Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет телекоммуникаций

Кафедра защиты информации

С. Н. Петров

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА. МИКРОКОНТРОЛЛЕРЫ AVR. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для направлений специальности 1-45 01 01-02 «Инфокоммуникационные технологии (сети инфокоммуникаций)», 1-45 01 01-05 «Инфокоммуникационные технологии (системы распределения мультимедийной информации)», 1-45 01 01-06 «Инфокоммуникационные технологии (лазерные информационно-измерительные системы)», 1-45 01 02-01 «Инфокоммуникационные системы (стандартизация, сертификация и контроль параметров)», 1-98 01 02 «Защита информации в телекоммуникациях»

Минск БГУИР 2016

УДК 004.31(076.5)
ББК 32.973.26-018.2я73
П30

Р е ц е н з е н т ы:
кафедра телекоммуникационных систем учреждения образования
«Белорусская государственная академия связи»
(протокол №9 от 06.04.2015);

начальник кафедры автоматизированных систем управления войсками
учреждения образования «Военная академия Республики Беларусь»,
кандидат технических наук, доцент А. В. Хижняк

Петров, С. Н.

П30 Цифровые и микропроцессорные устройства. Микроконтроллеры AVR. Лабораторный практикум : учеб.-метод. пособие / С. Н. Петров. – Минск : БГУИР, 2016. – 86 с. : ил.

ISBN 978-985-543-201-3.

Посвящено изучению 8-разрядных микроконтроллеров AVR семейства Mega. Рассмотрены программные средства разработки и отладки программ.

Содержит восемь лабораторных работ, каждая из которых включает краткие теоретические сведения, лабораторное задание, содержание отчета, контрольные вопросы и задания к каждой теме.

Лабораторные работы направлены на изучение архитектуры микроконтроллера, работы портов ввода/вывода, системы таймеров и прерываний, передачи данных по последовательному каналу связи, работы с периферийными устройствами. Для написания программ применяется язык ассемблера.

УДК 004.31(076.5)
ББК 32.973.26-018.2я73

ISBN 978-985-543-201-3

СОДЕРЖАНИЕ

Введение

Микропроцессорное устройство (МПУ) служит для выполнения арифметических, логических операций и операций управления, записанных в машинном коде. В данном учебно-методическом пособии рассматриваются вопросы изучения 8-разрядного микроконтроллера Atmega16 фирмы Atmel.

Особенностью устройств на базе МПУ является взаимодействие аппаратных узлов и программного обеспечения. Программная реализация алгоритма управления обеспечивается за счет циклического выполнения заданного набора команд. Программная реализация обеспечивает высокую степень гибкости, так как одно и то же МПУ путем замены управляющей программы может решать совершенно разные задачи или легко изменять набор решаемых задач.

Ядро AVR построено на технологии RISC и оптимизировано под кодирование на языке С. Это обеспечивает хорошее качество разработки с небольшими затратами времени по сравнению с разработкой программного обеспечения на ассемблере. Тем не менее, хотя написание программы на ассемблере занимает больше времени по сравнению с программами на языках высокого уровня, изучение основ программирования на ассемблере имеет ряд достоинств. Поскольку каждый ассемблер тесно связан набором команд процессора, изучение языка ассемблера дает возможность лучшего понимания деталей архитектуры, связи между аппаратным и программным уровнями, механизмов использования памяти, функционирования стека, вызова процедур. Следовательно, язык ассемблера может быть использован как инструмент для изучения архитектуры процессора.

Настоящее учебно-методическое пособие ориентировано на использование программ-симуляторов, имитирующих поведение МПУ под управлением программы. Интегрированная среда разработки (IDE) Atmel Studio 6 служит для разработки и отладки AVR-программ, в ее состав входят средства управления проектом, редактор текстов, симулятор, внутрисхемный эмулятор. Для знакомства со схемотехникой подключения к микроконтроллеру датчиков, устройств индикации, кнопок, переключателей и т. д. используется пакет аналого-цифрового моделирования Proteus 7.60 фирмы Labcenter Electronic.

Для закрепления материала по каждой теме предлагаются наборы заданий разного уровня сложности. Простые требуют незначительного изменения параметров или структуры алгоритма управления, более сложные требуют осмысленного объединения элементов алгоритмов из нескольких примеров.

1 ОБЩИЕ СВЕДЕНИЯ О МИКРОКОНТРОЛЛЕРЕ ATMEGA 16

1.1 Архитектура и структура микроконтроллера AVR Atmega 16

Микроконтроллер Atmega16 относится к семейству AVR (Advanced Virtual RISC) производства компании Atmel. Микроконтроллеры семейства AVR построены на гарвардской архитектуре, имеют 32 8-битных регистра общего назначения, в зависимости от модели поддерживают до 133 инструкций.

RISC (Reduced Instruction Set Computer) – архитектура микроконтроллера, в котором быстродействие увеличивается за счет упрощения инструкций, чтобы их декодирование было более простым, а время выполнения меньшим.

Гарвардская архитектура – архитектура, отличительным признаком которой является структура с разделенными устройствами памяти команд и данных и отдельными шинами команд и данных. Такая организация позволяет одновременно работать как с памятью программ, так и с памятью данных. Разделение шин доступа позволяет использовать для каждого типа памяти шины различной разрядности, причем способы адресации и доступа к каждому типу памяти также различны. Такая организация позволяет выполнять команды за один такт.

Еще одним решением, направленным на повышение быстродействия, является использование технологии конвейеризации. Конвейеризация заключается в том, что во время исполнения текущей команды производится выборка из памяти и дешифрация кода следующей команды. Причем длительность машинного цикла микроконтроллеров AVR составляет всего один период кварцевого резонатора.

Технические характеристики:

– flash-память программ объемом 16 Кбайт;

– оперативная память (статическое ОЗУ) объемом 1 Кбайт;

– память данных на основе ЭСППЗУ (EEPROM) объемом 256 байт (число циклов стирания/записи не менее 100 000);

– возможность программирования непосредственно в системе через последовательные интерфейсы SPI и JTAG;

– разнообразные способы синхронизации;

– статическая архитектура, минимальная тактовая частота равна нулю;

– арифметико-логическое устройство (АЛУ) подключено непосредственно к регистрам общего назначения (32 регистра);

– векторная система прерываний, поддержка очереди прерываний;

– наличие аппаратного умножителя;

– программное конфигурирование и выбор портов ввода/вывода;

– входные буферы с триггером Шмитта на всех выводах;

– на всех входах имеются индивидуально отключаемые внутренние подтягивающие резисторы сопротивлением 20 – 50 кОм;

– 131 высокопроизводительная команда;

– производительность порядка 16 MIPS при тактовой частоте 16 МГц;

– рабочее напряжение 4,5 – 5,5 В, тактовая частота 0 – 16 МГц.

Подтягивающий резистор (pull-up resistor) позволяет избежать чтения с входа различных помех, что повышает надежность устройства в целом. Однако повышает энергопотребление. В тех случаях, когда мощность входного сигнала высока, подтягивающий резистор подключать нет необходимости.

При разомкнутой кнопке задачей подтягивающего резистора является «дотягивание» напряжения на ножке порта до уровня VCC. Отсюда и название этого сопротивления. Без этого сопротивления ножка порта остается висеть в воздухе (при разомкнутой кнопке), т. е. не соединена ни с GND, ни с VCC. Этого состояния следует избегать, т. к. всякие наводки могут привести к неправильному состоянию конкретного входа. Подтягивающий резистор обеспечивает, таким образом, логическую единицу при разомкнутой кнопке.

Подтягивающее сопротивление имеет смысл только для входа.

Подтягивающее сопротивление на входе управляется с помощью регистра PORT. Таким образом, регистр PORT имеет две функции. Если порт настроен как выход, то этот регистр задает логический уровень соответствующих битов. Если же порт настроен как вход, то PORT указывает на включенные/выключенные подтягивающие сопротивления.

Встроенная периферия:

– два 8-разрядных таймера/счетчика с отдельным предварительным делителем, один – с режимом сравнения;

– один 16-разрядный таймер/счетчик с отдельным предварительным делителем и режимами захвата и сравнения;

– счетчик реального времени с отдельным генератором;

– программируемый сторожевой таймер;

– аналоговый компаратор;

– четыре канала широтно-импульсного модулятора ШИМ (разрешение формируемого сигнала может составлять от 1 до 16 бит);

– 8-канальный 10-разрядный аналого-цифровой преобразователь АЦП последовательного приближения;

– 8 несимметричных каналов;

– байт-ориентированный 2-проводный последовательный интерфейс;

– полнодуплексный универсальный синхронный/асинхронный приемо-передатчик (USART);

– последовательный двухпроводный интерфейс TWI (аналог I2C);

– последовательный синхронный интерфейс SPI (ведущий/ведомый).

Структурная организация

Арифметико-логическое устройство (АЛУ), выполняющее все вычисления, подключено непосредственно к 32 рабочим регистрам общего назначения (РОН), объединенным в регистровый файл. Благодаря этому любой РОН может использоваться практически во всех командах и как операнд-источник, и как операнд-приемник. Благодаря этому АЛУ выполняет одну операцию (чтение содержимого регистров, выполнение операции и запись результата обратно в регистровый файл) за один машинный цикл. Упрощенная архитектура ядра

AVR семейства Mega 16 показана на рисунке 1. Далее представлено описание выводов микроконтроллера.

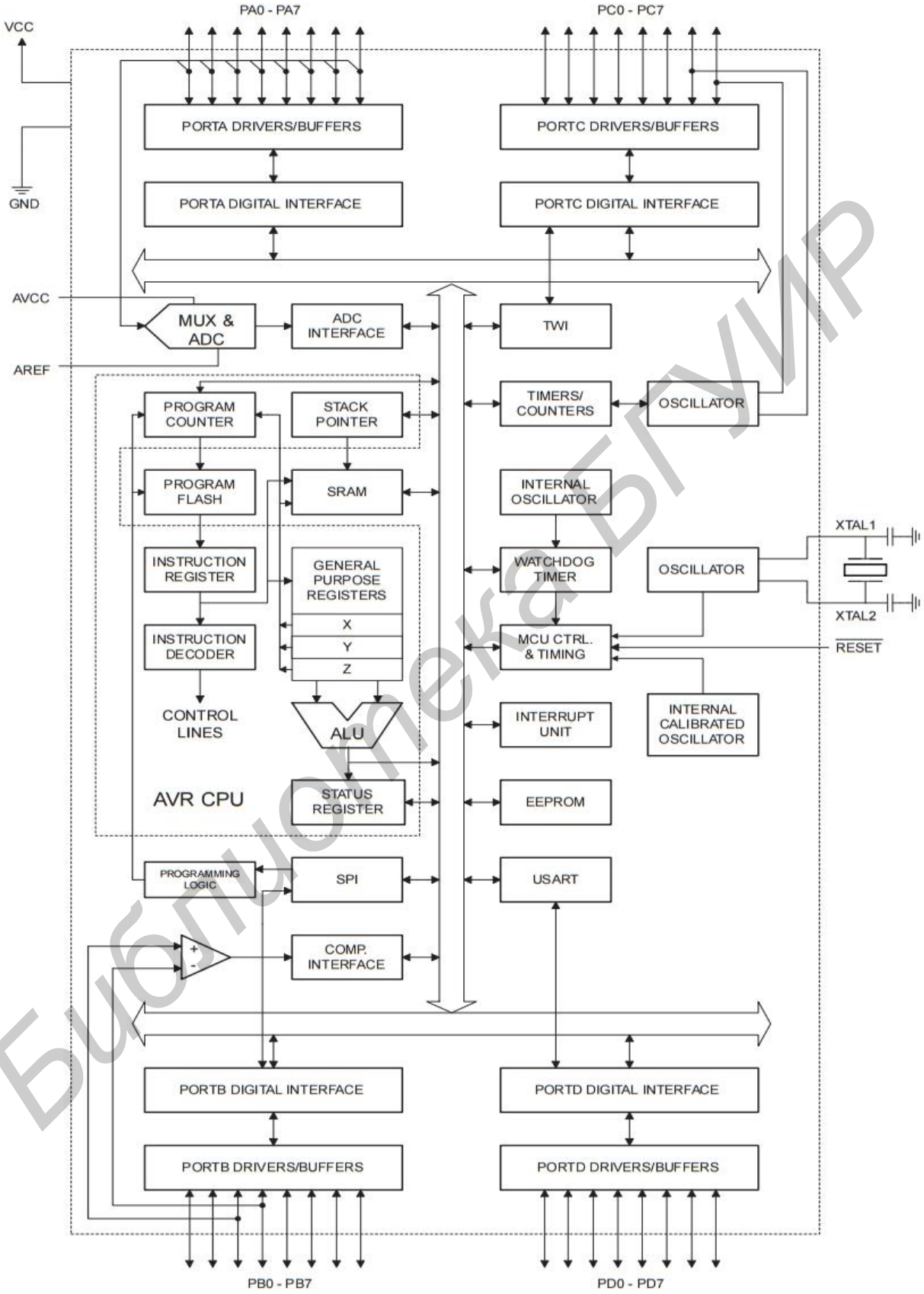

Рисунок 1 – Структурная схема микроконтроллеров AVR семейства Mega

Reset – вход сигнала сброса. Низкий уровень импульса на этом выводе, который длиннее, чем минимальная длина импульса, сгенерирует сброс, даже если тактирование не запущено.

XTAL1 – вход для инвертирующего усилителя генератора и вход для внутренней операционной схемы тактирования.

XTAL2 – выход для инвертирующего усилителя генератора.

AVCC – вывод напряжения питания для порта A и АЦП. Он должен быть внешне соединен с VCC, даже если АЦП не используется. Если АЦП используется, он должен быть соединен с VCC через фильтр низких частот.

AREF – это аналоговый установочный вывод для АЦП.

VCC – цифровое напряжение питания.

GND – земля.

1.2 Порты ввода/вывода

Микроконтроллер имеет четыре 8-разрядных параллельных порта Px (x = A, B, C, D). Выводы портов тристабильны.

Port A (PA7 – PA0). Порт A выступает в качестве аналоговых входов на АЦП. Порт A также служит в качестве 8-битного двунаправленного порта ввода/вывода, в случае когда АЦП не используется. Выводы порта могут обеспечить внутренние нагрузочные резисторы (выбираемые для каждого бита). Выходные буферы порта A имеют симметричные характеристики привода с высокой способностью стока и истока. Когда выводы PA0-PA7 используются как входы и на них небольшая внешняя нагрузка, они будут источниками тока, если нагрузочные резисторы активированы.

Port B (PB7 – PB0). Порт B – это двунаправленный порт ввода/вывода с внутренними нагрузочными резисторами (выбранными для каждого бита). Выходные буферы порта B имеют симметричные характеристики привода с высокой способностью стока и истока.

Port C (PC7 – PC0). Порт C – это двунаправленный порт ввода/вывода с внутренними нагрузочными резисторами (выбранными для каждого бита). Выходные буферы порта C имеют симметричные характеристики привода с высокой способностью стока и истока. Если включен интерфейс JTAG нагрузочные резисторы на выводах PC5(TDI), PC3(TMS) и PC2(TCK) будут активированы даже если произойдет сброс. Порт C используется для подключения интерфейса JTAG.

Port D (PD7 – PD0). Порт D – это двунаправленный порт ввода/вывода с внутренними нагрузочными резисторами (выбранными для каждого бита). Выходные буферы порта D имеют симметричные характеристики привода с высокой способностью стока и истока

Помимо функций ввода или вывода информации все порты имеют альтернативные функции. При использовании альтернативных функций выводов, регистры DDRx и PORTx должны быть установлены в соответствии с описанием альтернативных функций.

Каждый порт микроконтроллера AVR имеет восемь разрядов, каждый из которых привязан к определенной ножке корпуса. Каждый порт имеет три специальных регистра **DDRx, PORTx** и **PINx** (где x соответствует букве порта A, B, C или D). Назначение регистров представлено ниже.

DDRx – (Data Direction Register) настройка разрядов порта x на вход или выход. Отдельно взятый бит этого регистра отвечает за настройку отдельно взятого вывода этого порта.

PORTx – управление состоянием выходов порта x (если соответствующий разряд настроен как выход) или подключением внутреннего подтягивающего резистора (если соответствующий разряд настроен как вход).

PINx – чтение логических уровней разрядов порта x.

Регистр DDRx выбирает направление работы каждой отдельной ножки порта. Если в разряд регистра DDRx записана логическая 1, то соответствующая ножка будет сконфигурирована как выход. Нуль означает, что порт сконфигурирован как вход (состояние по умолчанию, которое устанавливается после сброса или включения питания).

Регистр PORTx управляет состоянием выводов порта «x». В зависимости от выбранного направления работы (вход или выход) порта или отдельно взятого вывода значение, записанное в регистр PORTx того же порта, может присваивать выводу различные состояния.

Если в разряд DDRx записан 0 и в соответствующий разряд PORTx также записан 0, то порт сконфигурирован как вход с высоким входным сопротивлением, что соответствует отключенному выходному состоянию (третье состояние). Если в разряд PORTx записана логическая 1 и в соответствующий разряд DDRx записана логическая 1, то порт сконфигурирован как выход и на выходе будет логическая 1. То есть биты PORTx управляют состоянием выходного порта при условии, что в соответствующий порту разряд DDRx записана логическая 1.

1.3 Система прерываний

Система прерываний (Interrupts) – одна из важнейших частей микроконтроллера. Все микроконтроллеры AVR имеют многоуровневую систему прерываний. Прерывание прекращает нормальный ход программы для выполнения приоритетной задачи, определяемой внутренним или внешним событием.

При помощи источников прерываний реализуется механизм синхронизации между процессором и периферийным устройством, т. е. процессор начнет прием данных и других действий над периферийным устройством только тогда, когда устройство будет к этому готово, путем генерации запроса на обработку прерывания (Interrupt ReQuest, IRQ) в зависимости от значения некоторого флага (состояние устройства / функции / события).

Прерывание (Interrupt) – сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается и управление передается процедуре обработки прерывания, соответствующей данному событию, после чего исполнение кода продолжается ровно с того места, где он был прерван (возвращение управления).

Процедура обработки прерывания (Interrupt Service Routine, ISR) – это функция/подпрограмма, которую следует выполнить при возникновении определенного события. Главное отличие этой процедуры от простых функций состоит в том, что вместо обычного возврата из функции (команда RET) используется возврат из прерывания (команда RETI, RETurn from Interrupt).

За каждым прерыванием закреплен вектор (ссылка), указывающий на процедуру обработки прерывания. Все векторы прерываний располагаются в самом начале памяти программ и вместе формируют таблицу векторов прерываний (Interrupt vectors table).

Каждому прерыванию соответствует определенный бит активации прерывания (Interrupt Enable bit). Таким образом, чтобы использовать определенное прерывание, следует записать в его бит активации прерывания логическую 1. Далее микроконтроллер не начнет обработку этих прерываний, пока в бит всеобщего разрешения прерываний (Global Interrupt Enable bit в регистре состояния SREG) не будет записана логическая 1. Чтобы запретить все прерывания в бит всеобщего разрешения прерываний следует записать логический 0.

В Atmega 16 предусмотрен 21 источник прерываний. Эти прерывания и сброс имеют различные векторы в области памяти программ. Каждому из прерываний присвоен отдельный бит, разрешающий данное прерывание при установке бита в 1, если бит I регистра состояния SREG разрешает общее обслуживание прерываний.

Для каждого события может быть установлен приоритет. Понятие приоритет означает, что выполняемая подпрограмма прерывания может быть прервана другим событием только при условии, что оно имеет более высокий приоритет, чем текущее. В противном случае центральный процессор перейдет к обработке нового события только после окончания обработки предыдущего.

Самые младшие адреса памяти программ определены как векторы сброса и прерываний. Полный список векторов прерываний приведен в таблице 1. Этот список определяет и приоритет различных прерываний. Меньшие адреса соответствуют более высокому уровню приоритета. Самый высокий уровень у RESET (сброс), следующий приоритет у INT0 – внешнего запроса прерывания 0 и т. д.

Таблица 1 – Векторы прерываний Atmega 16

Номер вектора	Адрес	Источник	Описание прерывания
1	$000	RESET	Вывод сброса и сброс от сторожевого таймера
2	$002	INT0	Внешнее прерывание 0
3	$004	INT1	Внешнее прерывание 1
4	$006	TIMER2 COMP	Совпадение таймера/счетчика 2
5	$008	TIMER2 OVF	Переполнение таймера/счетчика 2
6	$00A	TIMER1 CAPT	Захват таймера/счетчика 1
7	$00C	TIMER1 COMPA	Совпадение таймера/счетчика 1
8	$00E	TIMER1 COMPB	Совпадение таймера/счетчика 1
9	$010	TIMER1 OVF	Переполнение таймера/счетчика 1
10	$012	TIMER0 OVF	Переполнение таймера/счетчика 0
11	$014	SPI, STC	Полная последовательная передача
12	$016	USART, RXC	Usart, Rx complete
13	$018	USART, UDRE	Usart регистр данных пустой
14	$01A	USART, TXC	Usart, Tx complete
15	$01C	ADC	Аналого-цифровой преобразователь
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Аналоговый компаратор
18	$022	TWI	Двухпроводной последовательный интерфейс
19	$024	INT2	Внешнее прерывание 2
20	$026	TIMER0 COMP	Совпадение таймера/счетчика 0
21	$028	SPM_RDY	Загрузка программной памяти готова

1.4 Таймеры/счетчики

Таймеры/счетчики (TIMER/COUNTERS) можно использовать для точного формирования временных интервалов, подсчета импульсов на выводах микроконтроллера, формирования последовательности импульсов, тактирования приемопередатчика последовательного канала связи. В режиме ШИМ (PWM) таймер/счетчик может представлять собой широтно-импульсный модулятор и использоваться для генерирования сигнала с программируемыми частотой и скважностью. Таймеры/счетчики способны вырабатывать запросы прерываний, переключая процессор на их обслуживание по событиям и освобождая его от необходимости периодического опроса состояния таймеров. Поскольку основное применение микроконтроллеры находят в системах реального времени, таймеры/счетчики являются одним из наиболее важных элементов.

В МК Atmega 16 есть три таймера/счетчика – два 8-битных (Timer/Counter0, Timer/Counter2) и один 16-битный (Timer/Counter1). Восьмибитные таймеры/счетчики T0 и T2 могут использоваться для отсчета временных интервалов или как счетчики внешних событий, также у них имеется по одному блоку сравнения, позволяющему реализовать 1-канальный генератор ШИМ-сигнала.

Каждый из таймеров/счетчиков содержит специальные регистры. Например, таймер TC0 содержит:

– счетный регистр TCNT0;
– регистр сравнения OCR0 (Output Compare Register);
– конфигурационный регистр TCCR0 (Timer/Counter Control Register).
Три регистра относятся ко всем трем таймерам Atmega16:
– конфигурационный регистр TIMSK (Timer/Counter Interrupt Mask Register);
– статусный регистр TIFR (Timer/Counter Interrupt Flag Register);
– регистр специальных функций SFIOR (Special Function IO Register).

TCNTn (n – число 0, 1 или 2) – счетный регистр. Когда таймер работает, по каждому импульсу тактового сигнала значение TCNT0 изменяется на единицу. В зависимости от режима работы таймера счетный регистр может или увеличиваться, или уменьшаться. Регистр TCNT0 можно как читать, так и записывать (для задания начального значения) (рисунок 2).

бит	7	6	5	4	3	2	1	0
чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 2 – Структура регистра TCNT0

OCRn – регистр сравнения, в который возможно записать какое-либо число (рисунок 3). Его значение постоянно сравнивается со счетным регистром TCNT0, и в случае совпадения таймер может выполнять какие-то действия, например, вызывать прерывание, менять состояние вывода OC0 и т. д. в зависимости от режима работы.

бит	7	6	5	4	3	2	1	0
чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 3 – Структура регистра OCRn

TCCR0 – конфигурационный регистр таймера/счетчика T0 определяет источник тактирования таймера, коэффициент предделителя, режим работы таймера/счетчика T0 и поведение вывода OC0. Очень важный регистр (рисунок 4).

бит	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
чтение/запись	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 4 – Структура регистра TCCR0

Биты **CS02, CS01, CS00 (Clock Select)** – определяют источник тактовой частоты для таймера T0 и задают коэффициент предделителя. Все возможные состояния описаны в таблице 2.

Таблица 2 – Выбор тактовой частоты таймера T0

CS02	CS01	CS00	Описание
0	0	0	Источника тактирования нет, таймер остановлен
0	0	1	Тактовая частота микроконтроллера
0	1	0	Тактовая частота микроконтроллера /8
0	1	1	Тактовая частота микроконтроллера /64
1	0	0	Тактовая частота микроконтроллера /256
1	0	1	Тактовая частота микроконтроллера /1024
1	1	0	Внешний источник на выводе T0. Срабатывание по заднему фронту
1	1	1	Внешний источник на выводе T0. Срабатывание по переднему фронту

Биты **WGM10, WGM00** (**Wave Generator Mode**) определяют режим работы таймера/счетчика T0. Всего их может быть четыре – нормальный режим (normal), сброс таймера при совпадении (CTC) и два режима широтно-импульсной модуляции (FastPWM и Phase Correct PWM). Все возможные значения описаны в таблице 3.

