Министерство образования и науки Российской Федерации
Вологодский государственный университет
Кафедра управляющих и вычислительных систем

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ

Архитектура и программирование AVR-микроконтроллеров

Методические указания к лабораторным работам

Часть 1

Факультет электроэнергетический

Направление: 13.03.02 «Электроэнергетика и электротехника»
профиль «Электропривод и автоматика»

Вологда
2014

УДК 621.375 (03)

Микропроцессорные системы. Архитектура и программирование AVR-микроконтроллеров: методические указания к лабораторным работам. – Ч. 1. – Вологда: ВоГУ, 2014. – 39 с.

Описана первая часть курса лабораторных работ по программированию AVR-микроконтроллеров на языке ассемблера в интегрированной отладочной среде AVR Studio. Приведены основные сведения по AVR-архитектуре микроконтроллеров, языку ассемблера Atmel AVR. Методические указания подготовлены для студентов направления 13.03.02 «Электроэнергетика и электротехника» (профиль «Электропривод и автоматика»)

Утверждено редакционно-издательским советом ВоГУ

Составитель А.М. Водовозов, канд. техн. наук, профессор

Рецензент: Н.Д. Поздеев, канд. техн. наук, доцент, заведующий кафедрой электроснабжения,

ВВЕДЕНИЕ

Контроллером называется управляющее устройство, осуществляющее в технической системе разнообразные регулирующие или контролирующие функции. Контроллер, реализованный на одном кристалле полупроводника в виде интегральной схемы, считается микроконтроллером. По сути, микроконтроллер является однокристальным компьютером, ориентированным на выполнение задач управления техническими объектами.

Сегодня практически все известные производители интегральных схем имеют микроконтроллеры в своей номенклатуре изделий. Они различаются структурой процессорного ядра, системой команд, разрядностью, объемом памяти, набором устройств ввода/вывода, конструктивным исполнением, энергопотреблением и целым рядом других не менее важных параметров. Микроконтроллеры фирмы Atmel с AVR- архитектурой – одни из самых распространенных в России при разработке систем промышленной автоматики. Они характеризуются доступностью, простотой использования, хорошим сочетанием производительности, энергоэффективности и гибкости проектирования. Эти изделия сопровождаются бесплатным программным обеспечением, обеспечивающим пользователя совершенными инструментами для создания и отладки проектов.

Лабораторный практикум ориентирован на изучение AVR-микроконтроллеров и формирования у студентов навыков их программирования на языке ассемблера с использованием интегрированной отладочной среды Atmel AVR Studio.

1. АРХИТЕКТУРА AVR

Идея разработки нового AVR-архитектуры микроконтроллера принадлежит двум норвежским студентам - Альфу Богену (Alf-Egil Bogen) и Вегарду Воллену (Vegard Wollen). В 1995 году они предложили американской корпорации Atmel свой первый 8-битный AVR - микроконтроллер. В конце 1996 года корпорация выпустила опытный микроконтроллер AT90S1200, а во второй половине 1997-го – приступила к серийному производству нового семейства микроконтроллеров.

AVR- микроконтроллер, как и большинство аналогичных схем, состоит из трех связанных системными шинами элементов: процессорного ядра, памяти и набора программируемых функциональных блоков различного назначения (рисунок 1.1).

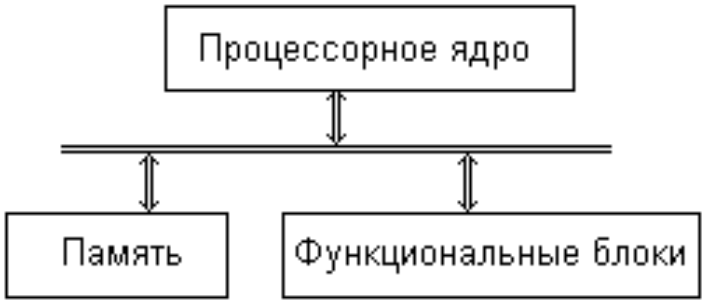


Рисунок 1.1 - Структура микроконтроллера

Процессорное ядро является основой микроконтроллера. Оно выполняет все вычислительные операции и одновременно управляет работой всех остальных элементов схемы. По системным шинам процессорное ядро обменивается данными с памятью и всеми функциональными блоками. Разрядность процессорного ядра определяет разрядность микроконтроллера. Наиболее распространены в настоящее время 8-битные микроконтроллеры. Вместе с тем, широкое применение в сложных высокопроизводительных системах находят и выпускаемые Atmel 32-битные изделия.

Множество инструкций (команд), которые понимает процессорное ядро, образует систему команд микроконтроллера. Система команд AVR-микроконтроллера близка к идеологии известной RISC-архитектуры (Reduced Instruction Set Computer). AVR-архитектура насчитывает в различных моделях до 133 различных инструкций. Большинство инструкций занимает только 1 ячейку памяти и выполняется за 1 такт работы ядра.

Всё множество инструкций микроконтроллеров AVR производитель условно разбивает на 4 группы:

- арифметические и логические инструкции;
- инструкции пересылки данных;
- инструкции ветвления;
- битовые инструкции.

В памяти (Memory) хранится программа работы микроконтроллера, исходные данные и все промежуточные результаты вычислений. Память AVR-микроконтроллера имеет Гарвардскую архитектуру. Она разделена на две независимые части: память данных (Data Memory) и память программ (Program Memory). 8-разрядная память данных является энергозависимой и выполняется по схемам SRAM (Static Random Access Memory). При выключении питания все данные в этой памяти теряются. 16-битная память программ энергонезависима и выполняется по технологии Flash-memory. Программа в эту память заносится на стадии программирования изделия и сохраняется при отключении питания.

Функциональные блоки различных типов обеспечивают взаимодействие микроконтроллера с внешним миром. Эти блоки могут выполнять самые различные функции: ввод и вывод информации, подсчет внешних событий и интервалов времени, передача внешних запросов на процессорное ядро, аналого-цифровые и цифроаналоговые преобразования сигналов, сравнение различных величин, контроль за напряжением питания и др. Набор функциональных блоков в AVR- микроконтроллерах достаточно широк. В большинстве микросхем присутствуют: развитая система прерываний, система сброса, система энергосбережения, параллельные порты ввода/вывода, таймеры/счетчики, последовательные интерфейсы, энергонезависимая память, аналого-цифровые преобразователи, аналоговые компараторы.

2. ИНТЕГРИРОВАННАЯ ОТЛАДОЧНАЯ СРЕДА AVR STUDIO

Подготовка программ для микроконтроллера всегда выполняется в несколько этапов:

º создание текста программы;

º трансляция текста в машинные коды и исправление синтаксических ошибок;

º отладка программы и устранение логических ошибок;

º окончательное программирование микроконтроллера.

Каждый из этапов требует использования специальных программных и аппаратных средств. Такие средства разрабатываются как производителями микроконтроллеров, так и независимыми фирмами и всегда ориентированы на конкретную архитектуру микроконтроллера. В лабораторном практикуме для проведения лабораторных работ используется интегрированная отладочная среда AVR Studio фирмы Atmel, распространяемая производителем бесплатно с сайта www.atmel.ru.

Исходным для создания проекта в среде AVR Studio является файл на языке ассемблера. Этот файл содержит текст программы и является входным для программ-трансляторов, которые, в свою очередь, создают четыре новых файла:

- º Файл-листинг (с расширением .lst) – отчет о работе транслятора. В нем приводится транслируемая программа в виде исходного текста, каждой строке которого сопоставлены соответствующие двоичные коды. Кроме того, листинг содержит сообщения о выявленных ошибках;
- º Объектный файл (с расширением .obj) – используется в дальнейшем как входной для программы-отладчика AVR Studio;
- º Файлы прошивки flash-памяти (с расширением .hex) и EEPROM (с расширением .eep), предназначены для работы с программаторами.

Отладка программы выполняется прямо в среде AVR Studio при помощи встроенного симулятора. Симулятор отображает на экране компьютера программу пользователя, состояние внутренних регистров микроконтроллера, ячеек памяти и регистров ввода/вывода. В результате, появляется возможность для наблюдения за изменениями в регистрах памяти и процессорном ядре микроконтроллера при выполнении тех или иных команд программы.

