2 (or 3)	Нет	½/3
SBIC	P,b 0≤P≤31 0≤b≤7	Пропустить, если бит в регистре I/O очищен	if l/O P(b)=0 then PC ← PC + 2 (or 3)	Нет	½/3
SBIS	P,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре I/O установлен	If l/O P(b)=1 then PC ←PC + 2 (or 3)	Нет	½/3
BRBS	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса установлен	if SREG(s)=1 then PC ← PC + k + 1	Нет	½
BRBC	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса очищен	if SREG(s)=0 then PC ← PC + k + 1	Нет	½
BREQ	k -64≤k≤+63	Перейти, если равно	if Rd=Rr (Z=1) then PC ← PC + k + 1	Нет	½
BRNE	k -64≤k≤+63	Перейти, если не равно	if Rd≠Rr(Z=0) then PC<←PC+ k+ 1	Нет	½
BRCS	k -64≤k≤+63	Перейти, если флаг переноса установлен	if C=1 then PC ← PC + k + 1	Нет	1/2
BRCC	k -64≤k≤+63	Перейти, если флаг переноса очищен	if C=0 then PC ← PC + k + 1	Нет	1/2
BRSH	K -64≤k≤+63	Перейти, если равно или больше (без знака)	if Rd<Rr(C=0) then PC←PC+ k+ 1	Нет	1/2
BRLO	k -64≤k≤+63	Перейти, если меньше (без знака)	if Rd<Rr (C=1) then PC ←PC + k + 1	Нет	1/2
BRMI	k -64≤k≤+63	Перейти, если минус	if N=1 then PC ← PC + k + 1	Нет	1/2
BRPL	k -64≤k≤+63	Перейти, если плюс	if N=0 then PC ← PC + k + 1	Нет	1/2
BRGE	k -64≤k≤+63	Перейти, если больше или равно (с учетом знака)	if Rd>Rr (N⊕V=0) then PC ← PC + k + 1	Нет	1/2
BRLT	k -64≤k≤+63	Перейти, если меньше чем (со знаком)	if Rd<Rr (N⊕V=1)then PC ←PC + k+ 1	Нет	1/2

Мнемоника	Операнды	Описание	Операция	Флаги	Кол-во циклов
BRHS	K -64≤k≤+63	Перейти, если флаг полупереноса установлен	if H=1 then PC ← PC + k + 1	Нет	1/2
BRHC	k -64≤k≤+63	Перейти, если флаг полупереноса очищен	if H=0 then PC ← PC + k + 1	Нет	1/2
BRTS	k -64≤k≤+63	Перейти, если флаг T установлен	if T=1 then PC ← PC + k + 1	Нет	1/2
BRTC	k -64≤k≤+63	Перейти, если флаг T очищен	if T=0 then PC ← PC + k + 1	Нет	1/2
BRVS	k -64≤k≤+63	Перейти, если флаг переполнения установлен	if V=1 then PC ← PC + k + 1	Нет	1/2
BRVC	k -64≤k≤+63	Перейти, если флаг переполнения очищен	if V=0 then PC ← PC + k + 1	Нет	1/2
BRIE	k -64≤k≤+63	Перейти, если глобальное прерывание разрешено	if I=1 then PC ← PC + k + 1	Нет	1/2
BRID	k -64k≤k≤+63	Перейти, если глобальное прерывание запрещено	if I=0 then PC ← PC + k + 1	Нет	1/2

ПРИЛОЖЕНИЕ Б

Таблица векторов прерываний микроконтроллера ATmega 8535

№ 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/счетчика 2 (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