Таблица 3 – Выбор режима работы таймера/счетчика T0

WGM01	WGM00	Режим работы таймера/счетчика
0	0	Normal
0	1	PWM, Phase Correct
1	0	CTC
1	1	Fast PWM

Биты **COM01, COM00** (Compare Match Output Mode) определяют поведение вывода OC0. Если хоть один из этих битов установлен в 1, то вывод OC0 перестает функционировать как обычный вывод общего назначения и подключается к схеме сравнения таймера счетчика T0. Однако при этом он должен быть еще настроен как выход. Поведение вывода OC0 зависит от режима работы таймера/счетчика T0. В режимах normal и CTC вывод OC0 ведет себя одинаково, а вот в режимах широтно-импульсной модуляции его поведение отличается.

Последний бит регистра TCCR0 – это бит **FOC0** (Force Output Compare). Этот бит предназначен для принудительного изменения состояния вывода OC0. Он работает только для режимов Normal и CTC. При установке бита FOC0 в единицу состояние вывода меняется соответственно значениям битов COM01, COM00. Бит FOC0 не вызывает прерывания и не сбрасывает таймер в CTC-режиме.

TIMSK (Timer/Counter Interrupt Mask Register) – регистр масок прерываний по таймерам/счетчикам (рисунок 5). Общий регистр для всех трех таймеров Atmega 16 содержит флаги разрешения прерываний. Таймер T0 может

вызывать прерывания при переполнении счетного регистра TCNT0 и при совпадении счетного регистра с регистром сравнения OCR0. Соответственно для таймера T0 в регистре TIMSK зарезервированы два бита – это TOIE0 и OCIE0. Остальные биты относятся к другим таймерам.

бит	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
чтение/запись	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 5 – Структура регистра TIMSK

TOIE0 – значение бита 0 запрещает прерывания по событию переполнение, а 1 – разрешает.

OCIE0 – значение 0 запрещает прерывания по событию совпадение, а 1 – разрешает.

Естественно прерывания будут вызываться, только если установлен бит глобального разрешения прерываний – бит I регистра SREG.

TIFR (Timer/Counter0 Interrupt Flag Register) – регистр флагов прерываний таймеров (рисунок 6).

бит	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 6 – Структура регистра TIFR

Общий для всех трех таймеров-счетчиков регистр. Содержит статусные флаги, которые устанавливаются при возникновении событий. Для таймера T0 – это переполнение счетного регистра TCNT0 и совпадение счетного регистра с регистром сравнения OCR0.

Если в эти моменты в регистре TIMSK разрешены прерывания и установлен бит I в регистре SREG, то микроконтроллер вызовет соответствующий обработчик.

Флаги автоматически очищаются при запуске обработчика прерывания. Также это можно сделать программно, записав 1 в соответствующий флаг.

TOV0 – устанавливается в 1 при переполнении счетного регистра.

OCF0 – устанавливается в 1 при совпадении счетного регистра с регистром сравнения.

SFIOR (Special Function IO Register) – регистр специальных функций ввода/вывода (рисунок 7). Один из его разрядов сбрасывает 10-разрядный двоичный счетчик, который делит входную частоту для таймера T0 и таймера

T1. Сброс осуществляется при установке бита **PSR10 (**Prescaler Reset Timer/Counter1 и Timer/Counter0) в 1.

бит	7	6	5	4	3	2	1	0
	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10
чтение/запись	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
начальное значение	0	0	0	0	0	0	0	0

Рисунок 7 – Структура регистра SFIOR

Сторожевой таймер (WatchDog Timer, WDT**)** предназначен для предотвращения аварийных последствий от случайных сбоев программы. Он имеет свой собственный RC-генератор, работающий на частоте 1 МГц. Как и для основного внутреннего RC-генератора, значение 1 МГц является приближенным и зависит прежде всего от величины напряжения питания.

Идея использования сторожевого таймера состоит в регулярном его сбрасывании под управлением программы или внешнего воздействия, до того как закончится его выдержка времени и не произойдет сброс процессора. Если программа работает нормально, то команда сброса сторожевого таймера должна регулярно выполняться, предохраняя поцессор от сброса.

Если микропроцессор случайно вышел за пределы программы (например от сильной помехи) либо зациклился на каком-либо участке программы, команда сброса сторожевого таймера, скорее всего, не будет выполнена в течение достаточного времени и произойдет полный сброс процессора, инициализирующий все регистры и приводящий систему в рабочее состояние.

1.5 Последовательные порты

Универсальный синхронный/асинхронный приемопередатчик (Universal Synchronous/Asynchronous Receiver and Transmitter, UART или USART) – удобный и простой последовательный интерфейс для организации информационного канала обмена данными между микроконтроллером и внешним миром. Способен работать в дуплексном режиме (одновременная передача и прием данных). Поддерживает протокол стандарта RS-232, что обеспечивает возможность организации связи с персональным компьютером.

Модули USART, реализованные в микроконтроллерах семейства, могут обнаруживать следующие внештатные ситуации: переполнение; ошибка кадрирования; ошибка контроля четности.

Для уменьшения вероятности сбоев в модуле реализована функция фильтрации помех. Для взаимодействия с программой предусмотрены три прерывания, запрос на генерацию которых формируется при наступлении следующих событий: «передача завершена», «регистр данных передатчика пуст» и «прием завершен».

Модуль состоит из трех основных частей: блока тактирования, блока передатчика и блока приемника.

Блок тактирования модулей USART содержит схему синхронизации (для работы в синхронном режиме) и контроллер скорости передачи.

Блок передатчика включает одноуровневый буфер, сдвиговый регистр, схему формирования бита четности и схему управления. Блок приемника, в свою очередь, содержит схемы восстановления тактового сигнала и данных, схему контроля четности, двухуровневый буфер, сдвиговый регистр, а также схему управления.

Буферные регистры приемника и передатчика располагаются по одному адресу пространства ввода/вывода и обозначаются как регистр данных UDR. В этом регистре хранятся младшие 8 бит принимаемых и передаваемых данных. При чтении регистра UDR выполняется обращение к буферному регистру приемника, при записи – к буферному регистру передатчика. В модулях USART буфер приемника является двухуровневым (FIFO-буфер), изменение состояния которого происходит при любом обращении к регистру UDR. В связи с этим не следует использовать регистр UDR в качестве операндов команд типа «чтение/модификация/запись» (SBI и CBI). Кроме того, следует быть аккуратными при использовании команд проверки SBIC и SBIS, поскольку они также изменяют состояние буфера приемника.

Для управления модулями USART используются три регистра: UCSRA, UCSRB и UCSRC (таблицы 4–7).

Таблица 4 – UCSRA (регистр управления)

№ бита	7	6	5	4	3	2	1	0
Имя бита	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
Доступность	R	R\W	R	R	R	R	R\W	R\W

Таблица 5 – Описание разрядов регистра UCSRA

Разряд	Название	Описание
7	RXC	Флаг завершения приема. Флаг устанавливается в 1 при наличии непрочитанных данных в буфере приемника (регистр данных UDR). Сбрасывается флаг аппаратно после опустошения буфера (в UART – после прочтения регистра данных). Если разряд RXCIE регистра UCSRB установлен, то при установке флага генерируется запрос на прерывание – «прием завершен»
6	TXC	Флаг завершения передачи. Флаг устанавливается в 1 после передачи всех разрядов посылки из сдвигового регистра передатчика при условии, что в регистр данных UDR не было загружено нового значения. Если разряд TXCIE регистра UCSRB (UCSRnB) установлен, то при установке флага генерируется прерывание «передача завершена». Флаг сбрасывается аппаратно при обработке прерывания или программно – записью в него логической 1

Разряд	Название	Описание
5	UDRE	Флаг опустошения регистра данных. Данный флаг устанавливается в 1 при пустом буфере передатчика (после пересылки байта из регистра данных UDR в сдвиговый регистр передатчика). Установленный флаг означает, что в регистр данных можно загружать новое значение. Если разряд UDRIE регистра UCR (UCSRB) установлен, генерируется запрос на прерывание – «регистр данных пуст». Флаг сбрасывается аппаратно при записи в регистр данных
4	FE	Флаг ошибки кадрирования. Флаг устанавливается в 1 при обнаружении ошибки кадрирования, если первый стоп-бит принятой посылки равен 0. Флаг сбрасывается при приеме стоп-бита, равного 1
3	DOR	Флаг переполнения. В USART флаг устанавливается в 1, если в момент обнаружения нового старт-бита в сдвиговом регистре приемника находится последнее принятое слово, а буфер приемника полон. В UART флаг устанавливается в 1, если новый кадр будет помещен в сдвиговый регистр приемника до того, как из регистра данных будет считано предыдущее слово. Флаг сбрасывается при пересылке принятых данных из сдвигового регистра приемника в буфер
2	PE	Флаг ошибки контроля четности. Флаг устанавливается в 1, если в данных, находящихся в буфере приемника, выявлена ошибка контроля четности
1	U2X	Удвоение скорости обмена. Если этот разряд установлен в 1, коэффициент деления предделителя контроллера скорости передачи уменьшается с 16 до 8, удваивая тем самым скорость асинхронного обмена по последовательному каналу. В USART разряд U2X используется только при асинхронном режиме работы
0	MPCM	Режим мультипроцессорного обмена. Разряд MPCM используется в режиме мультипроцессорного обмена. Если он установлен в 1, ведомый микроконтроллер ожидает приема кадра, содержащего адрес

Таблица 6 – Описание разрядов регистра UCSRB

Разряд	Название	Описание
7	RXCIE	Разрешение прерывания по завершении приема. Если разряд установлен в 1, то при установке флага RXC регистра UCSRA генерируется прерывание «прием завершен» (если флаг I регистра SREG установлен в 1)
6	TXCIE	Разрешение прерывания по завершении передачи. Если данный разряд установлен в 1, то при установке флага TXC регистра UCSRA генерируется прерывание «передача завершена» (если флаг I регистра SREG установлен в 1)
5	UDRIE	Разрешение прерывания при очистке регистра данных UART. Если данный разряд установлен в 1, то при установке флага UDRE регистра UCSRA генерируется прерывание – «регистр данных пуст» (если флаг I регистра SREG установлен в 1)
4	RXEN	Разрешение приема. При установке этого разряда в 1 разрешается работа приемника USART/UART и переопределяется функционирование вывода RXD (RXDn). При сбросе разряда RXEN работа приемника запрещается, а его буфер сбрасывается. Значения флагов TXC, DOR/OR и FE при этом становятся недействительными

Разряд	Название	Описание
3	TXEN	Разрешение передачи. При установке этого разряда в 1 разрешается работа передатчика UART и переопределяется функционирование вывода TXD. Если разряд сбрасывается в 0 во время передачи, выключение передатчика произойдет только после завершения передачи данных, находящихся в сдвиговом регистре и буфере
2	UCSZ2	Формат посылок. Этот разряд используется для задания размера передаваемых слов данных. В модулях USART он используется совместно с разрядами UCSZ1:0 регистра UCSRC. В модулях UART, если разряд CHR9 установлен в 1, осуществляется передача и прием девятиразрядных данных, если сброшен – восьмиразрядных
1	RXB8	8-й разряд принимаемых данных. При использовании девятиразрядных слов данных этот разряд содержит значение старшего разряда принятого слова. В случае USART содержимое этого разряда должно быть считано до прочтения регистра данных UDR
0	TXB8	8-й разряд передаваемых данных. При использовании девятиразрядных слов данных содержимое этого разряда является старшим разрядом передаваемого слова. Требуемое значение должно быть занесено в этот разряд до загрузки байта данных в регистр UDR

Таблица 7 – Описание разрядов регистра UCSRC

Разряд	Название	Описание
7	URSEL	Выбор регистра. Этот разряд определяет, в какой из регистров модуля производится запись. Если разряд установлен в 1, обращение производится к регистру UCSRC. Если же разряд сброшен в 0, обращение производится к регистру UBRRH
6	UMSEL	Режим работы USART. 0 – модуль USART работает в асинхронном режиме; 1 – модуль USART работает в синхронном режиме
5	UPM1	Режим работы контроля и формирования четности. Разряды определяют функционирование схем контроля и формирования четности
4	UPM0	
3	USBS	Этот разряд определяет количество стоп-битов, посылаемых передатчиком. Разряд в «0» – передатчик посылает 1 стоп-бит, если в 1, то 2 стоп-бита. Для приемника содержимое этого разряда безразлично
2	UCSZ1	Формат посылок. Совместно с разрядом UCSZ2 эти разряды определяют количество разрядов данных в посылках (размер слова)
1	UCSZ0	
0	UCPOL	Полярность тактового сигнала. Значение этого разряда определяет момент выдачи и считывания данных на выводах модуля. Используется только в синхронном режиме. В асинхронном режиме разряд сброшен в 0

Скорость приема/передачи USART

В асинхронном режиме, а также в синхронном режиме при работе в качестве ведущего скорость приема и передачи данных задается контроллером скорости передачи, функционирующим как делитель системного тактового сигнала с программируемым коэффициентом деления. Коэффициент определяется содержимым регистра контроллера UBRR (таблица 8). В блок приемника сформированный сигнал поступает сразу, а в блок передатчика – через дополнительный делитель, коэффициент деления которого (2, 8 или 16) зависит от режима работы модуля USART/UART. Регистр UBRR является

12-разрядным и физически размещается в двух регистрах ввода/вывода UBRRH и UBRRL.

При работе в асинхронном режиме скорость обмена определяется не только содержимым регистра UBRR, но и состоянием разряда U2X регистра UCSRA. Если этот разряд установлен в 1, коэффициент деления предделителя уменьшается в два раза, а скорость обмена соответственно удваивается. При работе в синхронном режиме этот разряд должен быть сброшен.

Скорость обмена определяется следующими формулами:

– асинхронный режим (обычный, U2Xn = 0) BAUD = fCK/16(UBRR + 1);

– асинхронный режим (ускоренный, U2Xn = 1) BAUD = fCK/8(UBRR +1);

– синхронный режим ведущего BAUD = fCK/2(UBRR + 1).

Здесь BAUD – скорость передачи в бодах; fCK – тактовая частота микроконтроллера; UBRR – содержимое регистра контроллера скорости передачи (0–4095).

Таблица 8 – Определение размера слова данных

UCSZ2	UCSZ1	UCSZ0	Размер слова данных
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	0	0	Зарезервировано
1	0	1	Зарезервировано
1	1	0	Зарезервировано
1	1	1	9 бит

Последовательный периферийный трехпроводный интерфейс SPI (Serial Peripheral Interface) предназначен для организации обмена данными между двумя устройствами. С его помощью может осуществляться обмен данными между микроконтроллером и различными устройствами, такими, как цифровые потенциометры, ЦАП/АЦП, FLASH-ПЗУ и др. Кроме того, через интерфейс SPI может осуществляться программирование микроконтроллера.

Двухпроводной последовательный интерфейс TWI (Two-wire Serial Interface) является полным аналогом базовой версии интерфейса I2C (двухпроводная двунаправленная шина). Этот интерфейс позволяет объединить вместе до 128 различных устройств с помощью двунаправленной шины, состоящей из линии тактового сигнала (SCL) и линии данных (SDA).

Аналого-цифровой преобразователь (A/D CONVERTER) служит для получения числового значения напряжения, поданного на его вход. Этот результат сохраняется в регистре данных АЦП. Какой из выводов микроконтроллера будет являться входом АЦП, определяется числом, занесенным в соответствующий регистр.

Расположение выводов микроконтроллера (цоколевка) в варианте исполнения корпуса DIP показано на рисунке 8. Упрощенная архитектура ядра микроконтроллеров AVR семейства Mega показана на рисунке 9.

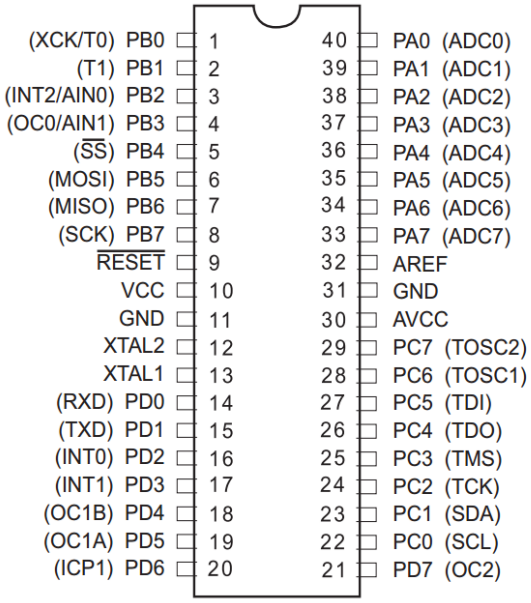

Рисунок 8 – Расположение выводов микроконтроллера Atmega 16

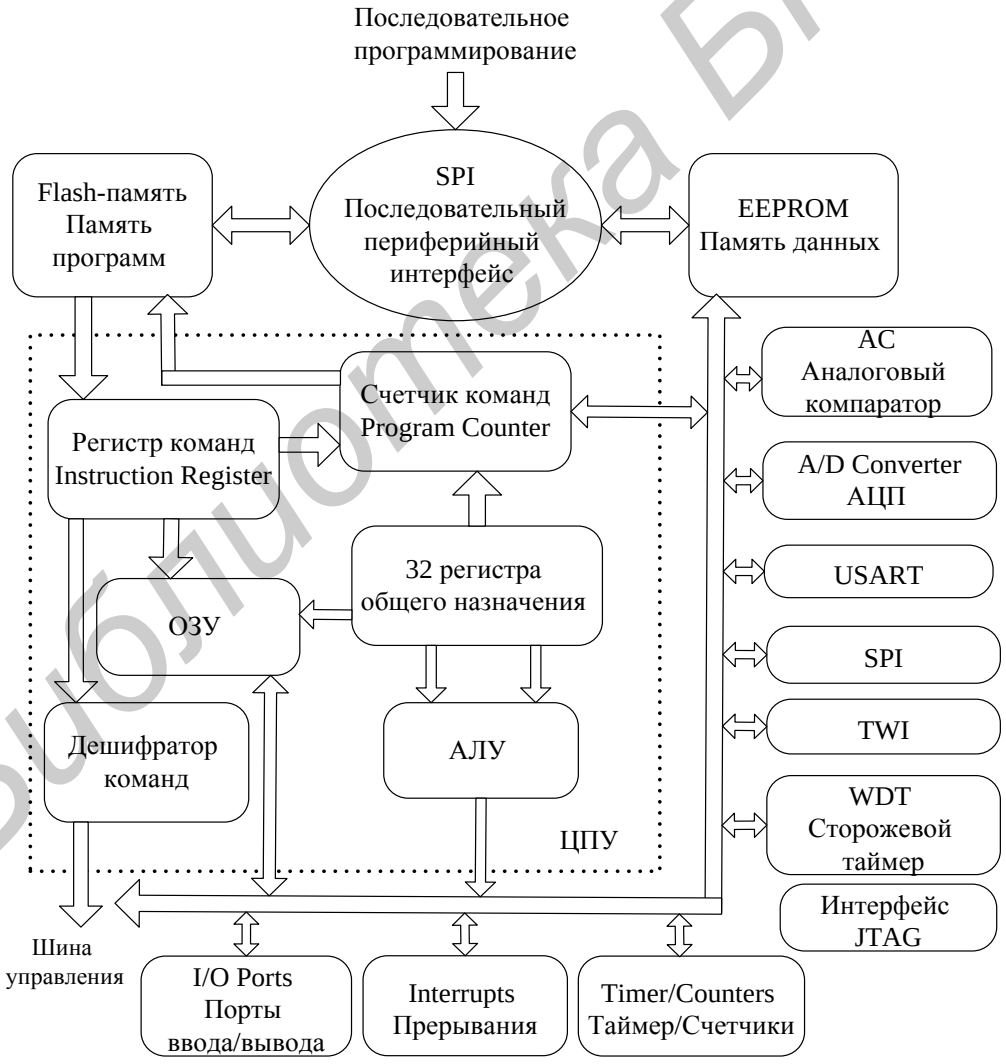

Рисунок 9 – Упрощенная архитектура ядра микроконтроллеров семейства Mega

Аналоговый компаратор (Analog Comparator, AC) сравнивает напряжения на двух выводах микроконтроллера. Результатом сравнения будет логическое значение, которое может быть прочитано из программы. Выход аналогового компаратора можно включить на прерывание от аналогового компаратора. Пользователь может установить срабатывание прерывания по нарастающему или спадающему фронту или по переключению.

Четырехпроводной интерфейс JTAG используется для тестирования печатных плат, внутрисхемной отладки, программирования микроконтроллеров. Многие микроконтроллеры семейства Mega имеют совместимый с IEEE Std 1149.1 интерфейс JTAG или debugWIRE для встроенной отладки. Кроме того, все микроконтроллеры Mega с flash-памятью емкостью 16 Кбайт и более могут программироваться через интерфейс JTAG.

Тактовый генератор вырабатывает импульсы для синхронизации работы всех узлов микроконтроллера. Внутренний тактовый генератор AVR может запускаться от нескольких источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя RC-цепочка). Минимальная допустимая частота ничем не ограничена (вплоть до пошагового режима). Максимальная рабочая частота определяется конкретным типом микроконтроллера.

1.6 Организация памяти

Память МК с гарвардской архитектурой разделяется на две различные области – память программ (Program Memory) и память данных (Data Memory). Эти области разделены на два непересекающихся адресных пространства, имеют разное назначение, размер и тип ячеек. Разделение адресных пространств обеспечивается системой команд и способами адресации.

Память программ (ПП) служит для хранения кодов программы и констант. Физическим носителем ПП выступает энергонезависимое постоянное запоминающее устройство (ПЗУ) типа FlashROM. Память программ располагается только в кристалле. Объем памяти микроконтроллера Atmega 16 составляет 16 × 1024 16-битных слов. Это и определяет разрядность регистра счетчика команд (PC, Program Counter), используемого для адресации памяти программ. Размер счетчика команд составляет 12 бит в зависимости от объема адресуемой памяти.

Память программ логически разделена на две части: область прикладной программы и область загрузчика. В последней может располагаться специальная программа (загрузчик), позволяющая микроконтроллеру самостоятельно управлять загрузкой и выгрузкой прикладных программ. Если же возможность самопрограммирования у микроконтроллера не используется, прикладная программа может располагаться и в области загрузчика. Карта памяти микроконтроллера Atmega 16 приведена на рисунке 10.

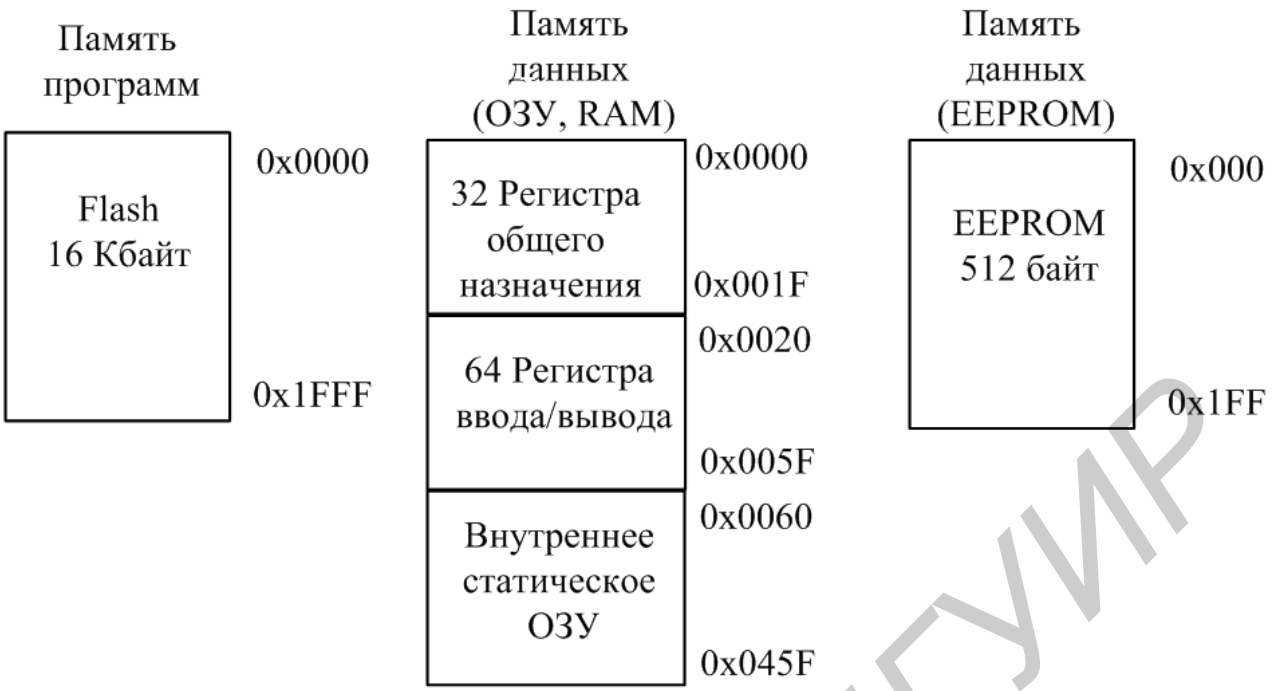

Рисунок 10 – Карта памяти микроконтроллера Atmega 16

По адресу 0x0000 памяти программ находится вектор сброса. Выполнение программы начинается с этого адреса после инициализации (сброса) микроконтроллера (по этому адресу должна размещаться команда перехода к инициализационной части программы). Начиная с адреса 0x0002 памяти программ, располагается таблица векторов прерываний.

Память данных имеет байтовую организацию и адресное пространство объемом до 64 Кбайт (16-битный адрес). Память данных микроконтроллера разделена на три части:

– регистровая память;

– оперативная память (статическое ОЗУ);

– энергонезависимое ЭСППЗУ (EEPROM).