Отладочная среда AVR Studio поддерживает выполнение программ в виде ассемблерного текста формата AVR Assembler. Программа версии 4 и выше функционирует в среде Windows XP/7/8.

Значок запуска программы изображен на рисунке 2.1.

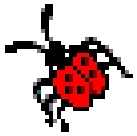


Рисунок 2.1 - Значок программы AVR Studio

В результате запуска AVR Studio на экране появляется стартовое окно программы (рисунок 2.2).

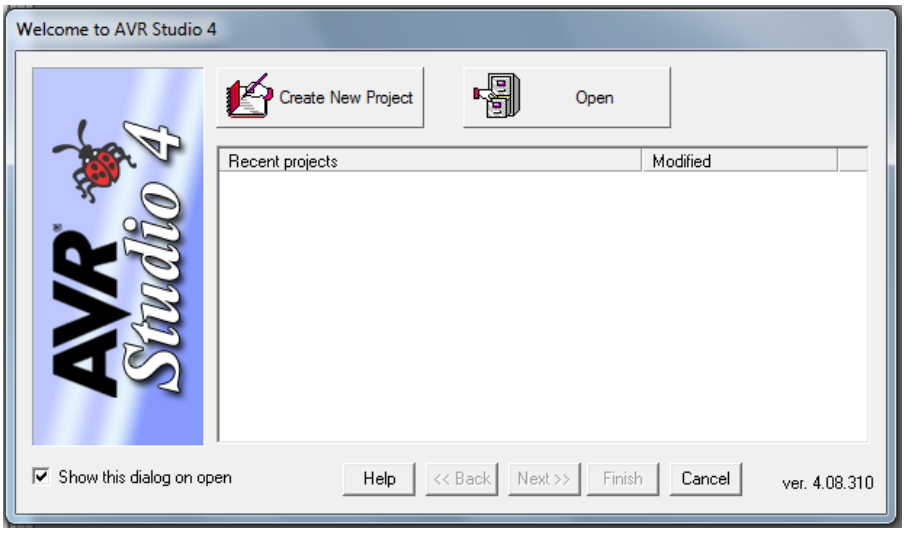
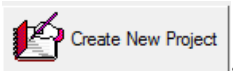


Рисунок 2.2 - Стартовое окно программы **AVR Studio**

Для создания нового проекта необходимо нажать на кнопку 

. В результате на экране появляется диалоговое окно Welcome to AVR Studio 4 (рисунок 2.3), в котором необходимо ввести название проекта (Project name) и задать его расположение (Location). Для нового проекта в операционной системе Windows предварительно создается рабочая папка. В наименовании папки и на всем пути до неё должны быть только символы латинского алфавита. Использование кириллицы не допускается.

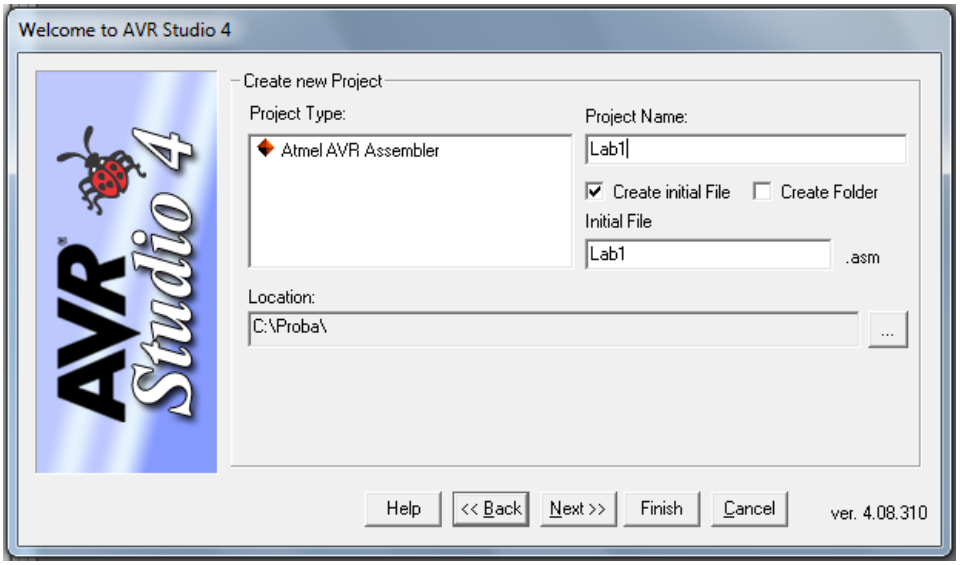
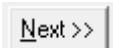


Рисунок 2.3 - Диалоговое окно нового проекта

Далее, после нажатия кнопки 

, в новом диалоговом окне (рисунок 2.4) выбирается отладочная среда (Debug Platform) и тип микроконтроллера (Device) :

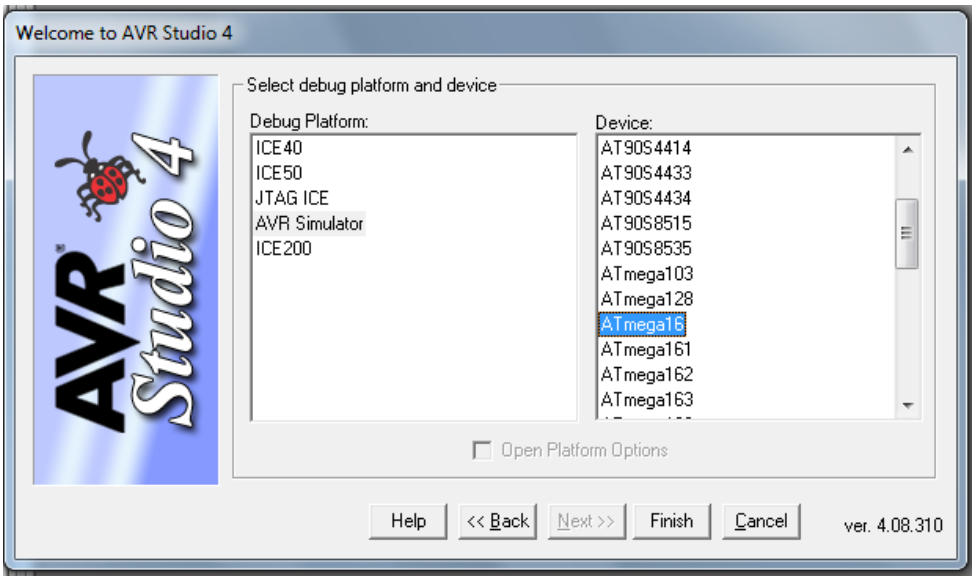


Рисунок 2.4 – Диалоговое окно выбора отладочной среды и типа микроконтроллера

Для лабораторных работ необходимо выбрать AVR Simulator и контроллер ATmega16. При нажатии кнопки Finish на экране появляется окно для создания и редактирования программы на языке ассемблера (рисунок 2.5).

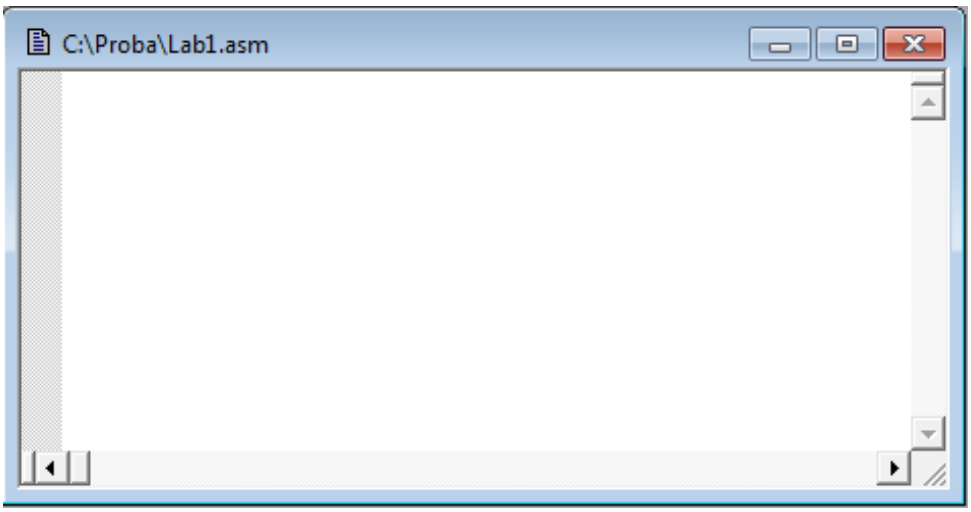


Рисунок 2.5 – Окно редактирования программы

Для проверки и компиляции программы в выпадающем меню Project (рисунок 2.6) предусмотрен пункт Build. В процессе компиляции создается файл с расширением hex, предназначенный для записи в память программ микроконтроллера.

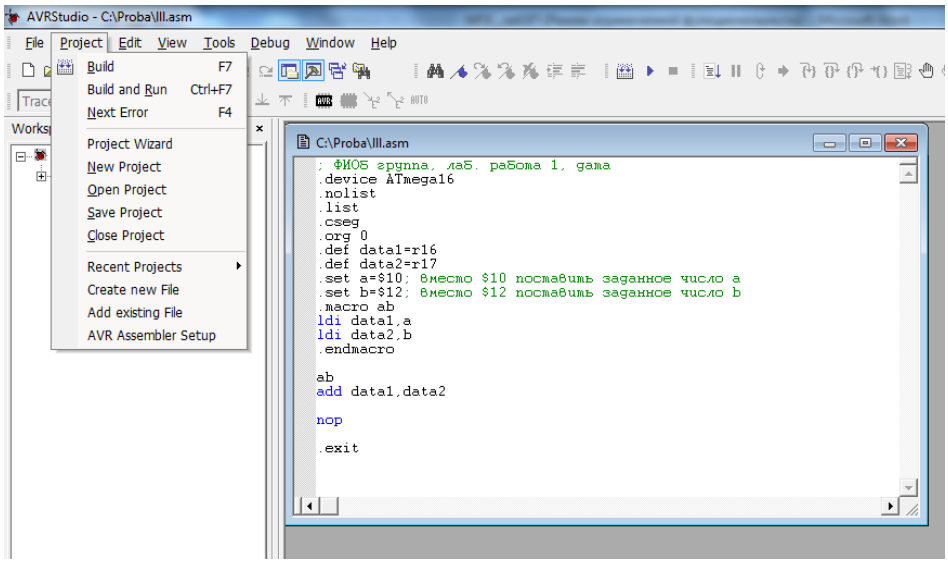


Рисунок 2.6 – Меню программы

После компиляции появится окно Build (рисунок 2.7), в котором приводится информация об используемых и созданных в процессе компиляции файлах и наличии ошибок. В последнем случае в окне указываются тип ошибки, номер строки с ошибкой и в конце общее число ошибок.

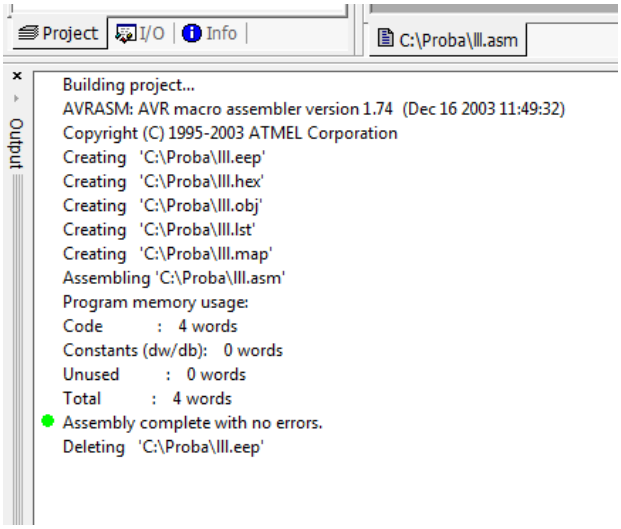


Рисунок 2.6 – Окно с результатами компиляции проекта

Для исправления ошибок необходимо вернуться к редактируемому файлу, а затем снова откомпилировать программу.

Отладочная среда AVR Studio позволяет отлаживать созданные программы. При этом работа микроконтроллера эмулируется. Эмуляция позволяет рассмотреть процесс выполнения программы, как если бы она была записана в микроконтроллер.

Для отладки программы необходимо в меню Project выбрать пункт Build and run.

Для настройки опций симуляции необходимо вызвать окно опций эмулятора (Simulation Options) в меню Debug. В пункте устройство (Device) нужно выбрать микроконтроллер Atmega16, в пункте Frequency – частоту 8 МГц.

Для просмотра регистров в главном меню программы предназначен пункт просмотр (View), в котором можно отметить нужные для отладки узлы микроконтроллера: регистры (Registers), порты ввода/вывода (I/O), ячейки памяти (Memory) и др.

AVR-Studio позволяет запустить программу в реальном времени, в пошаговом режиме, до указателя. В главном меню в пункте отладка (Debug) находятся все варианты запуска программы. Reset — сброс на начало программы (желтая стрелка указателя показывает на начало), Go — запуск в реальном времени (программа будет выполняться до тех пор, пока не будет выбран пункт Break), Step over - пошаговый режим (программа выполняется построчно, при этом останавливается после каждой команды, стрелка указывает на текущую команду), Run to cursor —выполнять до курсора (программа выполняется до места, отмеченного курсором в окне с редактируемой программой). Во время выполнения программы можно наблюдать за состоянием регистров после каждой команды, тем самым проверяется правильность операций, производимых микроконтроллером. Наиболее удобный режим для этого — пошаговый.

3. ЯЗЫК АССЕМБЛЕРА

Язык ассемблера – язык программирования микропроцессорных систем, ориентированный на определенную архитектуру системы. Программа, написанная на языке ассемблера, переводится в машинные коды с помощью специального компилятора, встроенного в отладочную среду.

Язык ассемблера использует систему команд процессорного ядра и специальные директивы, указывающие программе ассемблеру, как организовать различные секции программы, где располагать данные, как связать отдельные процедуры и т. д.

Компилятор транслирует исходные коды с языка ассемблера в объектный код. Полученный объектный код для исполнения должен быть загружен в Flash-память программ микроконтроллера.

Программа на языке ассемблера состоит из отдельных строк. Строка кода не должна быть длиннее 120 символов. Строчные и заглавные буквы рассматриваемый ассемблер не различает.

При написании текста программы используются директивы и инструкции.

Директивы ассемблера не транслируются в коды операций, они выполняют целый ряд различных вспомогательных операций, облегчающих процесс программирования. В языке AVR Assembler любая директива начинается с точки.

Инструкции преобразуются в машинные коды, заносятся в память и выполняются процессорным ядром. Они указывают, какие действия должно выполнить процессорное ядро.

Инструкции микроконтроллера и директивы языка ассемблера оперируют выражениями.

Выражения

Выражением считается набор операндов, связанных между собой операторами и функциями.

Операндами могут быть: метки, константы и переменные.

Метка может располагаться перед инструкцией или входить в директиву.

Если метка располагается перед инструкцией, то после неё ставится двоеточие (:). Метка отмечает ячейку памяти, в которой хранится строка программы. Например,

```
Lab23: mov r5,r7          ; строка в программе отмечена меткой Lab23
```

Если метка входит в директиву, то она рассматривается как один из операндов этой директивы и двоеточием не отмечается. Например,

```
.SET pina = $19           ; метка pina  с помощью директивы .SETсвязывается с числом $19
```

Константы являются целыми числами и задаются в одном из следующих форматов:

- десятичный, например, 255;
- шестнадцатеричный - начинается с символа $ или 0x, например, $0a или 0x0a;
- двоичный – начинается с символов 0b, например, 0b00001010.

Для 8-битной AVR-архитектуры минимальное значение константы равно 0, максимальное - равно 255=$ff=0xff=0b11111111.

Метке может быть присвоено конкретное значение. Для этого предназначена директива .EQU, связывающая константой с меткой. Эта метка может использоваться далее в программе. Например,

```
.EQU PINA = $19                ; метка PINA связывается с числом $19.
```

Метка, указывающая на константу в соответствии с директивой .EQU, не может быть впоследствии переопределена.