Регистровая память включает 32 регистра общего назначения (РОН), объединенных в файл, и служебные регистры ввода/вывода (РВВ). В обеих областях регистров ввода/вывода располагаются различные служебные регистры (регистр управления микроконтроллера, регистр состояния), а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. РОН – наиболее интенсивно используемая память для размещения данных и указателей.

Статическое ОЗУ объемом 1 Кбайт используют для хранения переменных помимо РОН. Микроконтроллер имеет возможность подключения внешнего статического ОЗУ объемом до 64 Кбайт. В адресном пространстве ОЗУ также расположены все регистры микроконтроллера, под них отведены младшие 96 (256) адресов, остальные адреса отведены под ячейки статического ОЗУ.

EEPROM-память используют для долговременного хранения различной информации, которая может изменяться в процессе функционирования готовой

системы (калибровочные константы, серийные номера, ключи). Ее объем составляет 512 Кбайт. Эта память расположена в отдельном адресном пространстве, а доступ к ней осуществляется с помощью определенных регистров ввода/вывода.

Стек – это особая область памяти данных, используемая процессором для временного хранения адресов возврата из подпрограмм, промежуточных результатов вычислений и т. д. В Atmega 16 стек реализован программно. Он размещается в памяти данных и должен начинаться от максимального адреса RAM. По мере заполнения стека его граница смещается к младшим адресам (стек растет вверх, адрес в указателе стека уменьшается). Предельное минимальное значение указателя стека для микроконтроллера Atmega 16 – 0x60, поскольку дальше начинается область SFR (Special Function Registers, регистры специального назначения).

В качестве указателя стека используется пара регистров ввода/вывода SPH и SPL, расположенных по адресам 0x3E (0x5E) и 0x3D (0x5D) соответственно.

Так как после подачи напряжения питания (или после сброса) в регистрах содержится нулевое значение, в самом начале программы указатель стека необходимо проинициализировать, записав в него значение верхнего адреса памяти данных. Например:

```
ldi  r16, low(RAMEND)      ;Загрузим в рабочий регистр младший байт значения
out  SPL, rl6              ;Инициализируем указатель стека
ldi  r16, high(RAMEND)     ;Загрузим в рабочий регистр старший байт значения
out  SPH, rl6              ;Инициализируем указатель стека
```

При вызове подпрограмм адрес команды, расположенной за командой вызова, сохраняется в стеке. Значение указателя стека при этом уменьшается на 2 или 3 в зависимости от размера счетчика команд. При возврате из подпрограммы этот адрес извлекается из стека и загружается в счетчик команд. Значение указателя стека соответственно увеличивается на 2 (3). То же происходит и во время прерывания. При генерации прерывания адрес следующей команды сохраняется в стеке, а при возврате из подпрограммы обработки прерывания он восстанавливается из стека. Стек доступен и программно. Для работы со стеком имеются две команды: команда занесения в стек (PUSH) и команда извлечения из стека (POP).

1.7 Регистры управления

Регистр состояния (статуса или признаков) программ SREG содержит флаги, показывающие текущее состояние микроконтроллера (таблица 9). После сигнала сброса инициализируется нулями. Каждый из восьми разрядов (битов) регистра называется флагом, который может быть установлен в 1 (в этом случае считается, что флаг установлен) или сброшен в 0 (в этом случае считается, что флаг сброшен). Большинство флагов сбрасывается или устанавливается автоматически в зависимости от результатов выполненной операции.

Таблица 9 – SREG (регистр состояния программ)

№ бита	7	6	5	4	3	2	1	0
Имя бита	I	T	H	S	V	N	Z	C
Доступность	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Флаг I (бит 7) – общее разрешение прерываний (Global Interrupt). Для разрешения прерываний этот флаг должен быть установлен в 1. Если флаг сброшен, то прерывания запрещены независимо от состояния битов регистра масок прерываний. Флаг сбрасывается аппаратно после входа в прерывание и восстанавливается командой RETI.

Флаг T (бит 6) – хранение копируемого бита (Transfer or Copy). Этот разряд можно использовать как ячейку для временного хранения информации размерностью в один бит. Бит регистра используется в качестве источника или приемника командами копирования битов BLD (Bit LoaD) и BST (Bit STore).

Флаг H (бит 5) – флаг половинного переноса (Half Carry). Этот флаг устанавливается в 1, если произошел перенос из младшей половины байта (тетрады) или заем из старшей половины байта при выполнении некоторых арифметических операций.

Флаг S (бит 4) – флаг знака (Sign). Этот флаг равен результату операции «Исключающее ИЛИ» (XOR) между флагами N и V. Этот флаг устанавливается в 1, если результат выполнения арифметической операции меньше нуля.

Флаг V (бит 3) – флаг переполнения дополнительного кода (Two's complement Overflow). Этот флаг устанавливается в 1 при переполнении числа в дополнительном коде. Используется при работе со знаковыми числами, представленными в дополнительном коде. В дополнительном коде записываются отрицательные числа.)

Флаг N (бит 2) – флаг отрицательного значения (Negaive). Этот флаг устанавливается в 1, если старший бит результата операции равен 1. Указывает на отрицательный результат арифметической или логической операции.

Флаг Z (бит 1) – флаг нуля (Zero). Этот флаг устанавливается в 1, если результат выполнения операции равен нулю.

Флаг C (бит 0) – флаг переноса (Carry). Этот флаг устанавливается в 1, если в результате выполнения операции произошел выход за границы байта (результат занимает 9 бит).

Биты регистра управления микроконтроллера **MCUCR** управляют выполнением основных функций (таблица 10).

Таблица 10 – MCUCR (регистр управления микроконтроллером)

№ бита	7	6	5	4	3	2	1	0
Имя бита	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
Доступность	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Биты регистра MCUCR отвечают за выбор режимов энергопотребления и условия генерации внешних прерываний (таблицы 11 и 12).

Бит SE – разрешение перехода в режим пониженного энергопотребления. Установка этого бита в 1 разрешает перевод микроконтроллера в режим пониженного энергопотребления. Переключение осуществляется по команде SLEEP.

Биты SM2, SM1, SM0 – выбор режима пониженного энергопотребления. Состояние этих битов определяет, в какой режим перейдет микроконтроллер после выполнения команды SLEEP.

Таблица 11 – Режимы энергопотребления

SM2	SM2	SM2	Режим энергопотребления
0	0	0	Idle (режим холостого хода)
0	0	1	ADC Noise Reduction (режим снижения шумов АЦП)
0	1	0	Power Down (режим микропотребления)
0	1	1	Power Save (экономичный режим),
1	1	0	Standby (режим ожидания)

Биты **ISC11, ISC00** – определяют условия генерации внешних прерываний.

Таблица 12 – Условия генерации внешних прерываний

ISCn1, n = 0, 1	ISCn0, n = 0, 1	Условие
0	0	По низкому уровню на выходе INTn
1	0	По спадающему фронту на выходе INTn
1	1	По нарастающему фронту на выходе INTn

2 СИСТЕМА КОМАНД ATMEGA 16

2.1 Режимы адресации

Формат команды описывает структуру машинного слова команды. Часть битов выделяют под код операции («что делать»), это обязательная часть любой команды. Остальные биты указывают операнды («над чем это делать»), т. е. данные, адрес ячейки данных или адрес перехода. Различают команды безоперандные, одно- и двухоперандные.

В двухоперандных командах обработки данных один из операндов называется источником [source], его содержимое не изменяется, другой – приемником [destination], в него заносится результат операции. Если команда имеет два операнда, то сначала указывают приемник, затем источник. Между приемником и источником обязательно должна стоять запятая (с любым числом пробелов до или после нее или вообще без них). Например, выражение sub r16, r17 означает, что из содержимого r16 нужно вычесть содержимое r17, а результат окажется в r16.

Команда = Операция + Операнд.

Способы адресации определяют способы указания (представления) операндов.

Реализованы следующие способы адресации для охвата всей памяти:

– прямая;
– регистровая;
– непосредственная;
– битовая;
– косвенная;
– косвенная со смещением;
– косвенная с предварительным декрементом;
– косвенная с последующим инкрементом.

При **прямой адресации** числовая константа в команде обозначает адрес ячейки памяти, содержимое которой является операндом команды. Данный способ адресации может применяться при обращении к любой ячейке адресного пространства SRAM. Имеются всего две команды: LDS и STS, каждая длиной в два слова (32 разряда). Первое слово содержит код операции и адрес регистра общего назначения, второе – 16-разрядный адрес ячейки, к которой направлено обращение. Например, LDS R16, 0x0200 – занести в регистр R16 значение, хранящееся в ячейке памяти с адресом 0x0200.

Регистровая адресация подразумевает, что операнд адресуется именем регистра.

Регистровая адресация с одним регистром. При этом способе адресации данные находятся в регистре Rd. Примером команд, использующих этот метод адресации, являются команды для работы со стеком (PUSH, POP) и обмена тетрадами в регистре (SWAP).

Регистровая адресация с двумя регистрами. Данный способ адресации применяется в командах, которые используют два регистра общего назначения: Rd и Rr. Этот вид адресации используют команды пересылки данных из регистра в регистр и большинство команд арифметических операций, а также ряд команд логических операций. При этих операциях результат сохраняется в регистре Rd. Например, ADD R16, R17.

Адресация регистра ввода/вывода. Этот вид адресации используют для выполнения обмена данных между регистром ввода/вывода, расположенным в адресном пространстве ввода/вывода, и одним из регистров общего назначения по командам IN и OUT. Например, OUT DDRA, R16 – занести в регистр DDRA значение, хранящееся в регистре R16.

Непосредственная адресация. Данные указываются непосредственно в виде константы в самой команде, а не в виде адреса. Непосредственная адресация используется командой пересылки константы в регистр LDI, а также некоторыми командами арифметических и логических операций. Например, LDI R16, 255 – занести в регистр R16 число 255.

Битовая адресация. Этот вид адресации позволяет указать один из восьми битов любого из 32 регистров общего назначения или первой половины регистров ввода/вывода с номерами 0 – 31, а также регистра SREG. Для этого нужно указать имя регистра общего назначения Ri (i = 0...31) либо имя регистра ввода/вывода Pi (i = 0–31), либо имя SREG и номер бита b (b = 0–7). Командами SBI и CBI осуществляется установка в 1 и сброс в 0 указанного бита регистра ввода/вывода, командами BLD, BST – обмен значениями бита T из регистра SREG и адресованного бита из регистра общего назначения.

Помимо этого есть группа команд битовых операций, обеспечивающая установку и сброс битов регистра состояния SREG. При записи мнемоники команд допускается использование символических (штатных) имен регистров ввода/вывода и имен битов. Существует группа команд условного перехода, где в качестве условия используется либо значение бита в регистре общего назначения (SBRC, SBRS), либо значение бита в регистре ввода/вывода (SBIC, SBIS). Например, sbic PINA, 7 – пропустить следующую команду, если бит 7 в регистре PINA равен нулю.

Косвенная адресация памяти данных (indirect addressing). Последние шесть регистров общего назначения (регистры R26 – R31) используются как регистры-указатели для косвенной адресации. Для этого они объединяются в три 16-разрядных регистра X, Y и Z (рисунок 11).

При косвенной адресации обращение направлено к ячейке памяти, адрес которой находится в 16-разрядном индексном регистре X, Y или Z. В роли этих регистров выступают пары регистров R26, R27 (регистр X), R28, R29 (регистр Y) и R30, R31 (регистр Z). Иначе говоря, операндом является содержимое ячейки памяти, адрес которой записан в индексном регистре. Применяется для адресации в массивах и переменных указателях.

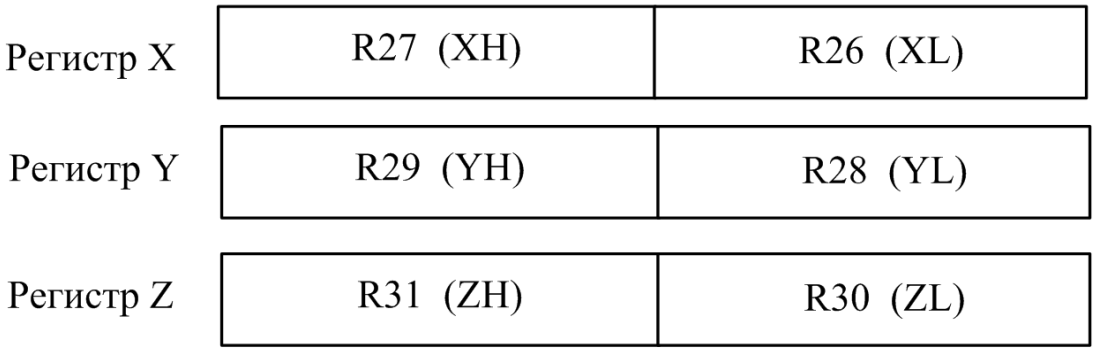

Рисунок 11 – Регистры-указатели X, Y и Z

Существует несколько разновидностей косвенной адресации.

Косвенная адресация памяти данных со смещением (indirect addressing with displacement). При этом способе адрес ячейки памяти определяется путем суммирования содержимого индексного регистра Y или Z с 6-разрядным смещением, задаваемым в команде. Этот способ адресации используют команды LDD (пересылка байта из ячейки памяти SRAM в регистр Rd) и STD (пересылка байта из регистра Rr в ячейку SRAM). Позволяет получить доступ ко всем элементам в структуре путем указания на первый элемент структуры (через адрес указателя) и добавления смещения к адресу указателя (смещение указывает на нужное поле в структуре), причем значение указателя не изменяется. Также этот режим доступа используется для доступа к переменным в программном стеке и для доступа к ячейкам массива.

Косвенная адресация памяти данных с предекрементом (indirect addressing with pre-decrement). При этом способе адресации содержимое индексного регистра X, Y или Z сначала уменьшается на 1, а затем производится обращение к памяти по полученному адресу. Этот способ адресации используют команды LD (пересылка байта данных из памяти в регистр Rd) и ST (пересылка байта данных из регистра Rr в память), всего шесть команд – по две для каждого регистра. Применяется для эффективного доступа к элементам массива и к переменным указателя, когда нужен декремент указателя до доступа.

Косвенная адресация памяти данных с постинкрементом (indirect addressing with post-increment). При этом способе адресации содержимое индексного регистра X, Y или Z сначала используется в качестве адреса обращения к памяти данных, а затем увеличивается на 1. Этот способ адресации используют команды LD (пересылка байта данных из памяти в регистр Rd) и ST (пересылка байта данных из регистра Rr в память), всего шесть команд – по две для каждого регистра. Применяется для эффективного доступа к элементам массива и к переменным указателя, когда нужен инкремент указателя после доступа.

Относительная адресация памяти программ. При этом способе адрес вычисляют путем сложения содержимого программного счетчика PC и константы к задаваемой в команде. Относительную адресацию используют команды относительного перехода (RJMP) и относительного вызова подпрограммы (RCALL), многочисленная группа команд условных переходов.

Структура программы на ассемблере

AVR-ассемблер не различает регистр букв.

Обязательным полем в строке является команда или директива.

Любая строка может начинаться с метки, которая является набором символов, заканчивающимся двоеточием. Метки используются для указания места, в которое передается управление при переходах, а также для задания имен переменных (позволяет перейти на эту строку из другого места программы). Метка не может начинаться с числа и совпадать с названием команды или регистра. Метка всегда заканчивается символом двоеточие (:). Метка может не находиться в одной строке с командой.

Комментарии отделяются от исполняемой строки символом «;» или символом «\\». Текст после точки с запятой (;) и до конца строки игнорируется компилятором.

Например label: .EQU var1=100 ; Устанавливает var1 равным 100.

Написание грамотных комментариев позволяет упростить понимание кода на ассемблере для самого автора программы (через какое-то время) и для других людей, работающих с этим кодом. Комментарий должен описывать, что было сделано и для чего это было сделано.

Входная строка может иметь одну из следующих форм:

метка: директива операнды ; комментарий

метка: инструкция операнды ; комментарий

; комментарий

Форматы представления чисел

В некоторые команды можно включать числовые значения. Числа по умолчанию считаются десятичными. Существуют следующие форматы представления чисел:

– десятичный (принят по умолчанию): 10, 255;

– шестнадцатеричный (два варианта записи): как в языке Си (0xFF3) или как в языке Pascal ($FF3). Не допускается форма записи типа FF3h;

– двоичный: 0b00001010, 0b11111111;

– восьмеричный (начинаются с нуля): 010, 077.

Выражения

При записи команд на ассемблере могут использоваться выражения, по которым в процессе ассемблирования программы вычисляются значения. Операндами выражений могут быть:

– числа (десятичные, шестнадцатеричные и двоичные);

– метки;

– коды символов ASCII ('A') и строки ASCII;

– символические имена, представляющие переменные, определенные директивой .SET, и константы, определенные директивой .EQU;

– текущее значение счетчика команд (PC).

Помимо операндов в выражения могут входить функции, например:

– Low (выражение) – возвращает младший байт выражения;

– High (выражение) – возвращает старший байт выражения;

– Exp2 (N) – возвращает 2^N;

– Log2 (N) – возвращает целую часть $\log_2 N$.

При записи выражений можно использовать операции отношения.

Операторы

Компилятор поддерживает ряд операторов, которые перечислены в таблице 13 (чем выше положение в таблице, тем выше приоритет оператора). Выражения могут заключаться в круглые скобки, такие выражения вычисляются перед выражениями за скобками.

Таблица 13 – Операторы AVR-ассемблера

Приоритет	Символ	Описание
14	!	Логическое отрицание
14	~	Побитное отрицание
14	-	Минус
13	*	Умножение
13	/	Деление
12	+	Суммирование
12	-	Вычитание
11	<<	Сдвиг влево
11	>>	Сдвиг вправо
10	<	Меньше чем
10	<=	Меньше или равно
10	>	Больше чем
10	>=	Больше или равно
9	==	Равно
9	!=	Не равно
8	&	Побитное И
7	^	Побитное исключающее ИЛИ
6	|	Побитное ИЛИ
5	&&	Логическое И
4	||	Логическое ИЛИ

Пример использования оператора побитного отрицания (~):

```
ldi r16, ~0xf0 ; Загрузить в регистр r16 число 0x0f.
```

Пример использования оператора суммирования (+):

```
ldi r30, c1+c2 ; Загрузить в регистр r30 сумму переменных c1 и c2.
```

2.2 Команды Atmega 16

Система команд Atmega 16 содержит 131 инструкцию. Как и для большинства других МК, этот набор можно условно разделить на четыре основные группы:

– арифметико-логические, включая сдвиги (собственно вычисления);

– передачи данных между ячейками памяти МК;

– ветвлений (переходов) в программе (формируют структуру программы);

– работы с битами.

По формату обрабатываемых данных различают команды, оперирующие с байтами и битами. Отдельные команды работают с 2-байтными числами.

Ниже при описании команд использованы следующие обозначения:

Rd – регистр-приемник, место, куда сохраняется результат выполнения команды;

Rr – регистр-источник в двухоперандных командах. Его значение после выполнения команды не изменяется;

Rdl – пара регистров R24, R26, R28, R30. Для инструкций ADIW и SBIW;

P, b – разряд b (b = 0, ..., 7) порта P;

Rr (b) – разряд b (b = 0, ..., 7) регистра Rr;

X,Y,Z – регистры косвенной адресации;

n – номер бита в регистре (от 0 до 7);

s – номер разряда в регистре SREG (от 0 до 7);

PC – содержимое программного счетчика;

K – восьмиразрядная константа данных. Используется в операциях с регистрами общего назначения R16–R31;

k – шестнадцатиразрядная адресная константа;

q – 6-разрядное смещение при работе с памятью данных;

STACK – область памяти SRAM, адресуемая указателем стека SP;

C, Z, N, V, S, H, T, I – биты регистра состояния SREG;

d, r = 0...31 во всех случаях (кроме специально отмеченных);

$\overline{\ }$ – инверсия;

• – логическое И;

∨ – логическое ИЛИ;

⊕ – исключающее ИЛИ.

Команды передачи данных. К этой группе относятся команды, которые переносят данные из одной области памяти в другую (включая регистры). Команды этой группы приведены в таблице 14.

Таблица 14 – Команды передачи данных

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Циклы
MOV	Rd, Rr	Перемещение между регистрами	Move Between Registers	Rd ←Rr	-	1
MOVW	Rd, Rr	Скопировать пару регистров	Copy Register Word	Rd+1:Rd ← Rr+1:R	-	1
LDI	Rd, K	Загрузить непосредственно значение	Load Immediate	Rd ← K	-	1
LD	Rd, X	Загрузить косвенно	Load Indirect	Rd ← (X)	-	2
LD	Rd, X+	Загрузить косвенно, инкрементировав впоследствии	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	-	2

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Циклы
LD	Rd, Y	Загрузить косвенно	Load Indirect	Rd ← (Y)	-	2
LD	Rd, Y+	Загрузить косвенно, инкрементировав впоследствии	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	-	2
LD	Rd, -Y	Загрузить косвенно, декрементировав предварительно	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	-	2
LDD	Rd, Y+q	Загрузить косвенно со смещением	Load Indirect with Displacement	Rd ← (Y + q)	-	2
LD	Rd, Z	Загрузить косвенно	Load Indirect	Rd ← (Z)	-	2
LD	Rd, Z+	Загрузить косвенно, инкрементировав впоследствии	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	-	2
LD	Rd, -Z	Загрузить косвенно, декрементировав предварительно	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ←(Z)	-	2
LDD	Rd, Z+q	Загрузить косвенно со смещением	Load Indirect with Displacement	Rd ← (Z + q)	-	2
LDS	Rd, k	Загрузить непосредственно из СОЗУ	Load Direct from SRAM	Rd ← (k)	-	2
ST	X, Rr	Записать косвенно	Store Indirect	(X) ← Rr	-	2
ST	X+, Rr	Записать косвенно, инкрементировав впоследствии	Store Indirect and Post-Inc.	(X)←Rr, X ←X + 1	-	2
ST	- X, Rr	Записать косвенно, декрементировав предварительно	Store Indirect and Pre-Dec.	X ←X - 1, (X) ←Rr	-	2
ST	Y, Rr	Записать косвенно из регистра в СОЗУ с использованием индекса Y	Store Indirect	(Y) ←Rr	-	2
ST	Y+, Rr	Записать косвенно, инкрементировав впоследствии	Store Indirect and Post-Inc.	(Y) ←Rr, Y ←Y + 1	-	2

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Циклы
ST	-Y, Rr	Записать косвенно, декрементировав предварительно	Store Indirect and Pre-Dec.	Y ←Y - 1, (Y) ←Rr	-	2
STD	Y+q, Rr	Записать косвенно со смещением	Store Indirect with Displacement	(Y + q) ←Rr	-	2
ST	Z, Rr	Записать косвенно из регистра в СОЗУ с использованием индекса Z	Store Indirect	(Z) ←Rr	-	2
ST	Z+, Rr	Записать косвенно, инкрементировав впоследствии	Store Indirect and Post-Inc.	(Z) ←Rr, Z ←Z + 1	-	2
ST	-Z, Rr	Записать косвенно, декрементировав предварительно	Store Indirect and Pre-Dec.	Z ←Z - 1, (Z) ←Rr	-	2
STD	Z+q, Rr	Записать косвенно со смещением	Store Indirect with Displacement	(Z + q) ←Rr	-	2
STS	k, Rr	Загрузить непосредственно в СОЗУ	Store Direct to SRAM	(k) ←Rr	-	2
LPM	-	Загрузить байт памяти программ	Load Program Memory	R0 ←(Z)	-	3
LPM	Rd, Z	Загрузить байт памяти программ	Load Program Memory	Rd ←(Z)	-	3
LPM	Rd, Z+	Загрузить байт памяти программ, инкрементировав впоследствии	Load Program Memory and Post-Inc.	Rd ←(Z), Z ←Z+1	-	3
SPM	-	Записать байт памяти программ	Store Program Memory	(Z) ←R1:R0	-	-
IN	Rd, P	Загрузить данные из порта I/O в регистр	In Port	Rd ← P	-	1
OUT	P, Rr	Записать данные из регистра в порт I/O	Out Port	P ←Rr	-	1
PUSH	Rr	Поместить регистр в стек	Push Register on Stack	STACK ←Rr	-	2
POP	Rd	Загрузить регистр из стека	Pop Register from Stack	Rd ←STACK	-	2

Команды арифметико-логических операций. К данной группе относятся операции сложения, вычитания и умножения. Операндами в командах данной группы могут быть только регистры общего назначения. Результат операции (кроме умножения) записывается по адресу первого операнда.

Значительная часть функций процессора осуществляется через логические операции с регистрами. Логические операции применимы только к РОН. В этой группе представлены стандартные логические операции: побитовое and (И), or (ИЛИ) и eor (исключающее ИЛИ), а также перевод в обратный код (com) и в дополнительный код (neg).

Составление программ в терминах комбинационной логики для МК не характерно, наиболее часто команды логических операций выполняют маскирование отдельных битов или их групп: так, операция andi temp,0b00001111 позволит оставить младшую тетраду переменной temp без изменений, а старшую – обнулить. Наоборот, команда ori temp,0b00001111 позволит оставить старшую тетраду без изменений, а в младшей все биты установить в единичное состояние. Операцию eor (исключающее ИЛИ) можно назвать элементом несовпадения – она позволяет зафиксировать те биты, которые совпадают (или не совпадают) в обоих операндах (совпадающие установятся в нули). Группа арифметико-логических операций представлена в таблице 15.