Переменные

Для присвоения переменным имен предназначена директива .SET, которая связывает метку и выражение. Метка может использоваться далее в программе вместо переменной. Например,

```
.SET PINA = $19          ; имя pina присваивается числу $19
```

Имя, присвоенное переменной в соответствии с директивой .set, может быть впоследствии изменено.

Операторы

Все операторы ассемблера перечислены в таблице 3.1.

Таблица 3.1. - Операторы языка Atmel AVR Assembler

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

Порядок выполнения операторов в выражении определяется приоритетом. Операторы с большим приоритетом выполняются в первую очередь. Кроме того, отдельные выражения могут быть заключены в круглые скобки. Такие выражения также вычисляются в первую очередь.

Функции

Язык ассемблера определяет ряд функций, которые можно использовать при написании директив (таблица 3.2)

Таблица 3.2. - Функции, определенные в языке ассемблера

low(expression)	возвращает младший байт выражения;
high(expression)	возвращает второй байт выражения;
byte2(expression)	то же что high;
byte3(expression)	возвращает третий байт выражения;
byte4(expression)	возвращает четвертый байт выражения;
lwRd(expression)	возвращает биты 0-15 выражения;
hwRd(expression)	возвращает биты 16-31 выражения;
page(expression)	возвращает биты 16-21 выражения;
exp2(expression)	возвращает 2 в степени, определяемой выражением в скобках;
log2(expression)	возвращает целую часть вычислений log2 от выражения в скобках.

Строка программы

Строка программы может иметь одну из четырёх форм:

- пустая строка
- комментарий
- [метка:] директива [операнды] [комментарий]
- [метка:] инструкция [операнды] [комментарий]

Пустая строка не несет никакой информации и используется для лучшего восприятия текста программы.

Комментарием считается любой текст, начинающийся с символа ; (точка с запятой). При написании комментария можно использовать символы кириллицы.

Позиции в квадратных скобках необязательны. Текст после точки с запятой (;) и до конца строки компилятором игнорируется.

Директивы

Директивы ассемблера не транслируются в коды операций, они используются для размещения программы в памяти, определяют макроинструкции, инициализируют память данных и выполняют ещё целый

ряд различных вспомогательных операций, облегчающих процесс программирования. Полный список директив ассемблера AVR приведен в таблице 3.3.

Таблица 3.3 - Директивы ассемблера

Директива	Описание
BYTE	зарезервировать байт под переменную
CSEG	сегмент кода
DB	задать постоянным(и) байт(ы) в памяти
DEF	задать символическое имя регистру
DEVICE	задать тип микроконтроллера
DSEG	сегмент данных
DW	задать постоянное(ые) слово(а) в памяти
EQU	установить символ равный выражению
ESEG	сегмент eeprom
EXIT	выход из файла
INCLUDE	включить код из другого файла
LIST	включить генерацию .lst - файла
NO.LIST	выключить генерацию .lst - файла
ORG	начальный адрес программы
SET	установить символ, равный выражению

.BYTE

Директива .BYTE резервирует байты в памяти данных. Директиве должна предшествовать метка.. Директива имеет один параметр, который указывает на число резервируемых байт. Директива может быть использована только в пределах сегмента данных. Например,

```
.DSEG
var1:   .BYTE 1          ; резервируется 1 байт для var1
table:  .BYTE tab_size ; резервируется tab_size байт
```

.CSEG

Директива .CSEG указывает на начало сегмента кода. Инструкции программы, размещенные после директивы .cseg при компиляции заносятся в flash-память программ микроконтроллера. Программа может иметь несколько кодовых сегментов, которые при компиляции будут объединены в один.

Например:

```
.cseg ; начало сегмента кода
ldi r30, $a0 ; инструкции
ldi r31,0b1110
```

.DSEG

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

Пример:

```
.DSEG   ; начало сегмента данных
varl:  .BYTE 1     ; резервировать 1 байт под переменную с именем varl
table: .BYTE tab_size ; резервировать tab_size байтов.
```

.ESEG

Директива .ESEG указывает на начало сегмента EEPROM памяти. Директива должна поддерживаться аппаратными средствами, позволяющими осуществить прямое программирование EEPROM в процессе прошивки. В противном случае программирование EEPROM может осуществляться только непосредственно из программы.

.ORG

Директива .ORG указывает на конкретную ячейку памяти, в которой будут размещены последующие данные. Используется только совместно с директивами .CSEG, .DSEG, .ESEG. Например:

```
.DSEG                ; начало сегмента данных
.ORG $37             ; установить адрес памяти данных на 37h
variable: .BYTE 1 ; зарезервировать 1 байт в памяти данных по адресу 37h
.CSEG
.ORG $10          ; установить программный счетчик на адрес 10h
mov r0,rl                ; инструкция программы
```

.DB

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

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

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

```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
```

.DW

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

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

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

.DEF

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

```
.DEF temp=r16
.DEF ior=r0
```

.EQU

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

```
.EQU io_offSET= 0x23
.EQU porta = io_offSET+ 2
```

.INCLUDE

Директива .INCLUDE предлагает ассемблеру начать читать из другого файла. Ассемблер будет компилировать этот файл до конца файла или до директивы EXIT. Включаемый файл может сам включать директивы .INCLUDE. Например,

```
.INCLUDE “m16DEF.inc”; включить в программу файл m16DEF.inc
```

.EXIT

Директива .EXIT позволяет ассемблеру остановить компиляцию текущего файла. Обычно ассемблер работает до конца файла. Если он

встретит директиву .EXIT, то продолжит компилировать со строки, следующей за директивой INCLUDE. Например:

```
.EXIT ; выйти из файла
```

.SET

Директива .SETприсваивает имя выражению. Это имя может использоваться далее в программе. Имя, присвоенное выражению в соответствии с директивой .SET, может быть впоследствии изменено. Например,

```
.SET pina = $19          ; имя pina присваивается числу $19
.SET porta = pina + 2    ; имя porta присваивается выражению $19+2=$1b
```

.DEVICE

Директива .DEVICE сообщает ассемблеру об используемом контроллере. Например,

```
.DEVICE atmega16      ; выбираем atmega16
```

.MACRO

Директива .MACRO сообщает ассемблеру о начале макроса - участка программы, которому присвоено имя и который компилируется всякий раз, когда ассемблер встречает это имя в тексте программы. Параметром директивы является название макроса. Макрос может содержать до 10 параметров. Эти параметры упоминаются как @0 - @9 в пределах макроопределения. При вызове макроса параметры перечисляются списком через запятую. Например,

```
.MACRO subi16         ; начало определения макроса
subi @1,low(@0)       ; вычитание младшего байта
sbci @2,high(@0)      ; вычитание старшего байта
.ENDMACRO             ; конец макроопределения
```

.ENDMACRO

Директива .ENDMACRO определяет конец макроса. Директива не имеет никаких параметров.

.LIST

Директива .LIST сообщает ассемблеру о необходимости генерации листинга программы. Листинг – это файл, содержащий исходный текст программы, адреса и коды инструкций. По умолчанию генерация листинга считается включенной. Вместе с директивой .NOLIST она позволяет генерировать листинги отдельных частей исходного файла.

.LISTMAC

Директива .LISTMAC включает в листинг программы текст макроса. По умолчанию в листинге показывается только вызов макроса с параметрами. Например,

```
.MACRO macx ; определение макроса
add  r0,@0
eor  r1,@1
.ENDMACRO          ; конец определения
.LISTMAC           ; включение макроса в листинг программы
macx r2,r1         ; выполнение макроса
NOLIST
```

.NOLIST

Директива .NOLIST выключает генерацию листинга. По умолчанию генерация листинга считается включенной. Директива может использоваться также вместе с директивой .LIST для генерации листинга отдельных частей программы. Например,

```
.NOLIST                ; отключение генерации листинга программы
.INCLUDE "MACRO.inc    ; подключение файла
.INCLUDE "const.DEF"   ; подключение файла
.LIST          ; включение генерации листинга
```

Инструкции

Инструкции ассемблера предписывают процессорному ядру выполнить определенные действия с операндами.

Адреса операндов, задействованных в выполнении любой инструкции программы, в явном или в неявном виде указываются в этой инструкции. Операнды могут находиться в регистрах общего назначения, в ячейках памяти данных, в регистрах ввода/вывода и даже в ячейках памяти программ используемого микроконтроллера. Ассемблеру об используемой микросхеме сообщает директива .DEVICE. Если какая-либо инструкция в программе не поддерживается выбранным контроллером или сегмент программы или сегмент EEPROM в программе больше, чем в выбранном устройстве, ассемблер генерирует предупреждение.

Микроконтроллер ATmega16, используемый в лабораторной работе, имеет определенный, описанный далее, набор регистров и ячеек памяти.

Регистры общего назначения

Ядро AVR имеет 32 регистра общего назначения: r0 – r31. Обращение к регистрам производится по их названиям. Кроме того, каждому регистру (ри-

сунок 3.1) соответствует дополнительно адрес в памяти данных SRAM, отображающий их в первых 32 ячейках пространства данных. Поэтому к каждому регистру можно обратиться и как к ячейке памяти.

Регистры общего назначения	SRAM	
r0	$00	
r1	$01	
r26	$1a	X-register low byte
r27	$1b	X-register high byte
r28	$1c	Y - register low byte
r27	$1d	Y - register high byte
r30	$1e	Z - register low byte
r31	$1f	Z - register high byte

Рисунок 3.1 - Файл регистров общего назначения ядра AVR

Шесть из 32 регистров (r26 …. r31) можно использовать парами. При этом они могут выполнять функцию 16-битных регистров . Эти три регистра двойной разрядности определяются как регистры X, Y и Z. При этом в четных регистрах хранятся младшие байты 16-битных переменных, а в нечетных – старшие (рисунок 3.2).

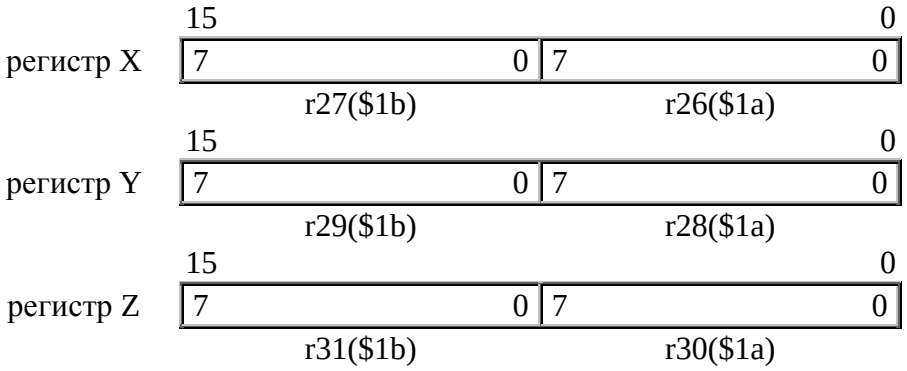


Рисунок 3.2 - Регистры косвенной адресации архитектуры AVR

Память данных

Микроконтроллер ATmega 16 имеет 1024 восьмибитные ячейки встроенной статической памяти типа SRAM. Общий объем статической памяти микроконтроллера равен 1кбайт. Сама память занимает адресное пространство $60…..$45f (рисунок 3.3). Размещенные в том же адресном пространстве 32 ячейки с адресами $0…1f используются микроконтроллером как регистры общего назначения r0 – r31, а 64 ячейки с адресами $2f…$5f – как регистры ввода/вывода.

Рисунок 3.3 - Статическая память данных микроконтроллера ATmega 16

Память программ

Флэш-память программ микроконтроллера ATmega16 имеет 8K 16-битных ячеек Flash-memory. Общий объем памяти 16 Кбайт. При этом всё адресное пространство флэш-памяти программ ($0000…$1fff) разделено на два раздела (рисунок 3.4): раздел программы начальной загрузки (Boot Program Section) и раздел прикладной программы (Application Program Section). Размер программы начальной загрузки может находиться в пределах от 256 до 2048 байт

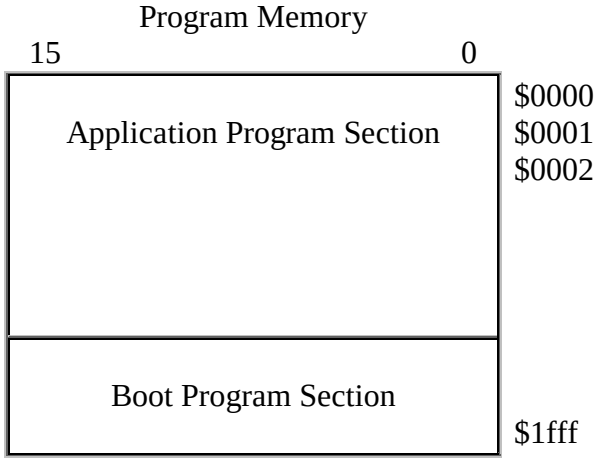


Рисунок 3.4. Организация flash-памяти программ микроконтроллера ATmega 16

Оба раздела могут быть программно заблокированы от записи. Кроме того, в системе команд микроконтроллера предусмотрена специальная инструкция spm, осуществляющая запись данных в раздел прикладной программы. Однако эта инструкция может быть использована только в разделе программы начальной загрузки.

Регистры ввода/вывода

Регистры ввода/вывода предназначены для управления различными функциональными блоками микроконтроллера, не входящими в процессорное ядро.

В микроконтроллере ATmega16 предусмотрено 64 регистра ввода/вывода (Input/output registers). Все они размещены в пространстве ввода/вывода процессорного ядра по адресам от $00 по $3f.

Одновременно регистры ввода/вывода представлены также в адресном пространстве памяти данных (рисунок 3.4) и к ним можно адресоваться как к обычным ячейкам памяти данных с адресами от $20 до $5f.

Полный список регистров ввода/вывода микроконтроллера ATmega16 приведен в приложении.

Обращение к регистрам ввода/вывода осуществляется с помощью специальных инструкций ввода/вывода как к элементам процессорного ядра по присвоенным в архитектуре адресам или именам. Для того чтобы AVR Assembler распознавал имена регистров, в программу с помощью директивы .INCLUDE необходимо подключить файл описания микроконтроллера. Все файлы описания имеют расширение inc и размещены в папке Appnotes программы AVR Studio. Файл описания микроконтроллера ATmega16 имеет имя m16def.inc.

Система команд

Система команд описывает весь набор инструкций, поддерживаемых микроконтроллером. Для микроконтроллеров с AVR архитектурой она содержит 130 инструкций, условно разделенных на четыре группы:

- инструкции пересылки данных,
- арифметические и логические инструкции,
- инструкции работы с битами,
- инструкции ветвления.

При описании инструкций использованы следующие обозначения:

- Rd - регистр-приемник результата ($0 \le d \le 31$),
- Rd* - регистр-приемник результата с номером более 16 ($16 \le d \le 31$),
- Rr - регистр-источник ($0 \le r \le 31$),
- Rdl: регистры r24, r26, r28, r30 (для инструкций adiw и sbiw),
- P- адрес регистра ввода/вывода,
- P*- адрес побитно адресуемого регистра ввода/вывода ($00-$1f)
- K8 - символьная или численная константа (0 - 255 бит)
- k - адресная константа
- b - номер бита в регистре (0 - 7)
- s - номер бита в регистре статуса (0 - 7)
- X, Y, Z - регистры косвенной адресации (X=r27:r26, Y=r29:r28; Z=r31:r30

4. ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1 «ИНСТРУКЦИИ РАБОТЫ С БИТАМИ»

Микроконтроллеры с архитектурой AVR поддерживают 28 инструкций для работы с битами (таблица 4.1). Они позволяют обращаться непосредственно к отдельным битам регистров процессорного ядра и выполнять с выбранными битами простейшие операции: пересылки, установки и сброса. Операндами команд могут быть как биты регистров общего назначения r0 – r31, так и биты регистров ввода/вывода с адресами $00 - $1f из списка, приведенного в приложении.