Таблица 15 – Группа арифметико-логических операций

Код команды	Операнды	Описание	Англоязычная версия	Операция	Флаги	Цик-лы
ADD	Rd, Rr	Сложение двух регистров	Add two Registers	Rd←Rd+Rr	Z,C,N,V,H,S	1
ADC	Rd, Rr	Сложение двух регистров с переносом	Add with Carry two Registers	Rd←Rd+Rr+C	Z,C,N,V,H,S	1
ADIW	Rdl, K	Сложение константы со словом	Add Immediate to Word	Rdh:Rdl← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Вычитание одного регистра из другого	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Вычитание константы из регистра	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Вычитание одного регистра из другого с переносом	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Вычитание константы из регистра с переносом	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1

Код команды	Операнды	Описание	Англоязычная версия	Операция	Флаги	Цик-лы
SBIW	Rdl, K	Вычитание непосредственно из слова	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Логическое И двух регистров	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Логическое И регистра и константы	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Логическое ИЛИ двух регистров	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Логическое ИЛИ регистра и константы	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Исключающее ИЛИ двух регистров	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	Одиночное дополнение	One’s Complement	Rd ←$FF - Rd	Z,C,N,V	1
NEG	Rd	Двойное дополнение	Two’s Complement	Rd ← $00− Rd	Z,C,N,V	1
SBR	Rd, K	Задать бит(ы) в регистре	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd, K	Удалить бит(ы) в регистре	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Инкремент	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Декремент	Decrement	Rd ← Rd - 1	Z,N,V	1
TST	Rd	Проверка нуля или отрицательного значения	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Очистить регистр	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Задать регистр	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Беззнаковое умножение	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	1
MULS	Rd, Rr	Знаковое умножение	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	1
MULSU	Rd, Rr	Знаковое умножение с беззнаковым	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	1
FMUL	Rd, Rr	Дробное беззнаковое умножение	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	1

Код команды	Операнды	Описание	Англоязычная версия	Операция	Флаги	Цик-лы
FMULS	Rd, Rr	Дробное знаковое умножение	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	1
FMULSU	Rd, Rr	Дробное знаковое умножение с беззнаковым	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	1

Команды ветвления (передачи управления)

Самая большая группа команд передачи управления начинается с букв br, (branch – ветка). Это команды условного перехода, которые считаются одними из самых главных в любой системе программирования, поскольку позволяют организовывать циклы. Эти команды по смыслу соответствуют конструкциям «if ... then … else». Наиболее часто употребляется пара brne (Branch if Not Equal – перейти, если не равно) и breq (Branch if Equal – перейти, если равно). В случае выполнения условия осуществляется переход по некоторому адресу, в случае невыполнения – по следующему адресу. Следовательно, команды ветвления обязательно употребляются в паре с одной из команд, устанавливающих флаг (в вышеуказанном примере – флаг нуля z в регистре состояния SREG). Для всех флагов регистра SREG есть особые пары команд, устанавливающие или сбрасывающие их.

Команды проверки пропуска

Это распространенная группа команд, которые осуществляют пропуск следующей по порядку команды в зависимости от состояния отдельного бита в РОН или РВВ. Основные из них – пары команд sbrs/sbrc (для РОН) и sbis/sbic (для РВВ). Они удобны для организации процедур, аналогичных оператору выбора CASE в языках высокого уровня. Обладают логикой «пропустить следующую команду, если условие выполняется». Команды ветвления приведены в таблице 16.

Таблица 16 – Группа команд ветвления

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Цик-лы
RJMP	k	Относительный переход	Relative Jump	PC ← PC + k + 1	-	2
IJMP	-	Косвенно перейти из Z	Indirect Jump to (Z)	PC ← Z	-	2
JMP	k	Перейти прямо	Direct Jump	PC ← k	-	3
RCALL	k	Относительный вызов подпрограммы	Relative Subroutine Call	PC ← PC + k+1	-	3
ICALL	-	Косвенный вызов из Z	Indirect Call to (Z)	PC ← Z	-	3

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Цик-лы
CALL	k	Прямой вызов подпрограммы	Direct Subroutine Call	PC ← k	-	4
RET	-	Возврат подпрограммы	Subroutine Return	PC←STACK	-	4
RETI	-	Прерывание возврата	Interrupt Return	PC←STACK	I	4
CPSE	Rd, Rr	Сравнить, пропустить, если равно	Compare, Skip if Equal	if (Rd = Rr) PC← PC + 2 or 3	-	1/2/3
CP	Rd, Rr	Сравнить	Compare	Rd −Rr	Z,N,V, C,H	1
CPC	Rd, Rr	Сравнить с переносом	Compare with Carry	Rd −Rr −C	Z,N,V, C,H	1
CPI	Rd, K	Сравнить регистр непосредственно	Compare Register with Immediate	Rd - K	Z,N,V, C,H	1
SBRC	Rr, b	Пропустить, если бит в регистре очищен	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ←PC + 2 or 3	-	1/2/3
SBRS	Rr, b	Пропустить, если бит в регистре установлен	Skip if Bit in Register is Set	if (Rr(b)=1) PC ←PC + 2 or 3	-	1/2/3
SBIC	P, b	Пропустить, если бит во входном/выходном регистре очищен	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ←PC + 2 or 3	-	1/2/3
SBIS	P, b	Пропустить, если бит во входном/выходном регистре установлен	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ←PC + 2 or 3	-	1/2/3
BRBS	s, k	Переход, если состояние флага – установлен	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	-	1/2
BRBC	s, k	Переход, если состояние флага – снят	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k+1	-	1/2
BREQ	K	Переход, если равно	Branch if Equal	if (Z = 1) then PC←PC+k+1	-	1/2
BRNE	K	Переход, если не равно	Branch if Not Equal	if (Z = 0) then PC←PC+k+1	-	1/2
BRCS	K	Переход, если перенос установлен	Branch if Carry Set	if (C = 1) then PC←PC+k+1	-	1/2

Мнемоника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Цик-лы
BRCC	K	Переход, если перенос снят	Branch if Carry Cleared	if (C = 0) then PC ← PC+k+1	-	1/2
BRSH	K	Переход, если одинаковый или высокий	Branch if Same or Higher	if (C = 0) then PC ← PC+k+1	-	1/2
BRLO	K	Переход, если низкий	Branch if Lower	if (C = 1) then PC ← PC+k+1	-	1/2
BRMI	K	Переход, если минус	Branch if Minus	if (N = 1) then PC ← PC+k+1	-	1/2
BRPL	K	Переход, если плюс	Branch if Plus	if (N = 0) then PC ← PC+k+1	-	1/2
BRGE	K	Переход, если больше или равен, знаковый	Branch if Greater or Equal, Signed	if (N ⊕V= 0) then PC← PC+k + 1	-	1/2
BRLT	K	Переход, если меньше нуля, знаковый	Branch if Less Than Zero, Signed	if (N ⊕V= 1) then PC ← PC+k+1	-	1/2
BRHS	K	Переход, если установлен флаг полупереноса	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC+k+1	-	1/2
BRHC	K	Переход, если флаг полупереноса снят	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC+k+1	-	1/2
BRTS	K	Переход, если установлен флаг T	Branch if T Flag Set	if (T = 1) then PC ← PC+k+1	-	1/2
BRTC	K	Переход, если снят флаг T	Branch if T Flag Cleared	if (T = 0) then PC ← PC+k+1	-	1/2
BRVS	K	Переход, если флаг переполнения установлен	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC+k+1	-	1/2
BRVC	K	Переход, если флаг переполнения снят	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC+k+1	-	1/2
BRIE	K	Переход, если прерывание включено	Branch if Interrupt Enabled	if (I = 1) then PC ← PC+k+1	-	1/2
BRID	K	Переход, если прерывание выключено	Branch if Interrupt Disabled	if (I = 0) then PC ← PC+k+1	-	1/2

Группа битовых и бит-проверочных команд. Команды этой группы представлены в таблице 17.

Таблица 17 – Группа битовых и бит-проверочных команд

Мнемо-ника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Циклы
SBI	P, b	Установить бит в регистр I/O	Set Bit in I/O Register	I/O(P,b) ← 1	-	2
CBI	P, b	Очистить бит в регистре I/O	Clear Bit in I/O Register	I/O(P,b) ← 0	-	2
LSL	Rd	Логически сдвинуть влево	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Логически сдвинуть вправо	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Сдвинуть влево через перенос	Rotate Left Through Carry	Rd(0)←C,Rd(n+1) ←Rd(n) C←Rd(7)	Z,C,N,V	1
ROR	Rd	Сдвинуть вправо через перенос	Rotate Right Through Carry	Rd(7)←C,Rd(n) ←Rd(n+1) C←Rd(0)	Z,C,N,V	1
ASR	Rd	Арифметически сдвинуть вправо	Arithmetic Shift Right	Rd(n) ←Rd(n+1) n=0..6	Z,C,N,V	1
SWAP	Rd	Перестановка тетрад	Swap Nibbles	Rd(3..0)←Rd(7..4), Rd(7..4)←Rd(3..0)	-	1
BSET	-	Установить флаг	Flag Set	SREG(s) ←1	SREG(s)	1
BCLR	-	Очистить флаг	Flag Clear	SREG(s) ←0	SREG(s)	1
BST	-	Переписать бит из регистра во флаг T	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	-	Загрузить флаг T в бит регистра	Bit load from T to Register	Rd(b) ← T	-	1
SEC	-	Установить флаг переноса	Set Carry	C ← 1	C	1
CLC	-	Очистить флаг переноса	Clear Carry	C ← 0	C	1
SEN	-	Установить флаг отрицательного значения	Set Negative Flag	N ← 1	N	1
CLN	-	Очистить отрицательный флаг	Clear Negative Flag	N ←0	N	1
SEZ	-	Установить флаг нуля	Set Zero Flag	Z ←1	Z	1

Мнемо-ника	Операнды	Описание	Англоязычная версия	Содержание операции	Флаги	Циклы
CLZ	-	Очистить флаг нуля	Clear Zero Flag	Z ←0	Z	1
SEI	-	Включить глобальное прерывание	Global Interrupt Enable	I ←1	I	1
CLI	-	Отключить глобальное прерывание	Global Interrupt Disable	I←0	I	1
SES	-	Установить флаг знака	Set Signed Test Flag	S ←1	S	1
CLS	-	Очистить флаг знака	Clear Signed Test Flag	S ←0	S	1
SEV	-	Установить флаг переполнения	Set Twos Complement Overflow	V ←1	V	1
CLV	-	Очистить флаг переполнения	Clear Twos Complement Overflow	V ←0	V	1
SET	-	Установить флаг T	Set T in SREG	T ←1	T	1
CLT	-	Очистить флаг T	Clear T in SREG	T ←0	T	1
SEH	-	Установить флаг полупереноса	Set Half Carry Flag in SREG	H ←1	H	1
CLH	-	Очистить флаг полупереноса	Clear Half Carry Flag in SREG	H ←0	H	1

Команды управления микроконтроллером

Операция NOP входит в набор команд всех ассемблеров и служит для заполнения ячеек памяти программ пустыми значениями, если это требуется. Команды этой группы представлены в таблице 18.

Таблица 18 – Группа команд управления микроконтроллером

Мнемоника	Операнды	Описание	Англоязычная версия	Флаги	Циклы
NOP	-	Выполнить пустую команду	No Operation	-	1
SLEEP	-	Установить режим SLEEP	Sleep	-	1
WDR	-	Сбросить сторожевой таймер	Watchdog Reset	-	1
BREAK	-	Прервать операцию	Break	-	нет

2.3 Директивы ассемблера

Кроме команд в ассемблерной программе могут встречаться директивы компилятора. Компилятор поддерживает ряд директив.

Директива – это предписание компилятору языка выполнить то или иное действие в момент компиляции. Директивы не транслируются непосредственно в код. Перед названием директивы ставится точка. Самыми употребительными являются директивы def, equ, include. Список директив приведен в таблице 19.

Таблица 19 – Директивы ассемблера

Директива	Описание
.BYTE	Зарезервировать байты в ОЗУ
.CSEG	Программный сегмент
.DB	Определить байты во flash или EEPROM
.DEF	Назначить регистру символическое имя
.DEVICE	Определить устройство, для которого компилируется программа
.DSEG	Сегмент данных
.DW	Определить слова во flash или EEPROM
.ENDMACRO	Конец макроса
.EQU	Установить постоянное выражение
.ESEG	Сегмент EEPROM
.EXIT	Выйти из файла
.INCLUDE	Вложить другой файл
.LIST	Включить генерацию листинга
.LISTMAC	Включить разворачивание макросов в листинге
.MACRO	Начало макроса
.NOLIST	Выключить генерацию листинга
.ORG	Установить положение в сегменте
.SET	Установить переменный символический эквивалент выражения

.BYTE – зарезервировать байты в ОЗУ

Директива BYTE резервирует байты в ОЗУ. Если есть необходимость ссылаться на выделенную область памяти, то директива BYTE должна быть предварена меткой. Директива принимает один параметр, который указывает количество выделяемых байтов. Эта директива может использоваться только в сегменте данных. Выделенные байты не инициализируются.

Синтаксис: МЕТКА: .BYTE выражение

Пример:

```
.DSEG
var1:   .BYTE 1              ; резервирует 1 байт для var1
table:  .BYTE tab_size       ; резервирует tab_size байт
        .CSEG
        ldi r30,low(table)   ; загружает младший байт регистра Z
        ldi r31,high(table)  ; загружает старший байт регистра Z
        ld r1,Z              ; загружает VAR1 в регистр 1
```

.CSEG – программный сегмент

Директива CSEG определяет начало программного сегмента. Исходный файл может состоять из нескольких программных сегментов, которые объединяются в один программный сегмент при компиляции. Программные сегменты имеют свои собственные счетчики положения, которые считают не побайтно, а пословно. Директива ORG может быть использована для

размещения кода и констант в необходимом месте сегмента. Директива CSEG не имеет параметров.

Синтаксис: .CSEG

```
Пример:     .DSEG              ; Начало сегмента данных
vartab:     .BYTE 4            ; Резервирует 4 байта в ОЗУ
            .CSEG              ; Начало кодового сегмента
const:      .DW 2              ; Разместить константу 0x0002 в памяти программ
            mov r1, r0         ; Выполнить действия
```

.DB – определить байты во flash или EEPROM

Директива DB резервирует необходимое количество байтов в памяти программ или в EEPROM. Если необходимо иметь возможность ссылаться на выделенную область памяти, то директива DB должна быть предварена меткой. Директива DB должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG). Параметры, передаваемые директиве, – это последовательность выражений, разделенных запятыми. Каждое выражение должно быть или числом в диапазоне (минус 128 – 255), или в результате вычисления должно давать результат в этом же диапазоне, в противном случае число усекается до байта. Если директива получает более одного параметра и текущим является программный сегмент, то параметры упаковываются в слова (первый параметр – младший байт), и если число параметров нечетно, то последнее выражение будет усечено до байта и записано как слово со старшим байтом, равным нулю, даже если далее идет еще одна директива DB.

Синтаксис: МЕТКА: .DB список выражений

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

.DEF – назначить регистру символическое имя

Директива DEF позволяет ссылаться на регистр через некоторое символическое имя. Назначенное имя может использоваться во всей нижеследующей части программы для обращений к данному регистру. Регистр может иметь несколько различных имен. Символическое имя может быть переназначено позднее в программе.

Синтаксис: .DEF Символическое_имя = Регистр

```
Пример:     .DEF temp=R16
            .DEF ior=R0
            .CSEG
            ldi temp, 0xf0          ; Загрузить 0xf0 в регистр temp (R16)
            In ior, 0x3f            ; Прочитать SREG в регистр ior (R0)
            eor temp, ior           ; Регистры temp и ior складываются
                                    ; по исключающему ИЛИ
```

.DSEG – сегмент данных

Директива DSEG определяет начало сегмента данных. Исходный файл может состоять из нескольких сегментов данных, которые объединяются в один сегмент

при компиляции. Сегмент данных обычно состоит только из директив BYTE и меток. Сегменты данных имеют свои собственные побайтные счетчики положения. Директива ORG может быть использована для размещения переменных в необходимом месте ОЗУ. Директива не имеет параметров. Пример использования директивы DSEG смотрите в примере к директиве .BYTE

.DW – определить слова во flash или EEPROM

Директива DW резервирует необходимое количество слов в памяти программ или в EEPROM. Если вы хотите иметь возможность ссылаться на выделенную область памяти, то директива DW должна быть предварена меткой. Директива DW должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG). Параметры, передаваемые директиве, – это последовательность выражений, разделенных запятыми. Каждое выражение должно быть числом в диапазоне (минус 32 768 – 65 535).

Синтаксис: МЕТКА: .DW expressionlist

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

.EQU – присвоить символьное имя

Директива EQU присваивает метке значение. Эта метка может позднее использоваться в выражениях. Метка, которой присвоено значение данной директивой, не может быть переназначена, и ее значение не может быть изменено.

Синтаксис: .EQU метка = выражение

Пример:
```
.EQU io_offset = 0x23
.EQU portA = io_offset + 2
.CSEG              ; Начало сегмента данных
clr r2             ; Очистить регистр r2
out portA, r2      ; Записать в порт A
```

.ESEG – сегмент EEPROM

Директива ESEG определяет начало сегмента EEPROM. Исходный файл может состоять из нескольких сегментов EEPROM, которые объединяются в один сегмент при компиляции. Сегмент EEPROM обычно состоит только из директив DB, DW и меток. Сегменты EEPROM имеют свои собственные побайтные счетчики положения. Директива ORG может быть использована для размещения переменных в необходимом месте EEPROM. Директива не имеет параметров.

Синтаксис: .ESEG

.INCLUDE – вложить другой файл

Встретив директиву INCLUDE, компилятор открывает указанный в ней файл, компилирует его, пока файл не закончится или не встретится директива EXIT, после этого продолжает компиляцию начального файла со строки, следующей за директивой INCLUDE.

Синтаксис: .INCLUDE «имя_файла»

.MACRO – начало макроса

С директивы MACRO начинается определение макроса. В качестве параметра директиве передается имя макроса. При встрече имени макроса позднее в тексте программы компилятор заменяет это имя на тело макроса. Макрос может иметь до 10 параметров, к которым в его теле обращаются через @0 – @9. При вызове параметры перечисляются через запятые. Определение макроса заканчивается директивой ENDMACRO.

Синтаксис: .MACRO макроимя

```
Пример:   .MACRO out1        ; Начало макроопределения
          ldi r16, @1        ; Установка значения в регистр
          out @0, R16        ; Вывод данных в порт или регистр порта
          .ENDMACRO          ; Конец макроопределения
```

.ORG – установить положение в сегменте

Директива устанавливает счетчик положения равным заданной величине, которая передается как параметр. Для сегмента данных она устанавливает счетчик положения в SRAM (ОЗУ), для сегмента программ это программный счетчик, а для сегмента EEPROM это положение в EEPROM. Если директиве предшествует метка, то метка размещается по адресу, указанному в параметре директивы. Перед началом компиляции программный счетчик и счетчик EEPROM равны нулю, а счетчик ОЗУ равен 32 (адреса 0 – 31 заняты регистрами). Для ОЗУ и EEPROM используются побайтные счетчики, а для программного сегмента – пословные.

Синтаксис: .ORG выражение

```
Пример:   DSEG               ; Начало сегмента данных
          .ORG 0x37          ; Установить адрес SRAM равным 0x37
variable: .BYTE 1            ; Зарезервировать байт по адресу 0x37H
          .CSEG
          .ORG 0x10          ; Установить программный
                             ; счетчик равным 0x10
          mov r0,r1          ; Команда будет размещена по адресу 0x10
```

.SET – установить переменный символический эквивалент выражения

Директива SET присваивает имени некоторое значение. Это имя позднее может быть использовано в выражениях. Причем в отличие от директивы EQU значение имени может быть изменено другой директивой SET.

Синтаксис: .SET имя = выражение

```
Пример:   .SET io_offset = 0x23
          .SET portA = io_offset + 2
          .CSEG                    ; Начало кодового сегмента
          clr r2                   ; Очистить регистр 2
          out portA, r2            ; Записать в порт A
          .SET portA = 0x20
          In r2, portA
```

3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЛАБОРАТОРНОГО ПРАКТИКУМА

3.1 Интегрированная среда разработки Atmel Studio

Для AVR созданы разные интегрированные среды разработки (IDE – Integrated Development Environment), т. е. системы программных средств, в состав которых входят:

– текстовый редактор;
– компилятор и/или интерпретатор;
– средства автоматизации сборки;
– отладчик.

Наиболее распространенные из них AVR Studio и Atmel Studio. Atmel Studio – бесплатная среда разработки от компании Atmel. Среда предназначена для разработки и отладки приложений на языке C/C++ и языке ассемблера для 8- и 32-битных микроконтроллеров Atmel AVR и микроконтроллеров линейки Cortex-M.

Учитывая широчайшие возможности Atmel Studio, в данное учебно-методическое пособие включено описание минимального набора функций, необходимых для выполнения лабораторных работ. Более подробно с возможностями Atmel Studio можно ознакомиться на сайте Atmel в разделе документации.

Чтобы создать новый проект, нужно выбрать пункт «New Project…». После чего откроется новое окно, где можно выбрать язык программирования и тип компиляции, название проекта, его месторасположение. Чтобы создать проект на ассемблере, нужно выбрать пункт Assembler и задать название проекта (рисунок 12). Для того чтобы загрузить существующий проект, нужно выбрать пункт «Open Project…».

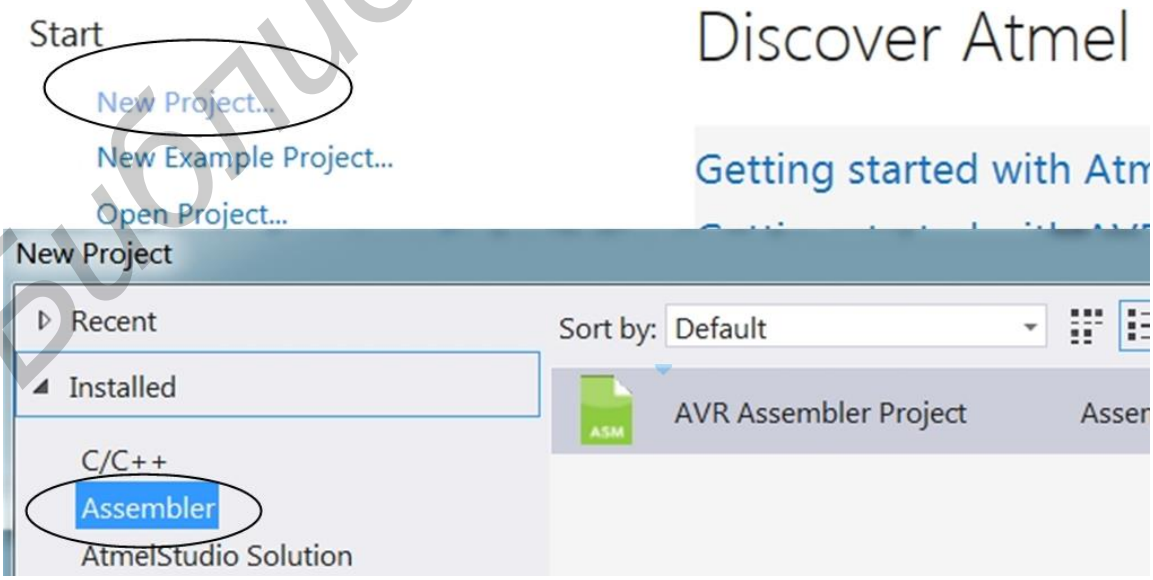

Рисунок 12 – Создание нового проекта на ассемблере

Далее в отрывшемся окне Device Selection выбираем семейство megaAVR, 8-bit и модель Atmega 16 (рисунок 13). В правой части окна приведен список устройств, работающих с этим микроконтроллером, а также документация.

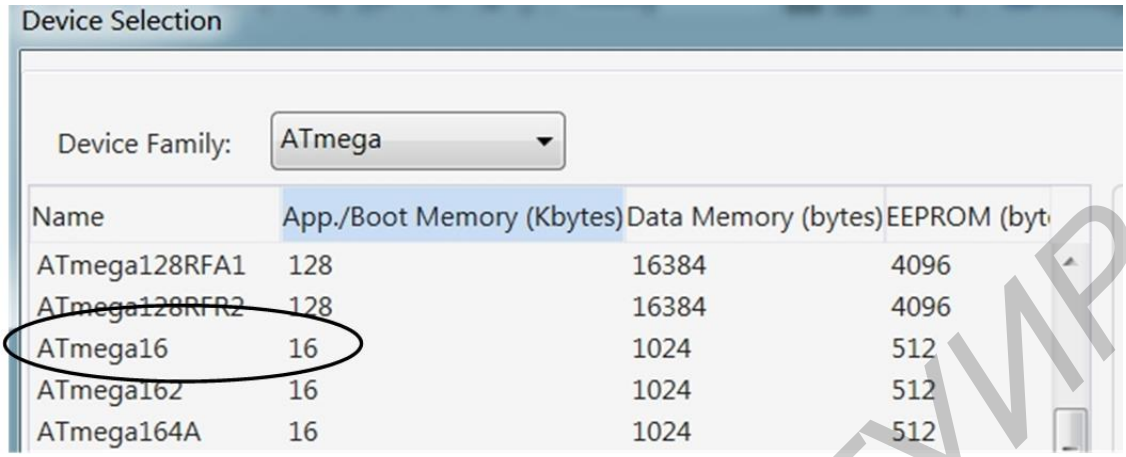

Рисунок 13 – Окно выбора микроконтроллера Device Selection

После подтверждения выбора микроконтроллера появляется основное окно Atmel Studio (рисунок 14), в котором производится редактирование и отладка программы. На начальном этапе шаблон кода содержит только время, дату создания и название файла проекта, а также имя пользователя.