Большая часть битовых инструкций работает с отдельными битами регистра статуса SREG, размещенного в пространстве регистров ввода/вывода (рисунок 4.1).

Регистр	Адреса	7	6	5	4	3	2	1	0
SREG	$3F ($5F)	I	T	H	S	V	N	Z	C

Рисунок 4.1 - Регистр состояния ядра AVR

Отдельные биты этого регистра обычно именуются флагами. Они используются при выполнении большинства логических и арифметических операций, при работе системы прерываний и других функциональных блоков микроконтроллера. Флаги регистра SREG имеют следующий смысл:

- бит 7 – I (Global Interrupt Enable) - флаг глобального разрешения прерывания, используется в системе прерываний;
- бит 6 – T (Bit Copy Storage) - бит сохранения копии;
- бит 5 – H (Half Overflow) – флаг полупереноса, устанавливается при выполнении арифметических операций, при возникновении переноса из младшей тетрады 8-битного числа в старшую;
- бит 4 – S (Sign Bit) - флаг знака; определяется операцией исключающего ИЛИ между флагом отрицательного результата N и флагом переполнения V;
- бит 3 – V (Overflow) – флаг переполнения; при алгебраическом сложении двух двоичных чисел признаком переполнения является наличие переноса в знаковый разряд суммы при отсутствии переноса из её знакового разряда (положительное переполнение) или наличие переноса из знакового разряда суммы при отсутствии переноса в её знаковый разряд (отрицательное переполнение); если и в знаковый, и из знакового разряда суммы есть переносы или нет этих переносов, то флаг переполнения сбрасывается;
- бит 2 – N (Negative) - флаг отрицательного результата, устанавливается, когда результат операции является отрицательным числом; при выполнении арифметических операций равен старшему биту результата;

- бит 1 – Z (Zero Flag) - флаг нулевого результата, устанавливается, когда результат операции равен нулю;
- бит 0 – C (Carry Flag) - флаг переноса, устанавливается, если разрядность результата операции более 8 бит.

Таблица 4.1. - Битовые инструкции микроконтроллера ATmega16

Мнемо-ника	Операнды	Описание	Операция	Флаги	Такт
sbi	P*,b	Set Bit in I/O Register	I/O(P,b) <- 1	None	2
cbi	P*,b	Clear Bit in I/O Register	I/O(P,b) <- 0	None	2
lsl	Rd	Logical Shift Left	Rd(n+1) <- Rd(n), Rd(0) <- 0	Z,C,N,V	1
lsr	Rd	Logical Shift Right	Rd(n) <- Rd(n+1), Rd(7) <- 0	Z,C,N,V	1
rol	Rd	Rotate Left Through Carry	Rd(0) <- C, Rd(n+1) <- Rd(n), C <- Rd(7)	Z,C,N,V	1
ror	Rd	Rotate Right Through Carry	Rd(7) <- C,Rd(n)<- Rd(n+1),C <- Rd(0)	Z,C,N,V	1
asr	Rd	Arithmetic Shift Right	Rd(n) <- Rd(n+1), n=0..6	Z,C,N,V	1
swap	Rd	Swap Nibbles	Rd(3..0) <- Rd(7..4), Rd(7..4) <- Rd(3..0)	None	1
bset	s	Flag Set	SREG(s) <- 1 SREG(s)		1
bclr	s	Flag Clear	SREG(s) <- 0 SREG(s)		1
bld	Rd, b	Bit load from T to Register	Rd(b) <- T	None	1
bst	Rr, b	Bit Store from Register to T	T <- Rr(b)	T	1
sec		Set Carry	C <- 1	C	1
clc		Clear Carry	C <- 0	C	1
sen		Set Negative Flag	N <- 1	N	1
cln		Clear Negative Flag	N <- 0	N	1
sez		Set Zero Flag	Z <- 1	Z	1
clz		Clear Zero Flag	Z <- 0	Z	1
sei		Global Interrupt Enable	I <- 1	I	1
cli		Global Interrupt Disable	I <- 0	I	1
ses		Set Signed Test Flag	S <- 1	S	1
cls		Clear Signed Test Flag	S <- 0	S	1
sev		Set Twos Complement Over-flow	V <- 1	V	1
clv		Clear Twos Complement Over-flow	V <- 0	V	1
set		Set T in SREG	T <- 1	T	1
clt		Clear T in SREG	T <- 0	T	1
she		Set Half Carry Flag in SREG	H <- 1	H	1
clh		Clear Half Carry Flag in SREG	H <- 0	H	1
sleep		Sleep	-	-	3
wdr		Watchdog Reset	-	-	1
nop		No Operation	-	-	1

Биты регистров ввода/вывода могут быть установлены или сброшены. Никакие другие битовые операции в этих регистрах не выполняются.

Биты регистров общего назначения могут быть сдвинуты в соседние ячейки, как влево (в сторону старших разрядов), так и вправо (в сторону младших разрядов). В операциях сдвига, кроме разрядов регистров, может участвовать и бит переноса C регистра SREG. Установку или сброс отдельных разрядов регистров общего назначения можно выполнить только командой bld, копирующей бит сохранения копии T из регистра SREG. В регистрах общего назначения, кроме того, можно с помощью команды swap поменять местами тетрады (полубайты).

Программа работы

Создать новый проект в программе AVR Studio. Присвоить проекту имя. Имя должно содержать фамилию разработчика и номер лабораторной работы в английской транскрипции.

Ввести в проект следующую программу.

```
; лаб. 1, ФИО, группа, дата
.DEVICE ATmega16
.CSEG
.ORG 0
.DEF data=r16
set
bld data,1
lsl data
swap data
ror data
sbi porta,2
.EXIT
```

Разобраться с со всеми директивами и инструкциями программы. Добавить в программу комментарии, поясняющие действия программы.

Откомпилировать программу. Зафиксировать содержимое памяти программ микроконтроллера (сделать скриншот). Найти в папке с программой файл листинга программы, внимательно изучить его и объяснить результат.

Запустить программу на выполнение в шаговом режиме. Зафиксировать содержимое программного счетчика, регистров общего назначения микроконтроллера, регистра статуса SREG, регистра ввода/вывода PORTA после выполнения каждой инструкции (сделать скриншоты). Результаты объяснить.

Составить схему алгоритма и написать программу на языке ассемблера для решения задачи в соответствии с вариантом задания:

1. Записать число 0b1010 в регистр r5; записать число 0b111 в регистр DDRD; установить флаг нулевого результата в регистре статуса SREG;
2. Записать число 0b1100 в регистр r7; записать число 0b10110 в регистр DDRB; установить флаг отрицательного результата в регистре статуса SREG;
3. Записать число 0b11000 в регистр r8; записать число 0b101 в регистр UDR; установить флаг разрешения прерывания в регистре статуса SREG;
4. Записать число 0b10101 в регистр r9; записать число 0b11001 в регистр DDRD; установить флаг знака результата в регистре статуса SREG;
5. Записать число 0b1000010 в регистр r10; записать число 0b1101 в регистр EEDR; установить флаг переноса в регистре статуса SREG;
6. Записать число 0b10101 в регистр r11; записать число 0b10101 в регистр PORTC; установить флаг полупереноса в регистре статуса SREG;
7. Записать число 0b11010 в регистр r12; записать число 0b101100 в регистр ADMUX; установить флаг переполнения в регистре статуса SREG;
8. Записать число 0b11010 в регистр r5; записать число 0b11100 в регистр SPDR; установить флаг нулевого результата в регистре статуса SREG;
9. Записать число 0b1100 в регистр r7; записать число 0b101100 в регистр EEDR; установить флаг отрицательного результата в регистре статуса SREG;
10. Записать число 0b11000 в регистр r8; записать число 0b10100 в регистр ADCH; установить флаг разрешения прерывания в регистре статуса SREG;
11. Записать число 0b10101 в регистр r9; записать число 0b110010 в регистр ADMUX; установить флаг знака результата в регистре статуса SREG;
12. Записать число 0b1000010 в регистр r10; записать число 0b1101 в регистр DDRA; установить флаг переноса в регистре статуса SREG;

13. Записать число 0b10101 в регистр r11; записать число 0b1010100 в регистр PORTA; установить флаг полупереноса в регистре статуса SREG;
14. Записать число 0b11010 в регистр r12; записать число 0b101100 в регистр EEDR; установить флаг переполнения в регистре статуса SREG;

Отладить программу, добиться её выполнения.

Содержание отчета

Титульный лист. Схема алгоритма. Листинги программ. Скриншоты результатов. Таблицы результатов. Выводы.

Контрольные вопросы

- Где могут храниться операнды в битовых инструкциях?
- Как изменяется содержимое регистра при выполнении инструкции арифметического сдвига влево?
- В каких регистрах можно выполнять операции сдвига?
- Какой флажок регистра статуса участвует в выполнении некоторых операций сдвига?
- Какие битовые операции можно выполнять в регистрах общего назначения?
- Какие битовые операции можно выполнять в регистрах ввода/вывода?

Лабораторная работа 2 «АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ИНСТРУКЦИИ»

Микроконтроллеры с архитектурой AVR поддерживают 31 инструкцию для выполнения арифметических и логических операций (таблица 4.2). Операнды этих инструкций всегда хранятся в регистрах общего назначения, в один из них (регистр-приемник) всегда направляется и результат вычислений. В большинстве инструкций участвуют флаги регистра статуса SREG.

Таблица 4.2. Арифметические и логические инструкции

Мнемо-ника	Операнды	Описание	Операция	Флаги	Такт
add	Rd, Rr	Add without Carry two Registers	Rd <- Rd + Rr	Z,C,N,V,H	1
adc	Rd, Rr	Add with Carry two Registers	Rd <- Rd + Rr + c	Z,C,N,V,H	1
adiw	Rdl, k8	Add Immediate from Word	Rdh:Rdl <- Rdh:Rdl+k	Z,C,N,V,S	2
sub	Rd, Rr	Subtract without Carry two Registers	Rd <- Rd – Rr	Z,C,N,V,H	1
subi	Rd*, k8	Subtract Constant from Register	Rd <- Rd – k	Z,C,N,V,H	1
sbc	Rd, Rr	Subtract with Carry two Registers	Rd <- Rd – Rr – c	Z,C,N,V,H	1
sbci	Rd*, k8	Subtract with Carry Constant from Register	Rd <- Rd – k – c	Z,C,N,V,H	1
sbiw	Rdl, k8	Subtract Immediate from Word	Rdh:Rd <- Rdh:Rdl–k	Z,C,N,V,S	2
and	Rd, Rr	Logical AND Registers	Rd <- Rd Rr	Z,N,V	1
andi	Rd*, k8	Logical AND Register and Constant	Rd <- Rd k	Z,N,V	1
or	Rd, Rr	Logical OR Registers	Rd <- Rd v Rr	Z,N,V	1
ori	Rd*, k8	Logical OR Register and Constant	Rd <- Rd v k	Z,N,V	1
eor	Rd, Rr	Exclusive OR Registers	Rd <- Rd Rr	Z,N,V	1
com	Rd	One’s Complement	Rd <- $ff – Rd	Z,C,N,V	1
neg	Rd	Two’s Complement	Rd <- $00 – Rd	Z,C,N,V,H	1
sbr	Rd*, k	SETBit(s) in Register	Rd <- Rd v k	Z,N,V	1
ser	Rd	SETRegister	Rd <- $ff	None	1
cbr	Rd*, k	Clear Bit(s) in Register	Rd <- Rd ($ff – k)	Z,N,V	1
clr	Rd	Clear Register	Rd <- $00	Z,N,V	1
inc	Rd	Increment	Rd <- Rd + 1	Z,N,V	1
dec	Rd	Decrement	Rd <- Rd –1	Z,N,V	1
tst	Rd	Test for Zero or Minus	Rd <- Rd Rd	Z,N,V	1
cp	Rd, Rr	Compare	Rd – Rr	Z, N,V,C,H	1
cpc	Rd, Rr	Compare with Carry	Rd – Rr – c	Z, N,V,C,H	1
cpi	Rd*, k8	Compare Register with Immediate	Rd – k	Z, N,V,C,H	1
mul	Rd, Rr	Multiply Unsigned	r1:r0 <- Rd x Rr	Z,C	2
muls	Rd, Rr	Multiply Signed	r1:r0 <- Rd x Rr	Z,C	2
mulsu	Rd, Rr	Multiply Signed with Unsigned	r1:r0 <- Rd x Rr	Z,C	2
fmul	Rd, Rr	Fractional Multiply Unsigned	r1:r0<-(Rd x Rr)<<1	Z,C	2
fmuls	Rd, Rr	Fractional Multiply Signed	r1:r0<-(Rd x Rr)<<1	Z,C	2
fmulsu	Rd, Rr	Fractional Multiply Signed with Unsigned	r1:r0<-(Rd x Rr)<<1	Z,C	2

Программа работы

Создать новый проект в программе AVR Studio. Присвоить проекту имя. Имя должно содержать фамилию разработчика и номер лабораторной работы в английской транскрипции.

Ввести в проект следующую программу.

```
; лаб. 1, ФИО, группа, дата
.DEVICE ATmega16
.CSEG
.ORG 0
.DEF data1=r16
.DEF data2=r17
.SET a=10; вместо $10 занести заданное число a
.SET b=12; вместо $12 занести заданное число b
.MACRO ab
ldi data1,a
ldi data2,b
.ENDMACRO

ab
add data1,data2
nop
.EXIT
```

Разобраться с со всеми директивами и инструкциями программы. Добавить в программу комментарии, поясняющие действия программы. Занести в программу числа a = 8xN и b = 260 - 7xN в соответствии с вариантом задания N.

Запустить программу на выполнение в шаговом режиме. Зафиксировать содержимое программного счетчика, регистров общего назначения, микроконтроллера и регистра статуса после выполнения инструкции add (сделать скриншоты). Результаты объяснить.

На основе существующей разработать новую программу, которая последовательно выполняет с теми же данными a и b следующие операции:

- вычитание a-b.
- вычитание b-a,
- логическое умножение a и b,
- логическое сложение a и b ,
- исключающее ИЛИ a и b,
- инверсию a,
- смену знака числа a,
- умножение целых чисел a и b.

Отладить программу. Запустить её на выполнение в шаговом режиме.

Занести в таблицу 4.3 результаты, полученные в ходе выполнения отдельных команд программы: содержимое регистра-приемника и регистра статуса.

Таблица 4.3 – результаты вычислений

Операция	Результат	SREG (I T H S V N Z C)
вычитание a-b.		
вычитание b-a,		
логическое умножение a и b,		
логическое сложение a и b ,		
исключающее или a и b,		
инверсия a,		
смена знака числа a,		
умножение целых чисел a и b.		

Объяснить полученные результаты.

Содержание отчета

Титульный лист. Листинги программ. Скриншоты результатов. Таблицы результатов. Выводы.

Контрольные вопросы

- Для чего предназначены директивы ассемблера?
- Для чего предназначены инструкции ассемблера?
- Какие виды меток используются в ассемблере?
- Что может являться операндом в директиве ассемблера?
- Какие форматы используются для представления констант?
- Сколько регистров общего назначения имеют микроконтроллеры с AVR архитектурой?
- В каких случаях устанавливаются флаги регистра статуса?
- Какие флаги входят в регистр статуса микроконтроллера?
- Какое максимальное число может быть записано в регистр общего назначения микроконтроллера с AVR архитектурой?

Лабораторная работа 3 «ИНСТРУКЦИИ ПЕРЕСЫЛКИ ДАННЫХ»

Инструкции пересылки осуществляют перемещение данных между ячейками памяти и регистрами процессорного ядра. В качестве операндов в инструкциях пересылки могут использоваться:

- регистры общего назначения,
- ячейки статической памяти данных (Data Memory),
- регистры ввода вывода,
- ячейки памяти программ (Program Memory).

Пересылка осуществляется от источника к приемнику, при этом копия данных всегда остается в источнике. Т.е. все инструкции пересылки практически осуществляют копирование данных.