Для включения нумерации строк кода необходимо выставить флажок на опции Line numbers в разделе Tools -> Options -> Text Editor -> All Languages -> General.

Выставить подходящий размер шрифта – Tools -> Options -> Environment -> Font and Colors -> Font: Consolas, Size:.

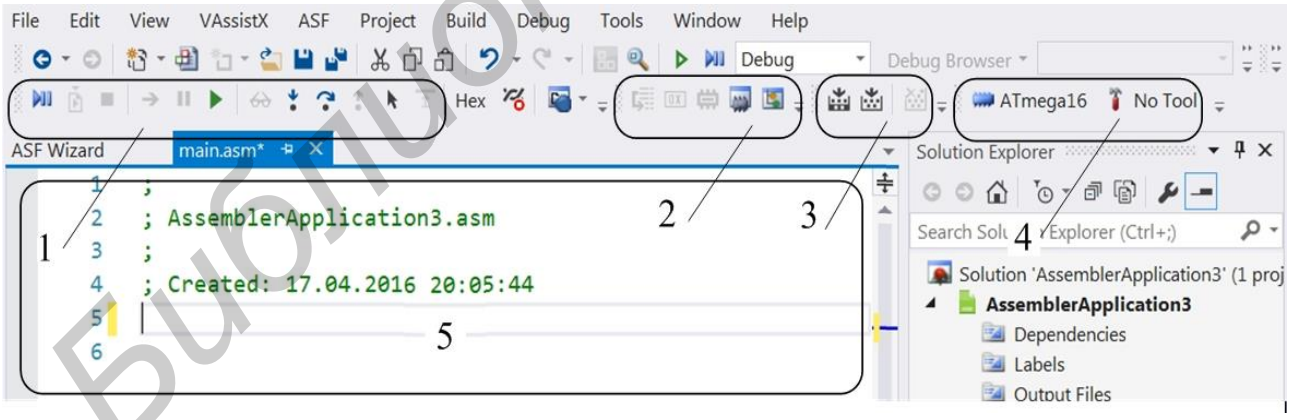

Рисунок 14 – Основное окно Atmel Studio

Можно выделить пять областей, активно используемых в процессе отладки.

Область 1 – Debug Toolbar Options содержит некоторые инструменты отладки (набор инструментов может быть настроен пользователем).

Область 2 – Atmel Debugger Toolbar Options содержит инструменты для просмотра содержимого памяти, регистров и портов микроконтроллера.

Область 3 – Build Toolbar Options содержит инструменты для создания (сборки) проекта.

Область 4 – Device and Debugger Toolbar Options позволяет выбрать целевой микроконтроллер и инструмент для отладки и прошивки.

Область 5 – основное окно, область для набора кода программы.

Отладку проекта необходимо начать с выбора инструмента для отладки (вкладка Device and Debugger Toolbar Options). По умолчанию выставлено значение NoTool. Щелчок по этому значку вызывает появление меню, в котором необходимо выбрать Selected debugger/programmer – Simulator (рисунок 15). Таким образом, для отладки будет использован программный эмулятор отладки, встронный в Atmel Studio.

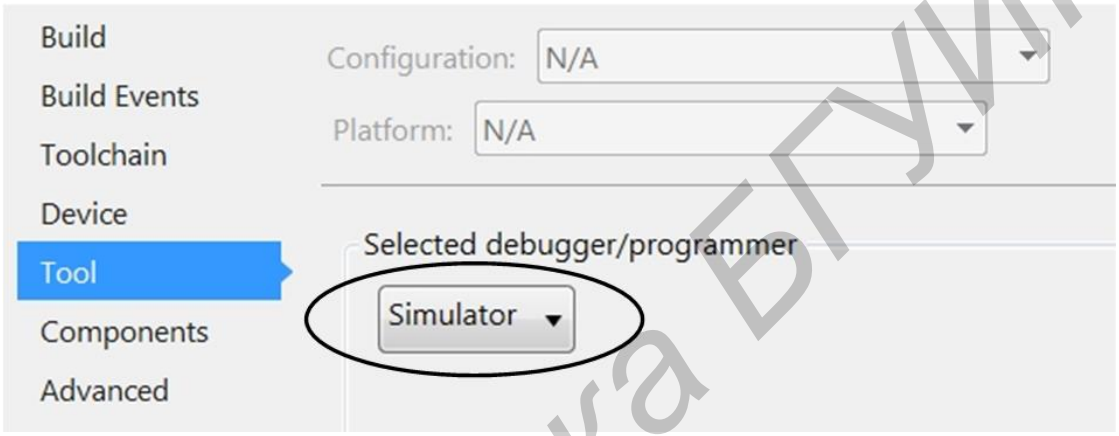

Рисунок 15 – Выбор отладчика

После того так устройство отладки было выбрано, становится возможным начать процедуру отладки. Для запуска отладки надо выбрать пункт меню Start Debugging and Break или использовать сочетание клавиш Alt+F5. Вкладка Debug содержит инструменты отладки (рисунок 16).

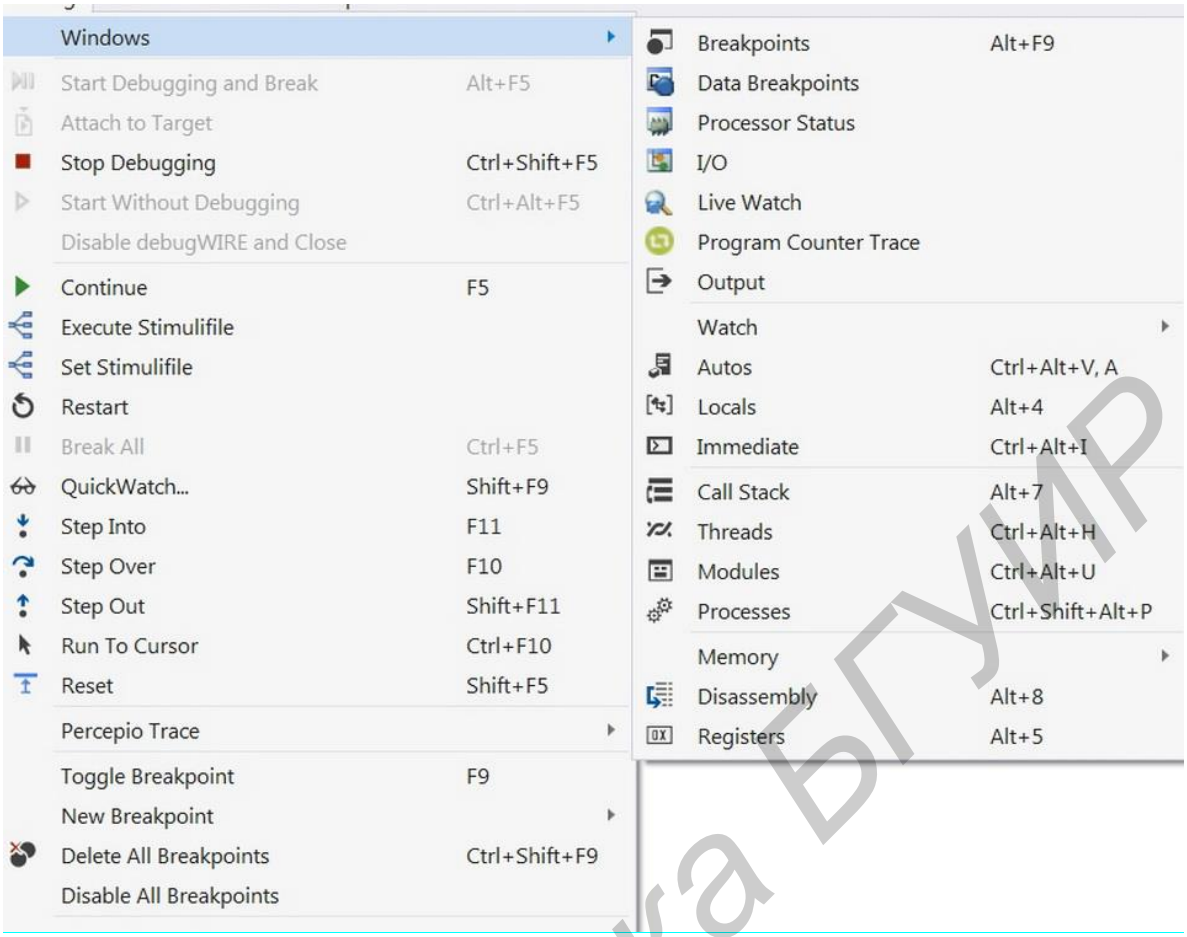

Рисунок 16 – Вкладка Debug

Start Debugging and Break – запуск отладки, начиная с первой команды в программе.

Start Without Debugging – создает проект без запуска отладки.

Break All – приостановка отладчика.

Stop Debugging – остановка и закрытие отладки. Возвращение в режим разработки.

Restart – перезапуск отладчика и перегрузка программы.

Reset – сброс программы к первой команде (отладка продолжается с первой команды в программе).

Disable debugWire and Close – опция доступна только при использовании интерфейса debugWire.

Step Into (F11) – выполнение одной инструкции (команды).

Step Over – выполнение одной инструкции. В том случае, если инструкция содержит вызов подпрограммы или функцию, то вызываемая подпрограмма будет также выполнена. Если вызываемая подпрограмма содержит точку останова, выполнение будет приостановлено.

Step Out – продолжает выполнение до тех пор, пока текущая функция не будет выполнена. Команда Step Out полностью выполняет текущую функцию. Если в процессе выполнения встретится контрольная точка, то выполнение

будет остановлено. Если выполняемая функция является функцией высокого уровня в иерархии подпрограмм, то будут выполнены все нижележащие функции, причем выполнение будет происходить до ближайшей контрольной точки или до тех пор, пока не будет дана команда останова.

Quick Watch – добавляет окно Quick Watch. Для использования этой функции нужно выделить интересующую переменную в тексте программы и нажать Quick Watch. В открывшемся окне отобразится значение выбранной переменной. В дальнейшем при выполнении программы значение этой переменной возможно поменяется. Используя окно Quick Watch, можно быстро отследить новые значения интересующей переменной.

Toggle Breakpoint – включает или выключает точку останова в месте расположения курсора.

New Breakpoint – создает новую точку останова в месте расположения курсора.

Disable All Breakpoints – очищает все точки останова, включая неактивные.

Clear All DataTips – очищает все установленные маркеры DataTips. Маркеры DataTips являются средством, упрощающим процесс отладки. В процессе отладки можно выделить интересующую переменную или регистр в коде программы и, щелкнув правой кнопкой мыши, выбрать Pin To Source. После чего рядом с выделенной переменной появится небольшое окно, в котором будет прописано название переменной и ее текущее значение. В процессе выполнения программы в этом окне будет отображаться актуальное значение переменной. Маркеры DataTip исчезнут, если указатель мыши будет переведен на другую строку. Чтобы закрепить маркер, необходимо щелкнуть на значке Pin to source.

Export Data Tips… – сохраняет все маркеры Data Tips в Visual Studio Shell.

Import DataTips… – загружает маркеры Data Tips из Visual Studio Shell.

Options and Settings – настройки раздела Debug.

Окно **Processor View** показывает регистры микроконтроллера (рисунок 17).

Программный счетчик (Program Counter). Программный счетчик всегда указывает на следующий адрес команды относительно выполняющейся в данный момент. Содержимое программного счетчика отображается в шестнадцатеричной системе.

Указатель стека Stack Pointer содержит адрес вершины стека. Если в микроконтроллере присутствует аппаратный стек, то он отражается в области указателя стека. Содержимое указателя стека можно изменять в режиме останова программы.

Содержимое регистров **X,Y,Z** используется для косвенной адресации. Регистр **SREG** показывает текущее состояние флагов.

Частота контроллера Frequency по умолчанию выставлена на 1 МГц.

Счетчик циклов Cycle counter содержит число циклов, прошедших с начала эмуляции.

Содержимое регистров может отображаться в шестнадцатеричной, десятичной и двоичной (флаги) формах представления. Для того чтобы изменить форму представления, нужно щелкнуть правой кнопкой мыши на значении и выбрать Display as… . Значения регистров и флагов могут быть модифицированы вручную.

В разделе **Registers** (рисунок 18) отображается содержимое регистров общего назначения (РОН).

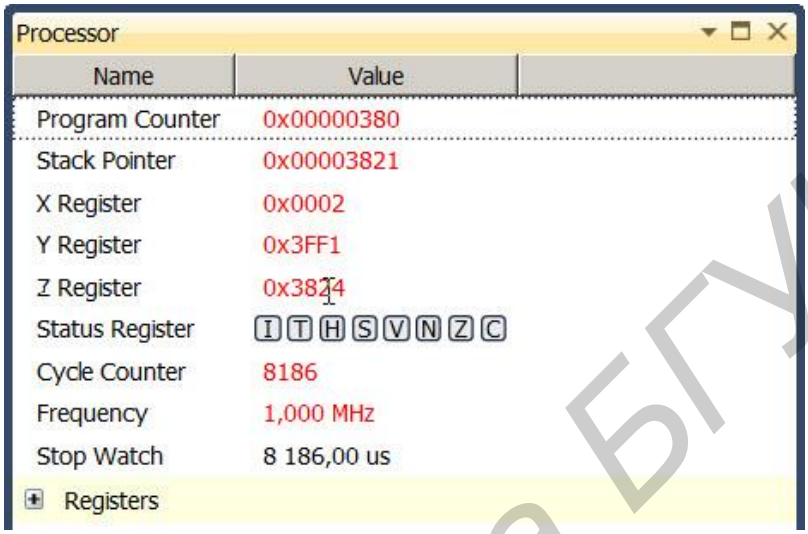

Рисунок 17– Окно Processor View

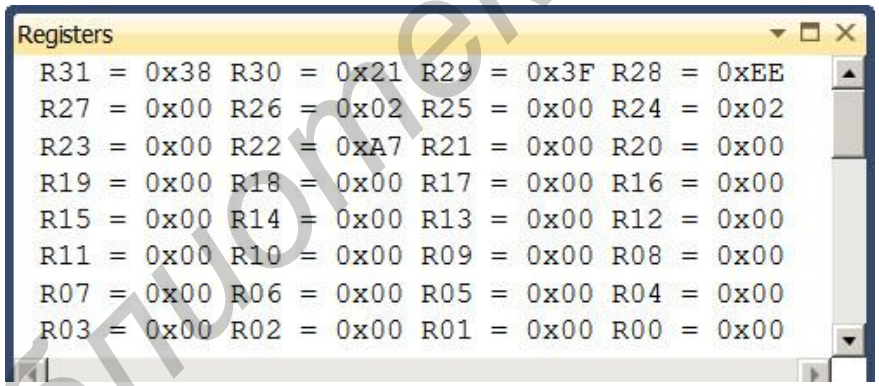

Рисунок 18 – Окно Registers

В окне **I/O View** отображается содержимое портов ввода/вывода (регистры ввода/вывода), счетчиков/таймеров, сторожевого таймера, регистра прерываний (рисунок 19).

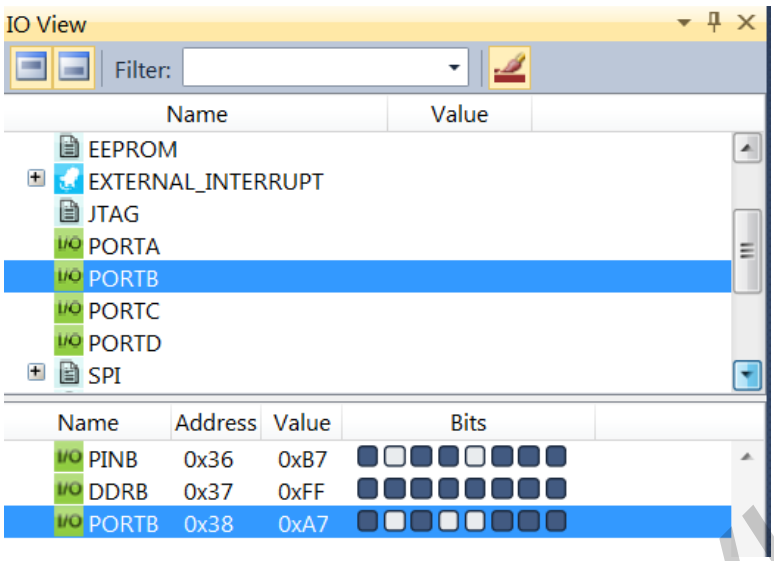

Рисунок 19 – Окно I/O View

Значения регистров могут быть изменены в режиме отладки путем щелчка мышью на значении регистра. Некоторые биты не могут быть изменены, они находятся в режиме «только чтение», некоторые биты могут быть в режиме «только запись». Когда бит установлен, он немедленно читается устройством, таким образом в окне I/O View отображаются актуальные значения. Бит, установленный в 1, отображается как черный квадрат, установленный в 0, – как белый квадрат. Когда бит меняет свое начальное значение квадрат отображается красным цветом. Если происходит переход из 0 в 1, то цвет квадрата меняется с белого на красный. Если происходит переход из 1 в 0, то цвет квадрата меняется с черного на белый с красной рамкой.

В окне **дампа памяти Memory** (рисунок 20) отображается содержимое всех видов памяти микроконтроллера (Flash, EEPROM, регистры, RAM).

Во вкладке **Address** можно указать адрес конкретной ячейки для просмотра. Во вкладке Columns можно настроить объем отображаемой памяти на мониторе компьютера (количество столбцов в таблице на экран монитора). По умолчанию значения представлены в шестнадцатеричной форме.

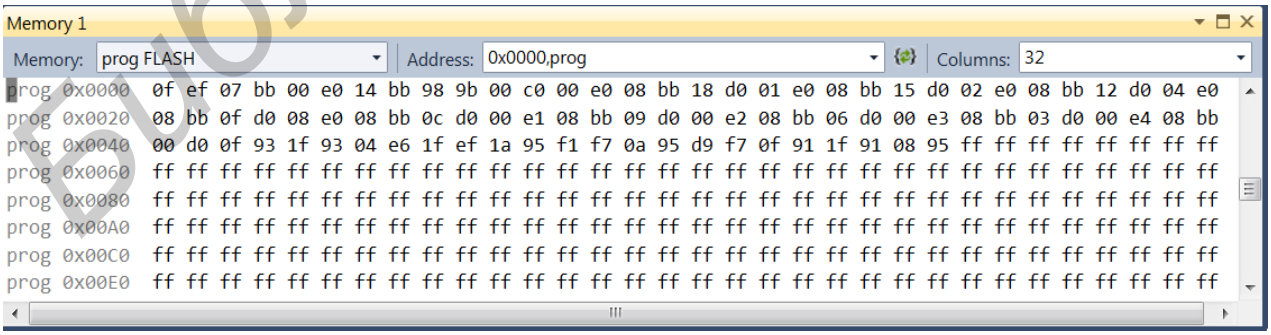

Рисунок 20 – Окно Memory

Точки останова. Точка останова является пометкой, которую можно установить в то место исходного кода, где исполнение должно быть

остановлено. После того как программа достигает точки останова, она останавливается. Установив несколько точек останова, можно следить за ходом выполнения внутри программы. Возможно создание условных точек останова. Эти точки останавливают выполнение программы, только если выполнено некоторое условие, например переменная имеет определенное значение.

Установить точку останова можно, щелкнув кнопкой мыши на серой области справа от кода. Все функции, связанные с точками останова, доступны с помощью контекстного меню, которое появляется при щелчке правой кнопкой мыши на маркере точки останова слева от области кода (рисунок 21).

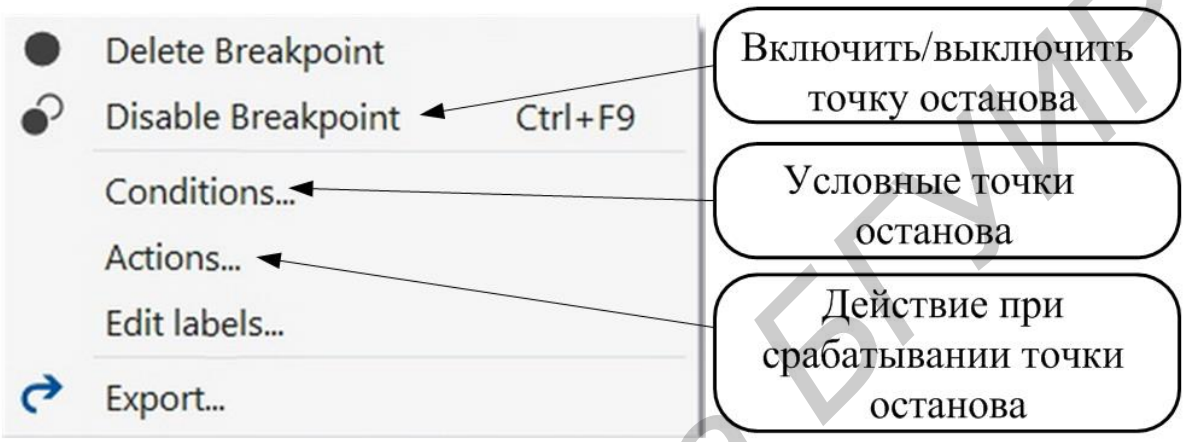

Рисунок 21 – Управление точками останова

Для создания условной точки останова необходимо задать условие. Условие может быть переменной, например «i», или выражением, например «i>30». Если выбран пункт «Is true», точка останова вступит в силу, если условие верно; для вышеприведенного примера – если «i» стало больше 30. Если выбран пункт «Has changed», точка останова вступит в силу при условии, что значение выражения изменилось с момента последнего прохождения точки останова.

Опция «When Hit» позволяет настроить действие, исполняемое при срабатывании точки останова. В текстовом поле можно ввести сообщение, которое будет отображаться при срабатывании точки останова. Сообщение появится в окне Output Window. Можно не только вывести текстовое сообщение, но и вызвать выполнение функции.

Для удобной работы с точками останова предназначено окно Breakpoints (Debug → Windows → Breakpoints). В данном окне отображается информация обо всех точках останова, используемых в программе, и их статусе (рисунок 22).

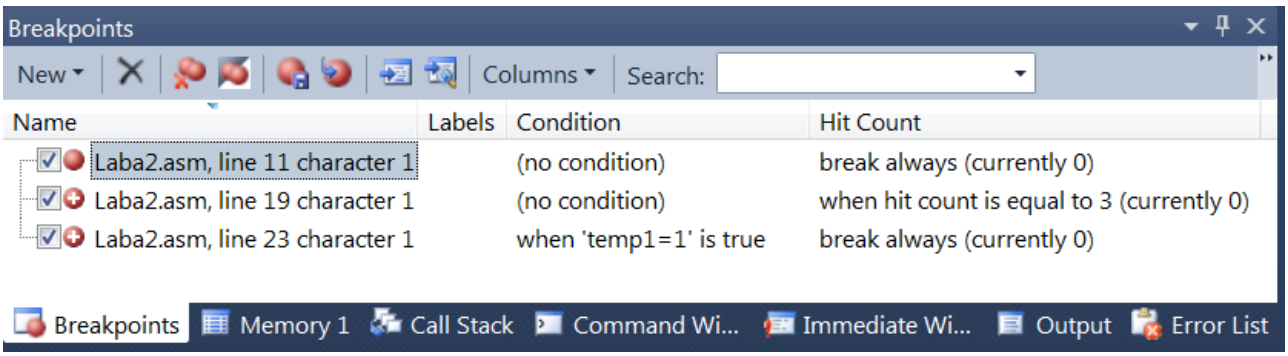

Рисунок 22 – Окно Breakpoints

Окно **Solution Explorer** (рисунок 23) содержит в себе файл-оглавление m16def.inc, в котором прописаны сопоставления символьных имен портов и регистров с адресами микроконтроллера. Здесь же отображается перечень файлов, сгенерированных в проекте (.hex, .map, .obj, .asm). Приводится полный список меток в поле Labels.

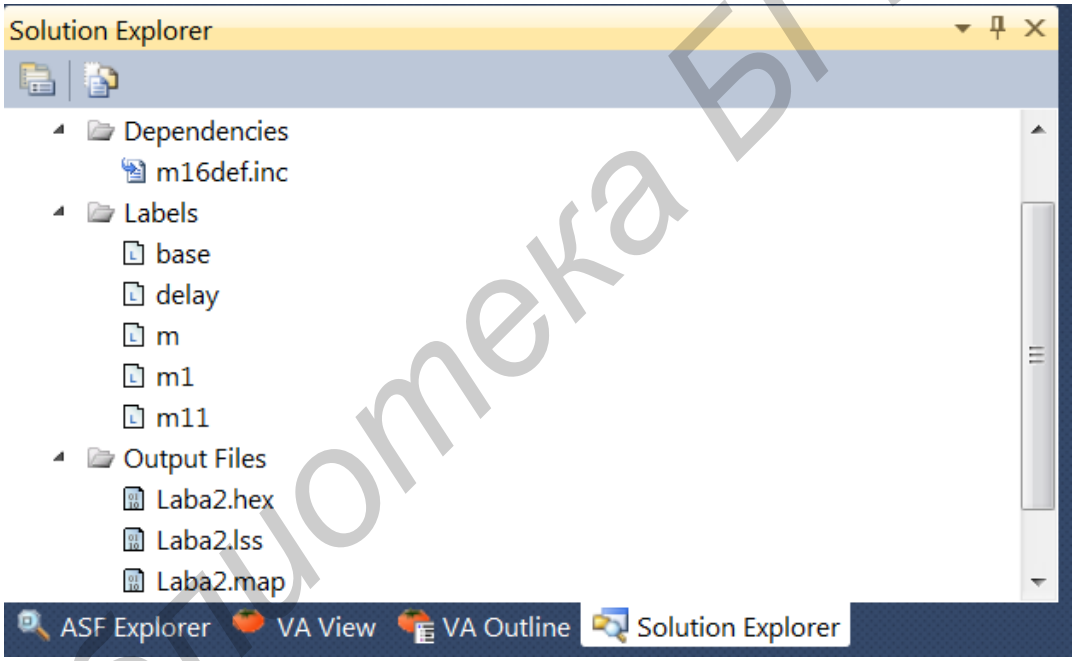

Рисунок 23 – Окно Solution Explorer

Если в процессе компиляции обнаруживается ошибка в коде, выводится соответствующее сообщение об ошибке в **Output Window**. Если дважды щелкнуть на строке, содержащей описание ошибки, среда покажет строку исходного кода, в которой эта ошибка содержится. Иногда компилятор может выдавать много сообщений об ошибках, которые имеют одну и ту же причину. Часто единственная ошибка приводит к появлению множества сообщений. Компилятор генерирует несколько сообщений, чтобы дать больше информации о проблемах, которые он обнаружил. В примере на рисунке 24 все сообщения были порождены одной ошибкой в синтаксисе. В этом случае рекомендуется просматривать сообщения об ошибках сверху вниз, начиная с первого.

Error List

3 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
2	PORTB: Unknown instruction or macro	Laba2.asm	19	0	Laba2
3	syntax error, unexpected ','	Laba2.asm	19	0	Laba2
1	tout: Unknown instruction or macro	Laba2.asm	19	0	Laba2

Рисунок 24 – Окно Error List

3.2 Среда автоматизированного проектирования Proteus

Пользовательский интерфейс

Симуляция работы микроконтроллера будет проводиться в среде Proteus. Proteus – пакет программ для автоматизированного проектирования электронных схем, включает в себя программу синтеза и моделирования электронных схем ISIS и программу разработки печатных плат ARES.

Для корректной работы программы рекомендуется располагать программу и проекты в папках (и путях к папкам) с названиями, не содержащими кириллицу и пробелы. Основное окно ISIS приведено на рисунке 25.

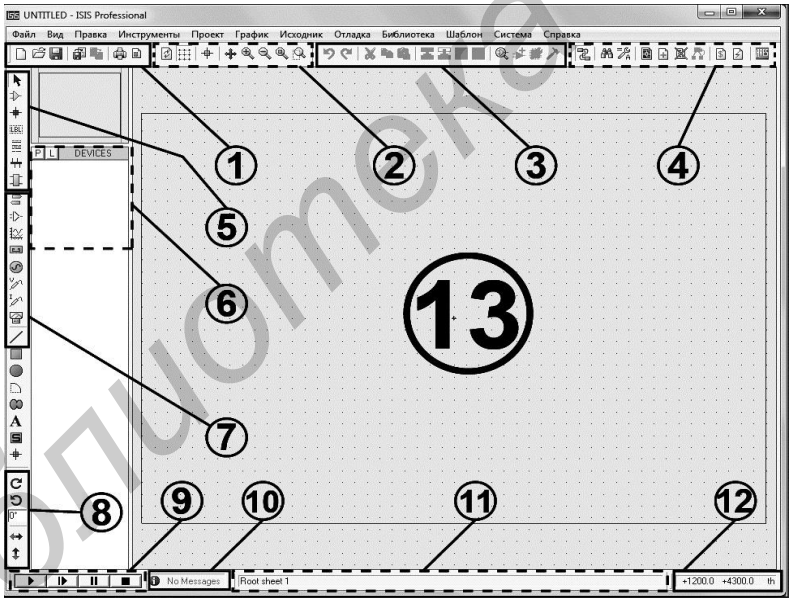

Рисунок 25 – Основное окно ISIS

Под основным меню находится панель инструментов. В нем можно выделить следующие секции:

1 – Файл/Печать;

2 – Вид/Масштаб;

3 – Редактирование;

4 – Инструментарий;

5 – левая панель инструментов;

6 – панель DEVICES отображает компоненты, задействованные в проекте;

7 – левая панель инструментов;

8 – панель ориентации;

9 – кнопки управления симуляцией;

10 – всплывающее окно «Лог сообщений»;

11 – таймер симуляции;

12 – окно координат. Указывает положение курсора относительно центра рабочей области;

13 – рабочая область.

Загрузка проекта

Чтобы открыть проект, необходимо в меню «Файл» выбрать пункт «Открыть проект» и проследовать по заданному пути. Для выполнения лабораторных работ созданы схемы. Необходимо выбрать схему в соответствии с выполняемой работой.

Для загрузки в модель микроконтроллера управляющей программы (предварительно созданной и отлаженной в Atmel Studio) необходимо открыть панель редактирования свойств компонента (Edit Component) микроконтроллера AVR Atmega 16 двойным щелчком левой кнопки мыши на микроконтроллере (рисунок 26).

В поле «Program File» необходимо указать путь к файлу прошивки .hex.

В поле «CKSEL Fuses» задается частота работы микроконтроллера. Если необходимо использовать нерегламентированную частоту работы, то ее указывают в поле «Clock Frecuency».

Схема на рисунке 27 используется при изучении системы команд микроконтроллера и его системы прерываний. Схемы на рисунках 28–29 используются при изучении принципов работы с устройствами ввода и вывода информации.

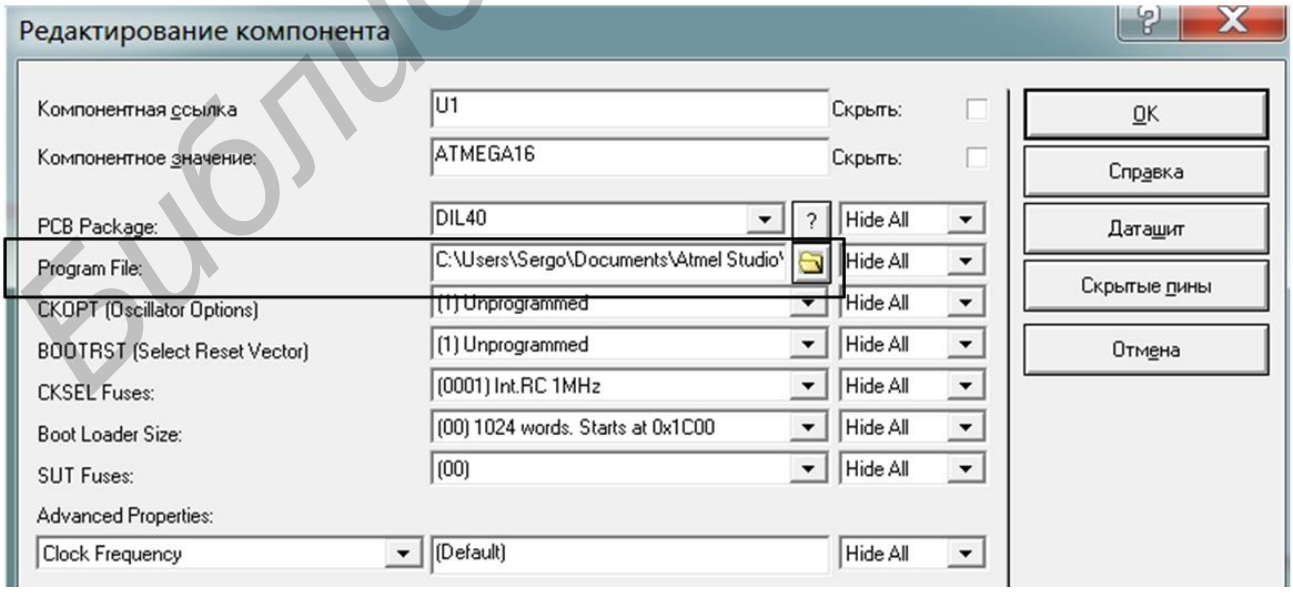

Рисунок 26 – Окно свойств микроконтроллера

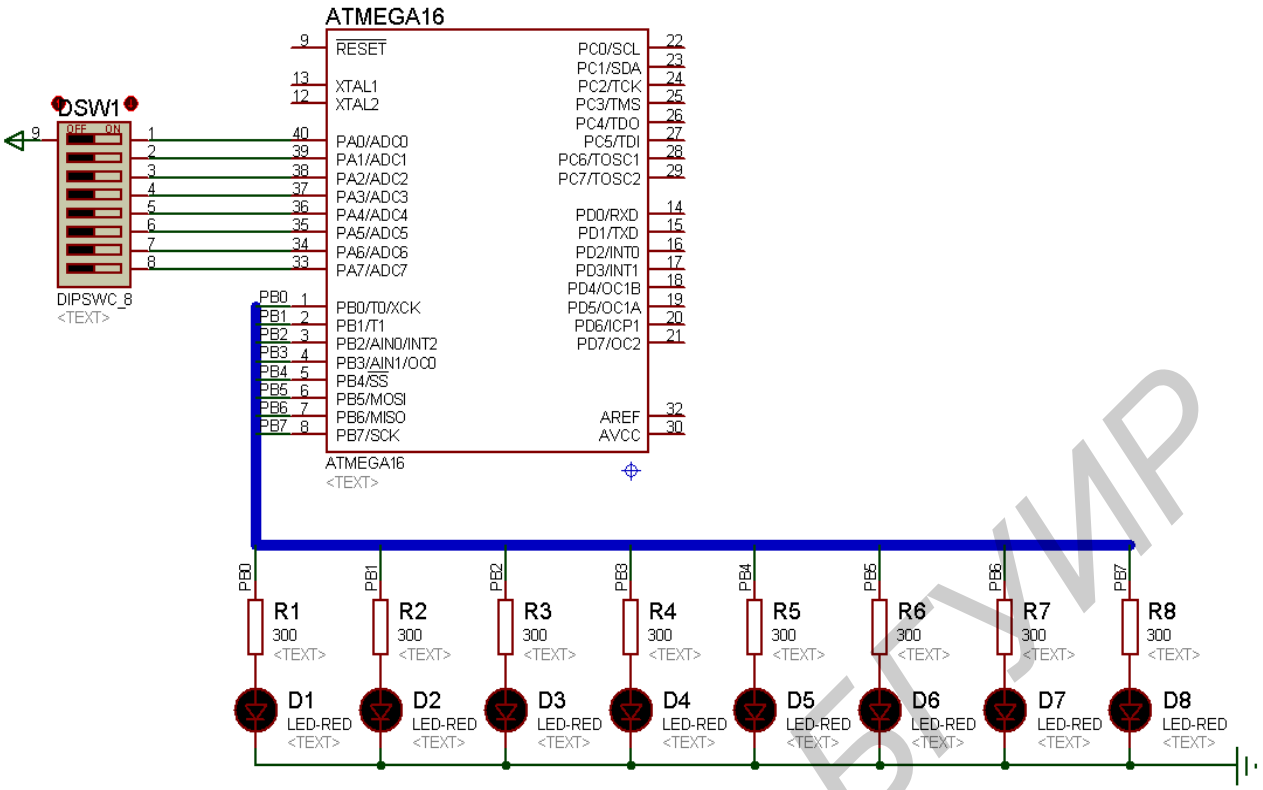

Рисунок 27 – Схема №1 для изучения системы команд микроконтроллера

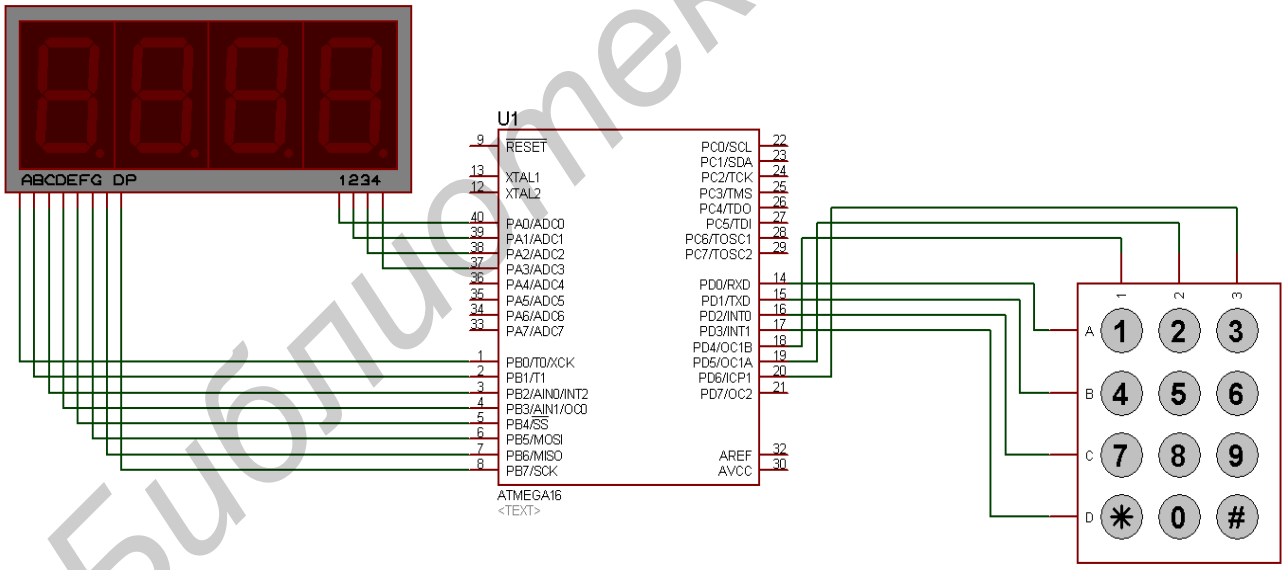

Рисунок 28 – Схема №2 для изучения работы устройств ввода/вывода

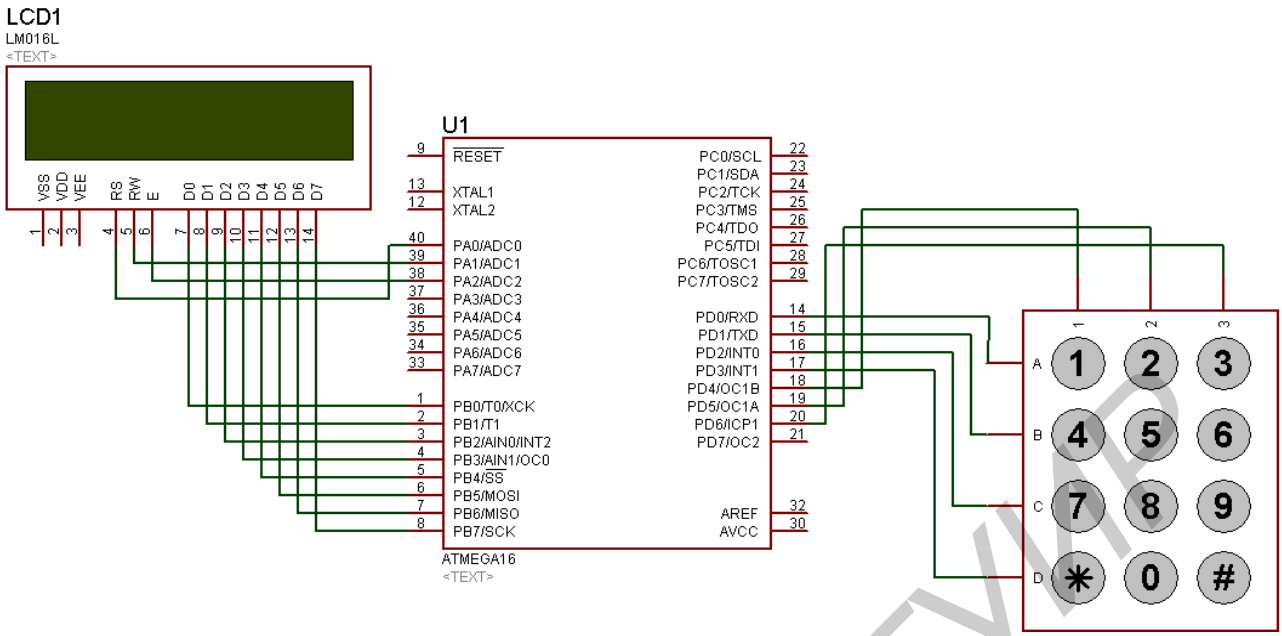

Рисунок 29 – Схема №3 для изучения работы устройств ввода/вывода

Периферийные устройства ввода/вывода

Светодиодные индикаторы и кнопки. Светодиоды и кнопки используются для отладки и макетирования систем. На схеме №1 содержатся перечисленные ниже блоки. Восьмикомпонентный переключатель DIPSWC_8 (рисунок 30). Переключатель имеет 1 вход и 8 выходов. На вход подается логическая 1 (питание). Переключатель в положении «ON» соответствует логической 1 на указанном выходе переключателя. Положению «OFF» соответствует логический 0. Переключатель подключен к портам PA0-PA7, настроенным на ввод данных. Переключатель DIPSWC_8 заменяет собой восемь кнопок. Порт B настроен как выход. К выводам PB0-PB7 подключены светодиоды, загорающиеся при логической 1 на выходе.

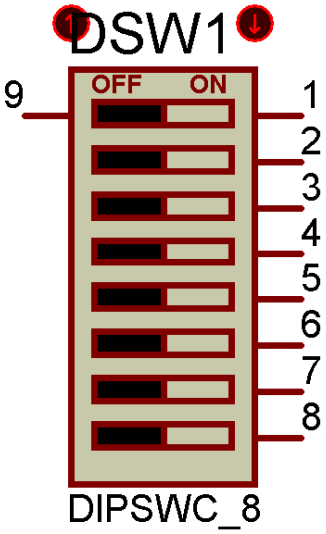

Рисунок 30 – Переключатель DIPSWC_8

Семисегментный индикатор. Индикатор представляет собой восемь светодиодов с общим анодом: семь светодиодов для отображения сегментов цифр, а восьмой светодиод отображает десятичную точку. Индикатор может отображать цифры от 0 до 9, а также некоторые буквы латинского алфавита. Сегменты обозначаются буквами от A до G; восьмой сегмент (десятичная точка – decimal point, DP) предназначен для отображения дробных чисел. Схема светодиодной матрицы представлена на рисунке 31.

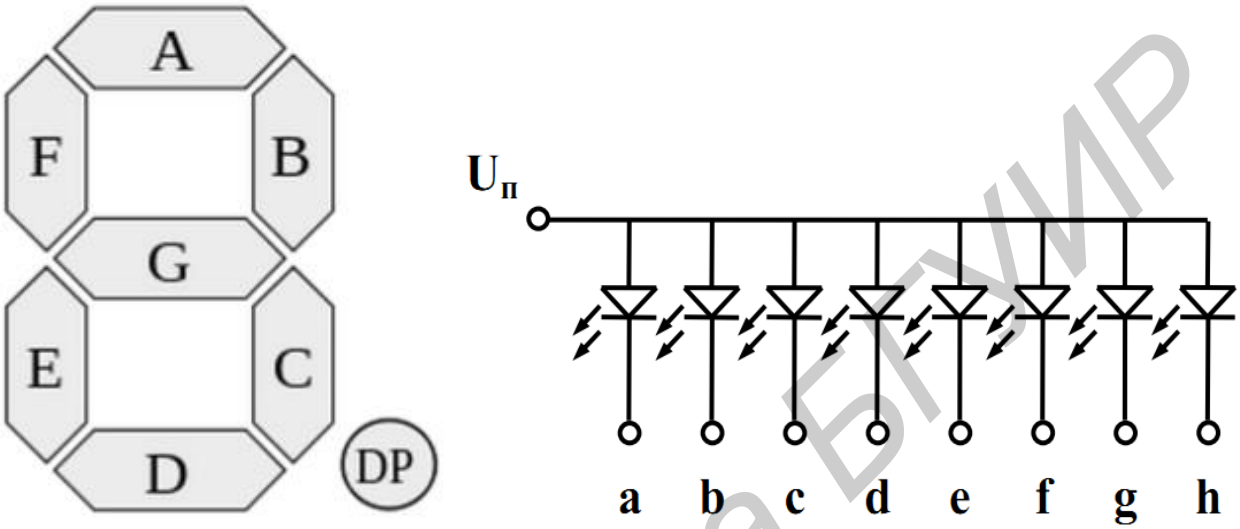

Рисунок 31 – Семисегментный индикатор

В схеме одноименные сегменты всех разрядов объединяются и обслуживаются семью выводами одного из портов микроконтроллера (PB0 – PB7). Общие выводы разрядов индикатора обслуживаются индивидуально четырьмя выводами другого порта микроконтроллера (PA0–PA3). Первый порт обеспечивает возбуждение определенной для каждой цифры комбинации сегментов, а второй порт активизирует тот или иной разряд индикатора. Суть динамического управления индикатором сводится к поочередному циклическому возбуждению разрядов индикатора. Если разряды индикатора будут подсвечиваться с частотой выше 25 Гц, то в силу инерционности человеческого зрения мерцания изображения заметны не будут. Каждому светодиоду необходим токоограничивающий резистор.

Пример – Для того чтобы возбудить сегмент A первого разряда индикатора, необходимо на порт PA0 подать логическую 1, а на порт PB0 – логический 0 (рисунок 32).

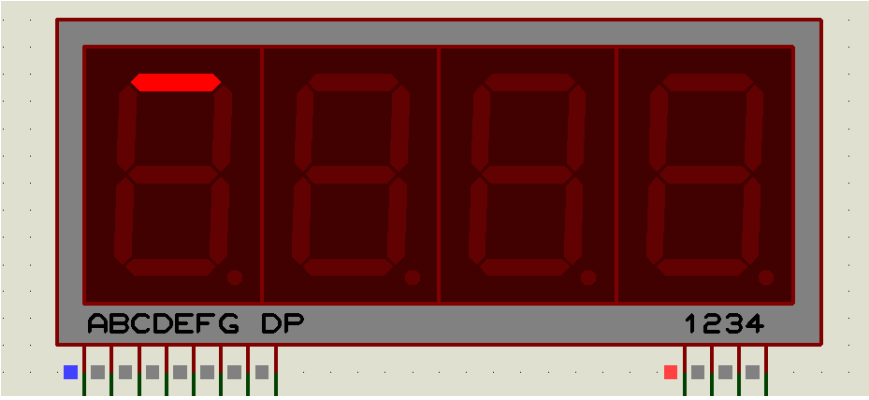

Рисунок 32 – Активация сегмента А

Цифровая клавиатура. Клавиатура представляет собой кнопочный блок, в котором кнопки размещены в виде матрицы на пересечении горизонтальных и вертикальных линий связи (рисунок 33).

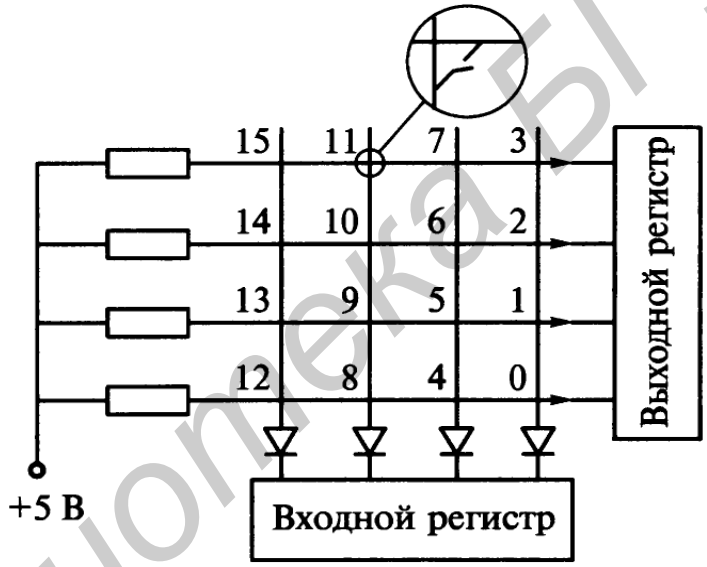

Рисунок 33 – Строение клавиатуры 4х4

Один ряд линий, например вертикальных, подключают к входному регистру, другой ряд (горизонтальных) – к выходному регистру. На входной регистр из контроллера подают код, содержащий нуль в одном разряде и единицы во всех остальных. При замыкании кнопки вертикального ряда, на котором присутствует сигнал 0, этот сигнал поступит в горизонтальную линию и по ней на выходной регистр.

Проверив состояние выходного регистра, контроллер может идентифицировать строку, а вместе со столбцом и номер замкнутой кнопки. С помощью последовательности сканирующих кодов вида 1110, 1101, 1011,0111 можно опросить состояние всех столбцов клавиатуры и установить номер замкнутой кнопки. Используя его как индекс, можно выбрать из таблицы

переходов начальный адрес процедуры, выполняемой при замыкании соответствующей кнопки.

При выполнении лабораторных работ будет использоваться клавиатура, состоящая из 12 кнопок (рисунок 34).

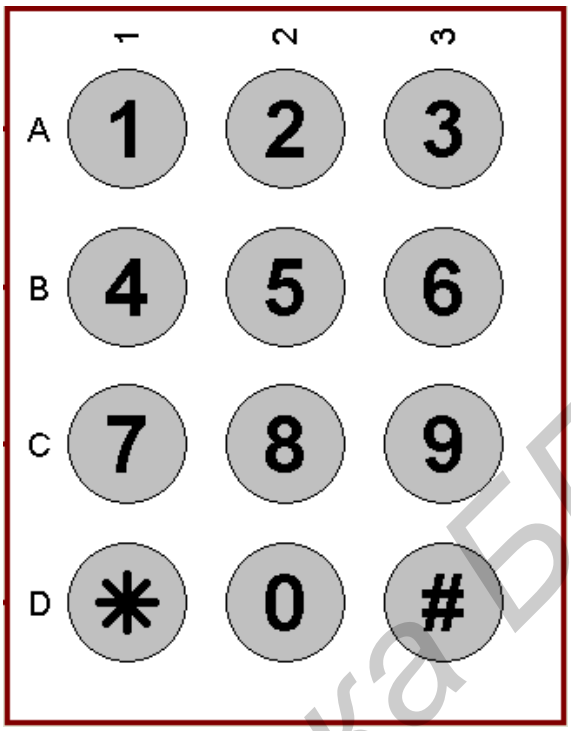

Рисунок 34 – Цифровая клавиатура

В качестве входных/выходных регистров для выводов A, B, C, D используются порты PD0–PD3. Для выводов 1, 2, 3 используются порты PD4–PD6 как выходные/входные.

Устройство отображения символьной информации. Устройство отображения символьной информации LCD LM016L представлено на рисунке 35.

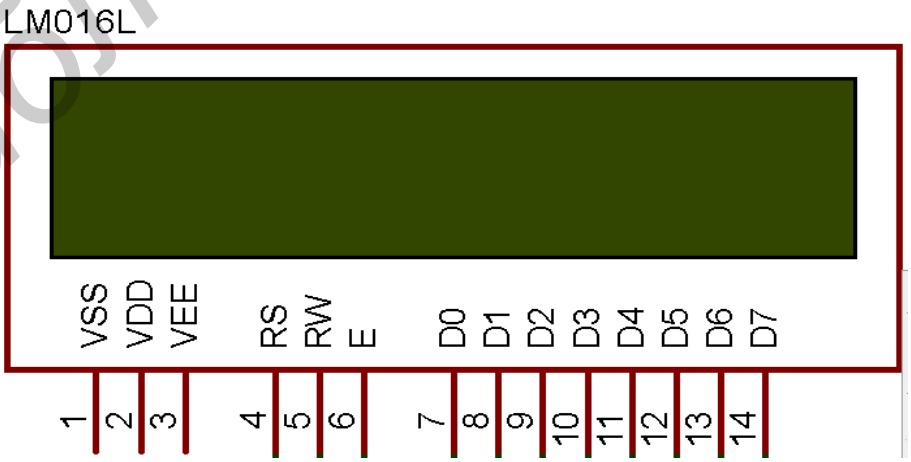

Рисунок 35 – Двухстрочный LCD-дисплей LM016L

Модули LCD-дисплеев состоят из собственно дисплея и схемы управления (контроллер). Контроллер нужен для того, чтобы отличить команды от данных и расшифровать их. Данный дисплей использует контроллер HD44780.

Интерфейс двухстрочного дисплея LM016L содержит восемь входов для передачи команд и данных в контроллер дисплея и три линии управления:

– Vss – земля (минус питания);

– Vcc(Vdd) +5В (плюс питания);

– Vee – контраст;

– RS – вход сигнала управления, сопровождающий передачу команд (RS = 0) и данных (RS = 1);

– RW – вход сигнала управления, определяющий тип обращения к дисплею – запись (RW = 0) и чтение (RW = 1);

– E – вход для синхронизирующего сигнала передачи по шине данных;

– D0:D7 – линии передачи данных.

Выводы RS, RW, E подключены к выводам PA0–PA2. Линии передачи данных D0–D7 подключены к портам PB0–PB7 соответственно.

Данный дисплей поддерживает два режима работы в 4- и 8-битном режиме. Восьмибитный режим отличается более высокой скоростью работы. В рамках данного лабораторного практикума будет рассмотрено использование 8-битного режима работы.

Система команд контроллера HD44780 приведена в таблице 20. Для выполнения каждой из команд требуется определенное время, указанное в таблице. В связи с этим при программировании вывода информации на дисплей необходимо после вывода каждой команды предусмотреть задержку в ожидании завершения заданной операции.

Таблица 20 – Система команд контроллера HD44780

№	D7	D6	D5	D4	D3	D2	D1	D0	Описание команды	Время
1	0	0	0	0	0	0	0	1	Очистка дисплея, курсор по адресу 0	До 1,64 мс
2	0	0	0	0	0	0	1	–	Курсор по адресу 0, дисплей относительно буфера DDRAM в начальной позиции	От 40 мкс до 1,6 мс
3	0	0	0	0	0	1	I/D	S	Курсор сдвигается вправо (I/D = 1) или влево (I/D = 0), сдвиг дисплея (S = 1) вместе с курсором	40 мкс
4	0	0	0	0	1	D	C	B	Включение (D = 1) или выключение (D = 0) дисплея, включение курсора (C = 1) или гашение (C = 0). Мигание курсора (B = 1)	40 мкс
5	0	0	0	1	S/C	R/L	–	–	Сдвиг курсора (S/C = 0) или дисплея (S/C = 1) вправо (R/L = 1) или влево (R/L = 0)	40 мкс
6	0	0	1	DL	N	F	–	–	Разрядность шины данных – четыре (DL = 0) или восемь (DL = 1) бит, количество строк дисплея – одна (N = 0) или две (N = 1), Шрифт – 5 x 7 (F = 0) или 5 x 10 точек (F = 1)	40 мкс
7	0	1	AG	AG	AG	AG	AG	AG	Установка адреса CGRAM	40 мкс

№	D7	D6	D5	D4	D3	D2	D1	D0	Описание команды	Время
8	1	AD	AD	AD	AD	AD	AD	AD	Установка адреса DDRAM	40 мкс
9	BF AC								Чтение состояния busy-флага и счетчика адреса	1 мкс
10	Данные								Запись данных в DDRAM или CGRAM	40 мкс
11	Данные								Чтение данных из DDRAM или CGRAM	40 мкс

Символьные данные для отображения на дисплее поступают в коде ASCII в буфер контроллера дисплея, объем которого составляет 80 байт. В зависимости от режима отображения (однострочный или двухстрочный) данные для вывода на экран представляют один 80-байтовый массив или два массива по 40 байт каждый. При двухстрочном выводе начальный адрес верхней строки (строка 0) составляет 00h, для нижней (строка 1) – 40h. В модели LM016L длина строки составляет 16 позиций. Из этого следует, что длина буферного массива превышает число позиций дисплея. Поэтому для отображения всех элементов массива данных окно дисплея перемещается вдоль массива.

Все элементы, попадающие в окно, отображаются на экране дисплея, остальные остаются вне зоны видимости. Курсор отображается на экране, если предварительно была введена команда отображать его и он находится в зоне видимости. В программе после передачи каждого символа данных необходимо предусмотреть временную задержку 40 мкс.

4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа №1

Изучение интегрированной среды разработки Atmel Studio для программирования микроконтроллеров AVR

Цель: получение навыков создания программного обеспечения при помощи интегрированной среды разработки; создание тестового проекта и отслеживание этапов выполнения программы.

1 Теоретические сведения

Теоретическая часть содержится в подразделе 3.1 «Интегрированная среда разработки Atmel Studio».

2 Практическое задание

Порядок выполнения лабораторной работы:

1. Запустите Atmel Studio.
2. Создайте файл проекта на языке ассемблера.
3. Создайте тестовый файл исходного кода согласно варианту задания (таблица 21)
4. Запустите отладчик, настройте проект Atmel Studio на используемый тип кристалла.
5. Запустите транслятор. Осуществите поиск и исправление ошибок.
6. В режиме симулятора откройте все основные окна проекта.
7. Добавьте точки останова в произвольных местах кода.
8. Установите маркеры DataTip напротив выбранных произвольно переменных.
9. Отследите работу программы в режимах прогона и пошагового выполнения.

Таблица 21 – Задание к лабораторной работе

Вариант 1	Вариант 2	Вариант 3

.def temp = r16
.def tempH = r17
.def Desired = rl8
.def Actual = rl9
rjmp init
Init:
ldi temp, 0b011100
out DDRB, temp
clr temp
out PortB, temp
ldi temp, 0b11101011
out ADCSR, temp
clr temp
out ADMUX, temp
	.def temp = r16
	.def reg_led = rl9
	.org $0
	rjmp init
	.org $001
	rjmp inter
	.org $00E
	rjmp adc
	init:
	ldi temp, low(RAMEND)
	out SPL, temp
	ldi temp, high(RAMEND)
	out SPH, temp
	ser temp
		.def temp = r16
		.def counter = r17
		rjmp init
		Init:
		ldi counter, 0b00000001
		ser temp
		out DDRB, temp
		ldi temp, 0b11111110
		out DDRD, temp
		clr temp
		out PortB, temp
		ldi temp, 0b00000001
		out PortD, temp
		Start:

Вариант 1	Вариант 2	Вариант 3

Start:
cbi ADMUX, 0
sbi ADCSR, ADCS
sbic ADCSR, ADSC
rjmp Start+2
in Desired, ADCH
com Desired
sbi ADMUX, 0
sbi ADCSR, ADSC
Wait:
sbic ADCSR, ADSC
out ADMUX, temp
rjmp Wait
	out DDRC, temp
	out PORTC, temp
	ldi temp,OxFB
	out DDRD, temp
	ldi temp,0x04
	out PORTD, temp
	clr temp
	out DDRA, temp
	out PORTA,t emp
	ldi temp,0x7F
	out GIMSK, temp
	ldi temp,0x02
	out MCUCR, temp
		sbis PinD, 0
		rjmp off
		sbi PortB, 0
		rjmp start
		off:
		cbi PortB, 0
		inc counter
		cpi counter, 11
		rjmp Start

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Листинг исходного кода на языке ассемблера.
4. Описание состояния окон отладки в режиме пошагового выполнения

4 Контрольные вопросы и задания

1. Перечислите основные функции пакета программ Atmel Studio.
2. Приведите алгоритм разработки типичной микропроцессорной системы.
3. Перечислите основные приемы комплексной отладки микропроцессорных систем.
4. Приведите структуру проекта Atmel Studio.
5. Опишите принципы организации и функционирования ядра Atmega 16.
6. Для чего используются точки останова?

Лабораторная работа №2

Изучение принципов организации памяти

Цель: разработка системы межмодульного обмена данными; разработка блок-схемы и программного обеспечения микропроцессорной системы для обмена и визуализации обмена межблочными данными.

1 Теоретические сведения

Теоретическая часть содержится в подразделе 1.6 «Организация памяти».

Часть памяти микроконтроллера отведена для РОН и регистров ввода/вывода. Для работы с регистрами ввода/вывода предназначены команды in (прочитать) и out (записать). Следующий сегмент занимает оперативная память (RAM). Для работы с этими ячейками предназначены команды:

sts – Store Direct to SRAM (прямая запись в ОЗУ);
lds – Load Direct from SRAM (прямая загрузка из ОЗУ);
ld – Load Indirect (косвенная загрузка из ОЗУ).

Пример

```
sts 0x62, Temp                  ; Ячейка памяти задана адресом
sts MemoryCell,Temp             ; Ячейка памяти задана именем
lds Temp1, 0x71
lds Temp1, MemoryCell_2
ld r16, X                       ; Ячейка памяти задана указателем X
```

Для доступа к flash-памяти программ используется механизм косвенной адресации через регистр Z. Для работы с ячейками ПЗУ предназначены команды:

lpm – Load Program Memory (косвенная загрузка из ПЗУ);
spm – Store Program Memory (косвенная запись в ОЗУ).

Команда spm записывает результат только в словах (R0:R1) и в большинстве случаев используется для бутлоадеров.

Пример

```
lpm r10, Z                      ; Загрузить в регистр r10 данные из ячейки ПЗУ
                                ; Адрес ячейки памяти задан указателем Z
```

Ассемблер позволяет задавать линейные массивы как в программной памяти (ПЗУ), так и в оперативной (ОЗУ).

CSEG – программный сегмент, в котором записывается программа.

Пример создания в ПЗУ массива Array, состоящего из 6 элементов, начиная с адреса, на котором стоит метка Array:

```
.cseg
Array:
.db 1, 15, 4, 9, 12, 145, 67, 90
```

DSEG – сегмент данных. В нем выделяется оперативная память. Сегмент данных прописывается в тексте раньше программного сегмента. Директива .dseg указывает на начало сегмента данных. Обычно сегмент данных состоит лишь из директив .byte и меток.

Выделим 3 ячейки памяти: 1 ячейку под переменную Digit; 1 ячейку под переменную Input;1 ячейку под переменную Status.

```
.dseg
Digit:  .byte 1          ; Резервируем 1 байт в SRAM
                         ; Метка Digit говорит о том, что каждое появление Digit в коде
                         ; будет заменено адресом этой ячейки памяти
Input:  .byte 1
Status: .byte 1
```

Обращение к элементам массива может происходить следующим образом:

```
.cseg                            ; Начало программного сегмента
lds r1, Digit                    ; Загружаем в r1 переменную Digit
sts Input, r1                    ; Загружаем в Input содержимое регистра r1
```

или с использованием косвенной адресации

```
ldi Xl, low (Digit)              ; Загружаем в Xl младший байт адреса Digit
ldi Xh, high (Digit)             ; Загружаем в Xh старший байт адреса Digit
ld r1, X                         ; Загружаем в r1 переменную Digit с
                                 ; использованием указателя X
```

2 Практическое задание

1. Перенесите массив из памяти программ в память данных, адреса массивов поочередно выдать на индикаторы.
2. Скопируйте содержимое 10 ячеек памяти в регистры и отобразите их содержимое на индикаторах поочередно.
3. Определите стек, выведите на индикаторы границы стека.
4. Выведите на индикаторы поочередно начальные адреса и содержимое первых ячеек областей ввода/вывода, памяти программ, памяти данных, EEPROM.
5. Прочитайте слово кода операции из памяти программ и выведите на индикатор.
6. Выполните операцию логического сложения двух участков SRAM.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Перечислите основные типы памяти AVR.
2. Почему AVR относят к процессорам со смешанной архитектурой?
3. С какой целью применяются два энергонезависимых устройства памяти в AVR?
4. Опишите алгоритм считывания содержимого flash-памяти.
5. Опишите алгоритм считывания содержимого ОЗУ.
6. Опишите алгоритм считывания содержимого EEPROM.

Лабораторная работа №3

Изучение системы команд микроконтроллера Atmega 16

Цель: изучение основ языка ассемблера микроконтроллера Atmega 16; создание программы на ассемблере, используя команды всех типов адресации.

1 Теоретические сведения

Теоретическая часть содержится в разделе 2 «Система команд Atmega 16».

Пример

Написать программу, которая обнулит регистр R20, а затем сложит число 3 с регистром R20 десять раз и запишет сумму в порт B.

```
.cseg
ldi R16, 10          ; R16 = 10, создаем счетчик на 10 повторений
clr R20              ; R20 = 0, обнуляем регистр R20
ldi R21, 3           ; R21 = 3, заносим число 3 в регистр R21
Loop:
add R20, R21         ; Складываем R20 и 3, заносим сумму в регистр R20
dec R16              ; Уменьшаем счетчик на 1
brne Loop            ; До тех пор пока счетчик не равен нулю, переход по метке Loop
out PortB, R20       ; посылаем результат сложения в  порт B
```

Пример

Написать программу, которая формирует в памяти программ массив из четырех элементов с начальным адресом 0x0300 и поочередно загружает элементы массива в регистр R16.

```
.cseg
.org 0x0300          ; Указываем на адрес первого элемента массива
array:
.db 0x11, 0x12       ; С помощью директивы .db заносим 4 числа (элементы массива)
.db 0x13, 0x14       ; во flash-память

loop2:
ldi R20, 4           ; Заносим в регистр R20 число 4, формируем счетчик

                     ; Формируем указатель Z
ldi zl, low (array*2)  ; Заносим в регистр zl (R30) младший байт адреса первого
                       ; элемента массива
ldi zh, high (array*2) ; Заносим в регистр zh (R31) старший байт адреса первого
                       ; элемента массива
loop:
lpm R16, Z+          ; Загружаем в регистр R16 число по адресу Z, затем
                     ; инкрементируем Z
dec R20              ; Декрементируем счетчик
brne loop            ; До тех пор пока R20 не равен нулю, переходим по метке loop
rjmp loop2           ; Когда R20 станет равным нулю, переходим по метке loop2
```

Примечание – При загрузке адреса первого элемента массива array в Z используется выражение array*2. Причина в том, что каждая команда содержит

два байта информации и занимает две ячейки ПЗУ. Поэтому счетчик команд считает два адреса как один. Метка содержит данные для счетчика команд. Следовательно, чтобы получить реальный адрес ПЗУ, необходимо увеличить адрес метки в 2 раза.

2 Практическое задание

Конкретные значения параметров (адреса, количество переменных) определяются преподавателем.

Варианты заданий к лабораторной работе:

№1. Напишите программу для сложения элементов массива заданной длины и помещения результата сложения в ячейку памяти, следующую за последним элементом.

№2. Напишите программу для сложения элементов двух массивов заданной длины и помещения результата сложения в ячейку памяти, следующую за последним элементом второго массива.

№3. Напишите программу для перемещения элементов массива заданной длины из одной области памяти в другую.

№4. Напишите программу для нахождения среднего арифметического элементов массива заданной длины и помещения результата вычислений в ячейку памяти, следующую за последним элементом.

№5. Напишите программу для нахождения минимального или максимального элемента массива заданной длины и помещения результата вычислений в ячейку памяти, следующую за последним элементом.

№6. Напишите программу для сортировки элементов массива заданной длины методом пузырька по возрастанию или убыванию.

Дайте комментарии к основным блокам программы (что именно делается и для чего это делается). После выполнения основного задания студент получает индивидуальное задание на модификацию программы.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Перечислите способы адресации Atmega 16. Приведите примеры команд.
2. Перечислите основные регистры Atmega 16 и их назначение.
3. Опишите особенности выполнения команд условного перехода (BREQ, BRNE).
4. Перечислите основные арифметические команды.
5. Перечислите основные команды передачи данных.
6. Опишите состав и назначение битов регистра флагов SREG.

Лабораторная работа №4

Изучение портов ввода/вывода микроконтроллера Atmega 16

Цель: приобретение навыков работы с внешними устройствами; имитация работы различных цифровых схем с использованием кнопочных переключателей, цифровых клавиатур, диодных индикаторов.

1 Теоретические сведения

Теоретическая часть содержится в подразделах 1.2 «Порты ввода/вывода» и 3.2 «Среда автоматизированного проектирования Proteus».

Пример программы управления портом панелей клавиш и диодов

К выводам порта B подключены светодиоды (LEDs), а к выводам порта A подключены кнопки (Switches). Программа зажигает светодиоды в соответствии с комбинацией нажатых кнопок.

```
.CSEG                    ; Начало сегмента кода
.def  tmp=r16            ; Присвоить регистру r16 имя tmp
                         ; Настройка портов
ldi tmp, 0b11111111      ; Записать в tmp 0b11111111
out ddrB, tmp            ; Настройка порта B на вывод
ldi tmp, 0b00000000      ; Записать в tmp 0b00000000
out ddrA, tmp            ; Настройка порта A на ввод
                         ; Основная часть программы
CIRCLE:
in tmp, pinA             ; Записать в tmp состояние кнопок
out portB, tmp           ; Передать состояние кнопок в порт B
rjmp CIRCLE              ; Перейти по метке CIRCLE
```

Пример

Реализовать секундомер с управлением от одной кнопки (start/stop). Счет идет от 0 до 255, результат выводится на светодиодные индикаторы в двоичном виде. Панель кнопок подключена к порту A (кнопка управления подключена к выводу PA7), панель светодиодов – к порту B.

```
.CSEG                    ; Начало сегмента кода
.def Temp = r16          ; Объявление имен регистров
.def Delay = r17         ; Объявление имен регистров
.def Delay2 = r18        ; Объявление имен регистров
RESET:
ser Temp                 ; Установить все биты регистра Temp в единицу
out DDRB, Temp           ; Настраиваем порт B на вывод
out PORTA, Temp          ; Подключаем подтягивающие резисторы к выводам порта A
clr Temp                 ; Сбрасываем все биты регистра Temp в нуль

LOOP:
out PORTB, Temp          ; Запись регистра Temp в порт B
sbic PINA, 7             ; Если 7-й бит порта A сброшен (соответствует нажатой
                         ; кнопке), то пропускаем следующую команду (inc Temp)
inc Temp                 ; Иначе увеличиваем значения регистра Temp на 1
```

```
DLY:                ; Задержка
dec Delay           ; Задержка сделана для того, чтобы по нажатию кнопки
brne DLY            ; выполнялся один цикл. Без задержки возможно
dec Delay2          ; выполнение нескольких циклов за одно нажатие кнопки
brne DLY
rjmp LOOP           ; Переход по метке LOOP
```

Подключение внутреннего подтягивающего резистора играет важную роль.

Режим высокоимпендансного входа включен по умолчанию, сопротивление порта очень велико. Этот режим хорош для прослушивания какой-либо шины данных, т. к. не оказывает на шину никакого влияния. В случае, когда вход ни к чему не подключен, напряжение будет на нем изменяться в зависимости от внешних наводок и электромагнитных помех.

При DDRx = 0 и PORTx = 1 замыкается ключ подтяжки и к линии подключается резистор в 100 кОм, что приводит неподключенную линию в состояние логической 1. Цель подтяжки – не допустить изменения состояния на входе под действием наводок.

Если на входе появится логический нуль (замыкание линии на землю кнопкой), то 100-комный резистор не сможет удерживать напряжение на линии на уровне логической 1 и на входе будет нуль.

С целью снижения энергопотребления и повышения надежности рекомендуется все неиспользованные выводы включить в режим Pull-Up.

При подключении подтягивающего резистора и нажатии кнопки на соответствующий вход поступает логический 0.

2 Практическое задание

Используя схему №1, приведенную на рисунке 27, для изучения системы команд микроконтроллера (возможно использование дополнительных кнопок, buttons), реализовать одно из следующих заданий:

1. Бегущий огонь из одного светодиода по линейке восьми светодиодов без смены направления.
2. Бегущий огонь из одного светодиода по линейке восьми светодиодов из стороны в сторону.
3. Цикл последовательного зажигания светодиодов.
4. Разбегающиеся огни из центра по сторонам.
5. Сбегающиеся огни из сторон к центру.
6. Бегущий огонь из семи светодиодов по линейке восьми светодиодов без смены направления.
7. Один цикл разбегания огней по нажатии клавиши.
8. Изменение направления бегущей строки двумя клавишами.
9. Увеличение частоты мигания светодиода при нажатии одной кнопки, уменьшение частоты мигания при нажатии другой кнопки.
10. Последовательное цикличное зажигание светодиодов с номера нажатой клавиши.

11. Последовательное цикличное зажигание светодиодов до номера нажатой клавиши.

12. Вывод на линейку светодиодов двоичного числа, одна клавиша – инкремент этого числа, другая – декремент.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Перечислите способы изменения состояния на линии порта.
2. Объясните назначение регистра PORTx.
3. Как поведет себя микроконтроллер в случае чтения или записи в регистр PORTx некоторого значения?
4. Объясните назначение регистра DDRx.
5. Как поведет себя микроконтроллер в случае чтения или записи в регистр DDRx некоторого значения?
6. Объясните назначение регистра PINx.
7. Как поведет себя микроконтроллер в случае чтения или записи в регистр PINx некоторого значения?
8. Поясните принцип управления подтягивающими резисторами.

Лабораторная работа №5

Изучение принципов работы с внешними устройствами, клавиатурой и семисегментным индикатором

Цель: изучение принципов подключения и работы дисплея и клавиатуры с динамическим сканированием; создание приложения для работы с матричной клавиатурой и дисплеем динамической индикации.

1 Теоретические сведения

Теоретическая часть содержится в подразделе 3.2 «Среда автоматизированного проектирования Proteus».

Динамическая индикация – это метод отображения целостной картины через быстрое последовательное отображение отдельных элементов этой картины. Визуально картинка получается статической благодаря инерционности человеческого зрения.

Первым делом необходимо сформировать рисунок цифр. Для этого необходимо в программу занести таблицу кодов семисегментного индикатора. Это можно сделать несколькими способами: просто расположить ее в памяти (в EEPROM или во flash-памяти) или сформировать 10 процедур (по одной на каждую цифру от 0 до 9), в которых непосредственно устанавливаются нужные разряды портов.

Для схемы подключения с общим катодом активация сегмента происходит путем подачи на него логической 1. Для того чтобы потушить сегмент, на него подают логический 0. Для схемы с общим анодом необходимо инвертировать значения сигналов.

Пример – Процедура для управления семисегментным индикатором с общим катодом.

```
.def count=r20
.CSEG                          ; Начало сегмента кода
.ORG 0x0000                    ; Адрес первого элемента таблицы
Digits:                        ; Таблица символов семисегментного индикатора
.DB 0b00111111, 0b00000110     ; Цифры 0, 1
.DB 0b01011011, 0b01001111     ; Цифры 2, 3
.DB 0b01100110, 0b01101101     ; Цифры 4, 5
.DB 0b01111101, 0b00000111     ; Цифры 6, 7
.DB 0b01111111, 0b01101111     ; Цифры 8, 9
ser r16                        ; Установка битов регистра r16 в 1
out ddrA, r16                  ; Настройка порта A в качестве выхода
out ddrB, r16                  ; Настройка порта B в качестве выхода
ldi r16, 0b00001110            ; Загрузить в регистр r16 число 0b00001110
out portA, r16                 ; Активировать крайний правый разряд индикатора
loop2:
ldi count, 10                  ; Загрузка в счетчик числа 10 (число цифр от 0 до 9)
ldi Zl, low (Digits)           ; Инициализация регистра-указателя Z
ldi Zh, high (Digits)          ; Помещаем в Z адрес первого элемента таблицы
```

```
loop:
lpm r17, Z+
out portB, r17
call pause                ; Вызов подпрограммы задержки
dec count                 ; Декремент счетчика
brne loop                 ; Переход по метке loop до тех пор, пока
                          ; значение в счетчике отлично от нуля,
rjmp loop2                ; Иначе переход по метке loop2

pause:                    ; Подпрограмма для формирования задержки
met2:
nop
met:
dec r18                   ; Декрементировать регистр r18
brne met                  ; До тех пор пока значение в r18 отлично от нуля,
dec r19                   ; переходить по метке met
brne met2
ret                       ; Возврат из подпрограммы
```

Пример – Идентификация нажатой кнопки.

Для того чтобы считать символ с клавиатуры, нужно на столбцы Col3–Col1 поочередно выставить логический 0. После того как на одном из портов столбца PB4–PB6 (Col3–Col1) возник логический 0 и на остальных портах столбца – логическая 1, проверяется состояние портов строк PB0–PB3 (Row4–Row1) на наличие 0. Регистр Z указывает на таблицу во flash-памяти, содержащую коды кнопок. После того как код определения кнопки закончит работу, регистр Z будет указывать на код нажатой кнопки. С помощью инструкции LPM можно прочитать этот код и сохранить его в регистр R0.

```
ReadKey:
ldi ZH, HIGH(2*KeyTable)  ; Z указывает на таблицу кодов кнопок
ldi ZL, LOW(2*KeyTable)
ldi rmp, 0b00111111
out pKeyOut, rmp          ; Чтение столбца 1 (Col1)
in rmp, pKeyInp           ; Чтение строк
ori rmp, 0b11110000       ; Маскирование старших битов
cpi rmp, 0b11111111       ; Нажата ли кнопка в этом столбце?
brne KeyRowFound          ; Найдена нажатая кнопка в этом столбце
adiw ZL, 4                ; В этом столбце не было нажатия,
                          ; перемещаемся на 4 кнопки дальше
                          ; Чтение столбца 2 (Col2)
ldi rmp, 0b01011111       ; PB5 = 0
out pKeyOut, rmp
in rmp, pKeyInp           ; Снова чтение строк
ori rmp, 0b11110000       ; Снова маскирование старших битов
cpi rmp, 0b11111111       ; Нажата кнопка в этом столбце?
brne KeyRowFound          ; Найдена нажатая кнопка в этом столбце
adiw ZL,4                 ; В этом столбце не было нажатия,
                          ; перемещаемся на 4 кнопки дальше
                          ; Чтение столбца 3 (Col3)
```

```
ldi rmp, 0b01101111          ; PB4 = 0
out pKeyOut, rmp
in rmp, pKeyInp              ; Последнее чтение строк
ori rmp, 0b11110000          ; Снова маскирование старших битов
cpi rmp, 0b11111111          ; Нажата кнопка в этом столбце?
breq NoKey                   ; Не нажата ни одна кнопка
KeyRowFound:                 ; Найден столбец, где нажата кнопка,
                             ; теперь надо узнать, в какой именно строке
lsr rmp                      ; Сдвиг логический влево, бит 0 при этом
                             ; смещается во флаг переноса
brcc KeyFound                ; Клавиша найдена
adiw ZL,1                    ; Выбираем следующую кнопку
rjmp KeyRowFound             ; Повторить сдвиг
KeyFound:                    ; Найдена нажатая кнопка
lpm                          ; Прочитать из таблицы код кнопки в R0
rjmp KeyProc                 ; Продолжить обработку кнопок
NoKey: rjmp NoKeyPressed     ; Не была найдена нажатая кнопка
KeyTable:                    ; Таблица кодов кнопок
.DB 0x0A, 0x07, 0x04, 0x01   ; Первый столбец, кнопки *, 7, 4 и 1
.DB 0x00, 0x08, 0x05, 0x02   ; Второй столбец, кнопки 0, 8, 5 и 2
.DB 0x0B, 0x09, 0x06, 0x03   ; Третий столбец, кнопки #, 9, 6 и 3
```

2 Практическое задание

Для выполнения лабораторной работы воспользуемся схемой №2 (см. рисунок 28). Создайте текст программы согласно варианту задания.

1. Составьте подпрограмму опроса клавиатуры.
2. Составьте программу ожидания нажатия комбинации клавиш.
3. Составьте программу подсчета количества нажатых клавиш.
4. Осуществите вывод кода нажатой клавиши на дисплей.
5. Составьте программу для увеличения или уменьшения кода, выдаваемого на индикаторы в зависимости от нажатия клавиш «*» или «#».
6. Создайте простой калькулятор.
7. Создайте таймер. По нажатии кнопки начинается отсчет от 0 до 9999.
8. Переведите число из двоичного кода в BCD-код.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Что такое семисегментный индикатор?
2. На чем основан принцип действия динамической индикации?
3. Назовите последовательность действий, необходимую для отображения символов на четырех индикаторах (один цикл работы).

Лабораторная работа №6

Изучение таймеров и системы прерываний

Цель: изучение системы прерываний микроконтроллера Atmega 16 и принципа работы таймера/счетчика; изучение основ работы таймеров и решения различных задач в реальном времени с их помощью.

1 Теоретические сведения

Теоретическая часть содержится в подразделах 1.3 «Система прерываний» и 1.4 «Таймеры/счетчики».

Таймер/счетчик является одним из самых ходовых ресурсов AVR-микроконтроллера. Работа таймера напоминает работу секундомера, его основное назначение – отсчитывать заданные временные интервалы. Кроме этого содержимое таймера может инкрементироваться внешними импульсами. В таком случае таймер превращается в счетчик событий. Кроме того, таймеры/счетчики могут выполнять ряд дополнительных функций – формирование ШИМ-сигналов, подсчет длительности входящих импульсов.

Пример – Бегущий огонь на светодиодах. Задержка между переключениями диодов регулируется с помощью таймера/счетчика T0.

```
.def temp=r16                ; Присвоить регистру r16 имя temp
.def temp2=r17               ; Присвоить регистру r17 имя temp2
.def lamp=r18                ; Присвоить регистру r18 имя lamp
.cseg                        ; Начало сегмента кода
.org 0                       ; По адресу 0 находится
jmp reset                    ; переход по метке reset
.org 0x012;                  ; По адресу 0x012 находится
jmp TIM0_OVF                 ; переход по метке TIM0_OVF (вектор прерывания)
reset:                       ; Обработчик прерывания по сбросу (reset)
ldi temp, low(ramend)        ; Инициализация стека
out spl, temp
ldi temp, high(ramend)
out sph, temp
ser temp                     ; Установить все биты temp в 1
out ddrB, temp               ; Настройка порта B на выход
ldi temp, 0b00000101
out tccr0, temp              ; Настройка предделителя. Частота счетчика T0 = тактовая
                             ; частота микроконтроллера /1024
ldi temp, 0b00000001
out timsk, temp              ; Разрешить прерывание по переполнению счетчика 0
ldi lamp, 0b00000001         ; Загрузить в lamp число 0b00000001
sei                          ; Разрешить глобальные прерывания (установить флаг I)
Basic: rjmp Basic            ; Бесконечный цикл
```

```
TIM0_OVF:          ; Обработчик прерывания по переполнению таймера T0
cli                ; Сбросить флаг I (запретить прерывания)
cpi lamp,1         ; Сравнить lamp с числом 1
breq diod2         ; При совпадении перейти по метке diod2
cpi lamp,2         ; Сравнить lamp с числом 2
breq diod3         ; При совпадении перейти по метке diod3
cpi lamp,4         ; Сравнить lamp с числом 4
breq diod4         ; При совпадении перейти по метке diod4
cpi lamp,8         ; Сравнить lamp с числом 8
breq diod5         ; При совпадении перейти по метке diod5
cpi lamp,16        ; Сравнить lamp с числом 16
breq diod6         ; При совпадении перейти по метке diod6
cpi lamp,32        ; Сравнить lamp с числом 32
breq diod7         ; При совпадении перейти по метке diod7
cpi lamp,64        ; Сравнить lamp с числом 64
breq diod8         ; При совпадении перейти по метке diod8
cpi lamp,128       ; Сравнить lamp с числом 128
breq diod1         ; При совпадении перейти по метке diod1

delay:             ; Задержка
ldi temp2, 0x00    ; Загрузить в temp2 число 0
out tcnt0, temp2   ; Задать начальное значение счетчика
sei                ; Разрешить глобальные прерывания
reti               ; Выйти из процедуры обработки прерывания

diod1:             ; Зажечь диод 1 и сформировать задержку
ldi lamp,1         ; Загрузить в lamp число 1
out portB, lamp    ; Передать lamp в portB
rjmp delay         ; Перейти по метке delay
diod2:             ; Зажечь диод 2 и сформировать задержку
ldi lamp,2         ; Загрузить в lamp число 1
out portB, lamp    ; Передать lamp в portB
rjmp delay         ; Перейти по метке delay
diod3:
ldi lamp,4
out portB, lamp
rjmp delay
diod4:
ldi lamp,8
out portB, lamp
rjmp delay
diod5:
ldi lamp,16
out portB, lamp
```

```
rjmp delay
diod6:
ldi lamp,32
out portB, lamp
rjmp delay
diod7:
ldi lamp,64
out portB, lamp
rjmp delay
diod8:
ldi lamp,128
out portB, lamp
rjmp delay
```

Можно выделить несколько основных блоков в данной программе. Указание векторов прерываний (по сбросу и по переполнению таймера/счетчика T0). Обработчик прерывания по сбросу содержит в себе код для инициализации стека, настройки порта B в качестве выхода, настройки режима работы таймера/счетчика T0 (тактовая частота, разрешение прерывания по переполнению), записи в регистр lamp числа 1, разрешения глобальных прерываний. Обработчик прерывания по переполнению таймера/счетчика T0 содержит в себе код определения числа, хранящегося в lamp, и активации соответствующего диода с формированием задержки.

2 Практическое задание

1. Напишите программу, которая увеличивает значение переменной temp на 2 каждую секунду, и выводит результат на панель светодиодов каждую секунду.
2. Напишите программу, которая работает как обратный таймер, уменьшает значение переменной temp каждую секунду на 1.
3. Напишите программу мигания светодиодами. Один светодиод мигает с частотой 10 Гц, второй – с частотой 20 Гц.
4. Напишите программу управления частотой мигания светодиода. Начальная частота мигания светодиодом – 80 Гц, по нажатии одной клавиши увеличивать частоту в два раза, при нажатии другой клавиши – уменьшать в два раза.
5. Посчитайте количество нажатий клавиши за 10 с.
6. С 4-секундной задержкой после нажатия INT запустите таймер с соответствующей индикацией.
7. Напишите программу, реализующую секундомер. Управление секундомером осуществляется тремя копками: первая – запуск, вторая – остановка, третья – сброс.
8. По нажатии INT постепенно зажигать светодиод (ШИМ).

9. Изменяйте клавишами яркость свечения светодиода (ШИМ).

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Как устроена структура системы прерывания Atmega 16?
2. Сколько таймеров/счетчиков общего назначения может располагаться на борту микроконтроллеров рассматриваемого семейства?
3. Каковы отличия между таймерами/счетчиками T0, T1 и T2?
4. Какие прерывания используются таймерами/счетчиками?
5. Как производить управление предделителями таймеров/счетчиков?
6. Каким образом выполняется настройка источника тактового сигнала?
7. Поясните работу таймеров/счетчиков в режиме Normal.
8. Поясните работу таймеров/счетчиков в режиме CTC.
9. Поясните работу таймеров/счетчиков в режиме Fast PWM.
10. Поясните работу таймеров/счетчиков в режиме Phase correct PWM.

Лабораторная работа №7

Изучение принципов работы с последовательным интерфейсом

Цель: реализация последовательного канала связи; изучение основ работы модуля USART в микроконтроллере Atmega 16 и составление программ с использованием USART.

1 Теоретические сведения

Теоретическая часть содержится в подразделе 1.5 «Последовательные порты».

Микроконтроллер Atmega 16 имеет в своем составе один модуль универсального синхронно/асинхронного приемопередатчика (USART). Модуль приемопередатчика обеспечивает полнодуплексный обмен по последовательному каналу, при этом скорость передачи данных может варьироваться в широких пределах. Поток данных, передаваемых по каналу USART, представляет собой совокупность посылок или кадров. Каждый кадр содержит стартовый бит, 8 или 9 бит данных и стоповый бит. Стартовый бит имеет уровень логического 0, стоповый – уровень логической 1.

Выводы микроконтроллера, используемые модулем USART, являются линиями порта D. В качестве входа приемника (RXD) используют вывод PD0, а в качестве выхода передатчика (TXD) – вывод PD1.

Пример подпрограммы передачи по интерфейсу USART

Эта подпрограмма принимает символ с виртуального терминала, увеличивает код принятого символа на 1 и передает получившееся значение обратно на терминал 15 раз.

```
.def temp=r16                    ; Присвоить регистру r16 имя temp
.def sys=r17                     ; Присвоить регистру r17 имя sys
.def count=r18                   ; Присвоить регистру r18 имя count
.equ bitrate=9600                ; Присвоить метке bitrate значение 9600
.equ mode=1000000/(16*bitrate)-1 ; Вычислить значение UBRR для обычного
                                 ; асинхронного режима и присвоить его метке mode
.cseg                            ; Начало программного сегмента
.org 0x0                         ; По адресу 0 находится
jmp reset                        ; переход по метке reset
.org 0x016                       ; По адресу 0x016 находится
jmp USART_RXC                    ; переход по метке USART_RXC
.org 0x01A                       ; По адресу 0x01A находится
jmp USART_TXC                    ; переход по метке USART_TX
reset:                           ; Обработчик прерывания по сбросу (reset)
ldi temp, high(ramend)           ; Инициализация стека
out sph, temp
ldi temp, low(ramend)
out spl, temp
```

```
ldi temp, high(mode)          ; Задать скорость передачи в бодах для
out UBRRH, temp               ; нормального асинхронного режима
ldi temp, low(mode)
out UBRRL, temp
ldi temp, 0b11011000          ; Загрузить в temp число 0b11011000
out UCSRB, temp               ; Настроить прерывания USART
ldi temp, 0b10000110          ; Загрузить в temp число 0b11011000
out UCSRC, temp               ; Задать количество разрядов данных в посылках
sei                           ; Разрешить глобальные прерывания
Basic: rjmp Basic             ; Бесконечный цикл

USART_RXC:          ; Обработчик прерывания по завершении приема USART
cli                           ; Сбросить флаг I (запретить прерывания)
sbis UCSRA, RXC               ; Пропустить следующую команду, если в регистре
                              ; UCSRA установлен флаг RXC
rjmp USART_RXC                ; Перейти по метке USART_RXC
in sys, UDR                   ; Записать в sys значение из регистра данных UDR
inc sys                       ; Инкрементировать sys
out UDR, sys                  ; Передать в регистр данных UDR значение из sys
ldi count, 0                  ; Загрузить в count число 0
UR:
sei                           ; Разрешить глобальные прерывания
reti                          ; Выйти из процедуры обработки прерывания

USART_TXC:          ; Обработчик прерывания по завершении передачи USART
Cli                           ; Сбросить флаг I (запретить прерывания)
sbis UCSRA, UDRE              ; Пропустить следующую команду, если в регистре
                              ; UCSRA установлен флаг UDRE
rjmp USART_TXC                ; Перейти по метке USART_TXC
inc count                     ; Инкрементировать count
cpi count, 15                 ; Сравнить count с числом 15
breq loop                     ; Перейти по метке loop в случае совпадения
inc sys                       ; Инкрементировать sys
out UDR, sys                  ; Передать в регистр данных UDR значение из sys
UT:
Sei                           ; Разрешить глобальные прерывания
Reti                          ; Выйти из процедуры обработки прерывания
loop:
ldi count, 0                  ; Обнулить счетчик count
rjmp UT                       ; Перейти по метке UT
```

2 Практическое задание

Для выполнения лабораторной работы необходимо подключить к микроконтроллеру виртуальный терминал (virtual terminal).

Вывод RXD терминала подключается к выводу TXD микроконтроллера. Вывод TXD терминала подключается к выводу RXD микроконтроллера. В настройках терминала выбрать позиции Hex Display Mode и Echo Typed Characters.

Варианты заданий:

1. Примите 1 байт и байт подтверждения окончания передачи.
2. Примите 1 байт и байт подтверждения окончания передачи.
3. Примите 2 5-битных кадра.
4. Передайте 2 5-битных кадра.
5. Примите подряд 3 байта, каждый с различной скоростью передачи.
6. Передайте подряд 3 байта, каждый с различной скоростью передачи.
7. Примите массив из 10 байт.
8. Передайте массив из 10 заданных байтов.
9. Примите последовательность байтов, где каждая пара – скорость передачи последующей пары байт.
10. Передайте последовательность байтов, где каждая пара – скорость передачи последующей пары байт.
11. Определите скорость передачи USART.
12. Выведите побайтово посылку на индикаторы при получении данных из линии.
13. Настройте USART на произвольный режим работы, осуществите передачу массива данных из одной области памяти в другую через USART.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Какие прерывания используются модулем USART?
2. Какие внештатные ситуации может обнаруживать модуль USART?
3. Перечислите регистры управления модулем USART, опишите их назначение.
4. Как настроить скорость приема/передачи?
5. Поясните, можно ли передавать данные с одной скоростью, а принимать с другой.
6. Какие недостатки появятся у модуля USART в случае приема/передачи с ошибкой в несколько процентов?
7. Как формируется кадр USART?
8. Каким образом происходит прием/передача 9-битных данных?
9. Приведите пример расчета скорости последовательного обмена.

Лабораторная работа №8

Изучение принципов работы с внешними устройствами, жидкокристаллической панелью

Цель: разработка системы символьного и псевдографического отображения информации; реализация на программном уровне драйвера управления ЖК-панелью.

1 Теоретические сведения

Теоретическая часть содержится в подразделе 3.2 «Среда автоматизированного проектирования Proteus».

Пример

После инициализации портов микроконтроллера, таймера и дисплея производится однократный вывод сообщения на верхнюю строку дисплея. Поскольку текст сообщения не изменяется при повторных запусках программы, он размещен в памяти программ. Символы кнопок отображаются на нижней строке дисплея. После заполнения строки дисплея весь экран очищается с помощью соответствующей команды.

Управляющие сигналы поступают на дисплей из микроконтроллера по линиям PA5–PA7. Входы клавиатуры (вертикальные) соединены с входами PD0–PD3, горизонтальные входы – с PD4–PD7.

```
.def temp=r16
.def cols=r18
.def rots=r21
.def key=r19
.def scancod=r20
.def lcd=r22
.def lcd_count=r23
.equ rs=5
.equ rw=6
.equ e=7
.org 0x00
rjmp init
.org 0x07
rjmp scankes
init:                          ; Инициализация
ldi temp, low (RAMEND)         ; Инициализация указателя стека
out spl, temp
ldi temp, high (RAMEND)
out sph, temp
ser temp                       ; Инициализация портов
out DDRA, temp                 ; Порт PA на вывод
out DDRC, temp                 ; Порт PC на вывод
ldi temp, 0x0F                 ; Линии порта PD0–PD3 на вывод
out DDRD, temp                 ; Линии порта PD4–PD7 на ввод
ldi temp, 0xF0
```

```
out DDRD, temp
ldi temp, 0x05                          ; Инициализация таймера 0
out TCCR0, temp                         ; Коэффициент деления 1024
ldi temp, (<<TOIE0)
out TIMSK, temp                         ; Разрешение прерываний от таймера
rcall delay2ms                          ; Инициализация ЖК-дисплея
ldi led, 0x38                           ; 8-битовый режим вывода,
rcall ledcom                            ; 2 строки, шрифт 5×7
ldi led, 0x06
; Направление сдвига курсора вправо без сдвига дисплея
rcall ledcom
ldi led, 0x0C                           ; Включить дисплей,
rcall ledcom                            ; Погасить курсор
ldi led, 0x01
; Очистить дисплей и установить курсор в нулевую позицию
rcall ledcom
rcall delay2ms
ldi led, 0x80                           ; Установка адреса буферной памяти для
rcall ledcom                            ; вывода верхней строки дисплея
ldi count_lcd, 12                       ; Вывод на верхнюю строку дисплея
ldi zl, low (str_0*2)
ldi zh, high( str_0*2)
out0: lpm
adiw zl, 1
mov lcd, r0
rcall lcddat
dec count_lcd
brne out0
ldi lcd, 0xC0                           ; Установка адреса буферной памяти для
rcall lcdcom                            ; вывода нижней строки дисплея
ldi count_lcd,17
sei                                     ; Разрешение прерываний
loop: rjmp loop
scankeys:                               ; Подпрограмма обработки клавиатуры
clr key
clr scancod
ldi cols,4
sec
scan: rol scancod                       ; Формирование очередного скан-кода
out PORTD, scancod                      ; Вывод на клавиатуру
clc
ldi rots, 4
in temp, PIND                           ; Ввод состояния клавиатуры
mm:rol temp
brcc nn                                 ; Проверка замыкания кнопки
rjmp press                              ; Переход при замыкании (C=0)
nn: inc key                             ; Увеличение номера кнопки
dec rots                                ; Уменьшение номера строки
brne mm
dec cols                                ; Уменьшение номера ряда
brne scan
```

```
press: cpi key,16
breq fl                          ; Выход при отсутствии замыканий
rcall lcd_str_l                  ; Вывод на дисплей
fl: reti
; Подпрограмма вывода значения клавиши на нижнюю строку дисплея
lcd_str_l:
dec count_lcd
brne met
; При достижении конца строки обновить счетчик вывода на дисплей, очистить дисплей
и установить курсор в нулевую позицию
ldi count_lcd,16
ldi led, 0x01
rcall lcdcom
rcall delay2ms
ldi lcd, 0xC0                    ; Установка адреса буферной памяти на
rcall ledcom                     ; начало нижней строки
met: ldi zl, low (str_l*2)       ; Определение символа клавиши по ее
ldi zh, high (str_l*2)           ; номеру и вывод на дисплей
add zl, key
brcc me2
inc zh
me2: lpm
mov lcd, r0
rcall lcddat
ret
lcdcom:                          ; Подпрограмма вывода на дисплей данных
out PORTC, lcd                   ; Вывод команды
ldi temp, 0x80                   ; Установка режима записи команд
out PORTA, temp
cbi PORTA, e                     ; Фронт 1/0 строба
rcall delay40us
ret
lcddat:                          ; Подпрограмма вывода на дисплей данных
out PORTC, lcd                   ; Вывод символа
ldi temp, 0xA0                   ; Установка режима записи данных
out PORTA, temp
cbi PORTA, e                     ; фронт 1/0 строба
rcall delay40us
ret
delay40us:                       ; Задержка 40 мкс при Fclk = 3,69 МГц
ldi rl8, 48
d0: dec rl8
brne d0
ret
delay2ms:                        ; Задержка 2 мс
ldi rl7, 48
d1: rcall delay40us
dec rl7
brne dl
ret
; Текст сообщения для верхней строки дисплея
```

```
str_0: .db «K», «E», «Y», «P», «A», «D», « » , «+» « » «L» , «C» , «D»
; Обозначения клавиш, выводимые на нижнюю строку дисплея
str_l: .db «/» , «x» , «-» , «+» «9», «6», «3» = «8», «5», «2», «0», «7», «4», «l», «*»
```

2 Практическое задание

1. Составьте программу вывода сообщения по нажатии кнопки.
2. Составьте программу отображения числа в шестнадцатеричной системе счисления по нажатии кнопки
3. Составьте программу отображение числа в двоичной системе по нажатии кнопки.
4. Составьте программу вывода блуждающего символа по нажатии кнопки.
5. Составьте программу вывода сообщения на две строки по нажатии кнопки.
6. Составьте программу посимвольного вывода сообщения по нажатии кнопки.

3 Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Блок-схема алгоритма программы.
4. Листинг кода программы на языке ассемблера с комментариями.

4 Контрольные вопросы и задания

1. Какое количество символов содержит код ASCII?
2. Приведите алгоритм программирования контроллера дисплея для 4-разрядной шины данных.
3. Приведите временные диаграммы цикла записи данных и команд в память дисплея.
4. Перечислите основные доступные команды дисплея.

Литература

1. Евстифеев, А. В. Микроконтроллеры AVR семейства Mega : руководство пользователя / А. В. Евстифеев. – М. : Изд. дом «Додэка-XXI», 2007. – 592 с.
2. Ревич, Ю. В. Практическое программирование микроконтроллеров Atmel на языке ассемблера / Ю. В. Ревич. – 2-е изд. испр. – СПб. : БХВ-Петербург, 2011. – 362 с.
3. Хартов, В. Я. Микроконтроллеры AVR. Практикум для начинающих / В. Я. Хартов. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2012. – 240 с.
4. Трамперт, В. Измерение, управление и регулирование с помощью AVR микроконтроллеров / В. Трамперт. – Киев : МК-Пресс, 2006. – 200 с.
5. Белов, А. В. Разработка устройств на микроконтроллерах AVR / А. В. Белов. – М. : Наука и техника, 2012. – 528 с.

Св. план 2015, поз. 48

Учебное издание

Петров Сергей Николаевич

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА. МИКРОКОНТРОЛЛЕРЫ AVR. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. И. Герман*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 10.06.2016. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,23. Уч.-изд. л. 5,0. Тираж 100 экз. Заказ 275.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6