При адресации памяти данных в микроконтроллере может быть задействован стек – область памяти, организованная по принципу: последний зашел – первый вышел. Запись в стек и извлечение из стека не требует знания адресов ячеек памяти, в которые записываются данные.

Стек организуется в памяти данных SRAM и для работы с ним в процессорном ядре предусматривается специальный регистр указатель стека – SP (Stack Pointer). Указатель стека хранит адрес последней записанной ячейки памяти в области стека. При записи данных в стек содержимое указателя стека уменьшается на 1, а при чтении из стека – увеличивается на 1.

Микроконтроллер ATmega16 оснащен 16-разрядным указателем стека, размещенным в двух регистрах SPH (Stack Pointer High) и SPL (Stack Pointer Low) пространства ввода/вывода по адресам $3e ($5e) и $3d ($5d). Поскольку рассматриваемый микроконтроллер имеет SRAM с адресами от $000 до $045f (рисунок 2.3), то в нем используется только 11 бит указателя стека SP0….SP10 (рисунок 4.2).

		7	6	5	4	3	2	1	0
SPH	$3e ($5e)						SP10	SP9	SP8
SPL	$3d ($5d)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0

Рисунок 4.2 - Регистры указателя стека микроконтроллера ATmega16

Начальное значение указателя стека должно задаваться в программе до первого обращения к стеку. Обычно стек размещают в последних ячейках памяти данных. У микроконтроллера ATmega16 последняя ячейка SRAM имеет адрес $45f, а в файл описания микроконтроллера «m16def.inc» включена директива

```
.equ    RAMEND =$45F.
```

Поэтому в начале пользовательской программы рекомендуется выполнить загрузку указателя стека с помощью следующего набора инструкций:

```
ldi r16,high(RAMEND)
out SPH,r16 ; Set Stack Pointer to top of RAM
ldi r16,low(RAMEND)
out SPL,r16
```

В системе команд микроконтроллеров AVR предусмотрено 34 инструкции, осуществляющие пересылку данных (таблица 4.4). Время выполнения инструкций, работающих с регистрами, равно 1 такту. Инструкции, обращающиеся к ячейкам памяти данных, выполняются за 2 такта, а обращающиеся к ячейкам памяти программ – за 3 такта.

Таблица 4.4 - Инструкции пересылки AVR-микроконтроллеров

Мнемоника	операнды	Описание инструкции	выполняемая операция	Такты
mov	Rd, Rr	Move Between Registers	Rd <- Rr	1
movw	Rd, Rr	Copy Register WoRd	Rd+1:Rd <- Rr+1:Rr	1
ldi	Rd*, k	Load Immediate	Rd <- k	1
ld	Rd, x	Load Indirect	Rd <- (x)	2
ld	Rd, x+	Load Indirect and Post-Inc.	Rd <- (x), x <- x + 1	2
ld	Rd, - x	Load Indirect and Pre-Dec.	x <- x - 1, Rd <- (x)	2
ld	Rd, y	Load Indirect	Rd <- (y)	2
ld	Rd, y+	Load Indirect and Post-Inc.	Rd <- (y), y <- y + 1	2
ld	Rd, - y	Load Indirect and Pre-Dec.	y <- y - 1, Rd <- (y)	2
ldd	Rd, y+q	Load Indirect with Displacement	Rd <- (y + q)	2
ld	Rd, z	Load Indirect	Rd <- (z)	2
ld	Rd, z+	Load Indirect and Post-Inc.	Rd <- (z), z <- z+1	2
ld	Rd, -z	Load Indirect and Pre-Dec	z <- z - 1, Rd <- (z)	2
ldd	Rd, z+q	Load Indirect with Displacement	Rd <- (z + q)	2
lds	Rd, k	Load Direct from SRAM	Rd <- (k)	2
ST	x, Rr	Store Indirect	(x) <- Rr	2
st	x+, Rr	Store Indirect and Post-Inc.	(x) <- Rr, x <- x + 1	2
st	- x, Rr	Store Indirect and Pre-Dec.	x <- x - 1, (x) <- Rr	2
st	y, Rr	Store Indirect	(y) <- Rr	2
st	y+, Rr	Store Indirect and Post-Inc.	(y) <- Rr, y <- y + 1	2
st	- y, Rr	Store Indirect and Pre-Dec.	y <- y - 1, (y) <- Rr	2
std	y+q,Rr	Store Indirect with Displacement	(y + q) <- Rr	2
st	z, Rr	Store Indirect	(z) <- Rr	2
st	z+, Rr	Store Indirect and Post-Inc.	(z) <- Rr, z <- z + 1	2
st	-z, Rr	Store Indirect and Pre-Dec	z <- z - 1, (z) <- Rr	2
std	z+q,Rr	Store Indirect with Displacement	(z + q) <- Rr	2
sts	k, Rr	Store Direct to SRAM	(k) <- Rr	2
lpm		Load Program Memory	r0 <- (z)	3
lpm	Rd, z	Load Program Memory	Rd <- z)	3
lpm	Rd, z+	Load Program Memory and Post-Inc.	Rd <- z), z=z+1	3
spm		Store Program Memory	(z) <- r1:r0	-
in	Rd, p	In Port	Rd <- p	1
out	p, Rr	Out Port	p <- Rr	1
push	Rr	Push Register on Stack	stack <- Rr; sp<- sp-1	2
pop	Rd	Pop Register from Stack	sp<- sp+1, Rd <- stack	2

Программа работы

Составить схему алгоритма и программу на языке ассемблера для решения задачи в соответствии с вариантом задания. Отладить программу, добиться её выполнения.

Варианты заданий:

1. Переписать данные из $10 ячеек памяти программ, начиная с адреса $30, в ячейки памяти данных, начиная с адреса $100.
2. Переписать данные из 10 ячеек памяти программ, начиная с адреса 30, в ячейки памяти данных, начиная с адреса 100. Данные расположить в обратном порядке.
3. Переписать данные из $10 ячеек памяти данных, начиная с адреса $60, в ячейки памяти данных, начиная с адреса $100.
4. Переписать данные из 10 ячеек памяти данных, начиная с адреса 300, в ячейки памяти данных, начиная с адреса 100. Данные расположить в обратном порядке.
5. Занести в $10 ячеек стека последовательность чисел $1,$2,$3……. Данные из стека в обратном порядке перенести в память данных, начиная с адреса $100.
6. Занести в 10 ячеек стека последовательность чисел 1,2,3……. Данные из стека в том же порядке перенести в память данных, начиная с адреса 100.
7. Занести в $12 ячеек памяти данных, начиная с адреса 120, убывающую последовательность чисел, $10, $11…. Одновременно все данные в обратном порядке записать в стек.
8. Расположить в обратном порядке данные в 15 ячейках памяти данных, начиная с адреса $250. При решении использовать инструкции с косвенной адресацией.
9. Расположить в обратном порядке данные в 15 ячейках памяти данных, начиная с адреса $250. При решении использовать стек.
10. Поменять местами 10 байт данных в двух областях памяти данных, начинающихся с адресов $60 и $160.
11. Заполнить область памяти данных с адресами $100 - $120 последовательностью четных десятичных чисел. При выполнении задачи использовать инструкции с косвенной адресацией.
12. Заполнить область памяти данных с адресами $200 - $220 убывающей последовательностью шестнадцатеричных чисел, начиная с числа $120. При выполнении задачи использовать стек.
13. Переписать данные из 10 ячеек памяти программ, начиная с адреса $300, в ячейки памяти данных, начиная с адреса $100. При решении задачи использовать стек.

Содержание отчета

Титульный лист. Схема алгоритма. Листинги программ. Скриншоты результатов. Таблицы результатов. Выводы.

Контрольные вопросы

- Какие регистры общего назначения можно использовать в инструкциях с непосредственной адресацией??
- Как влияют инструкции пересылки на содержимое регистра статуса?
- В каких регистрах хранится адрес ячейки памяти при использовании инструкций с косвенной адресацией?
- Как изменяется содержимое регистра косвенной адресации при выполнении инструкции с предекрементом/с постинкрементом?
- Можно ли одной инструкцией скопировать данные из одной ячейки памяти данных в другую?
- Где можно использовать инструкцию записи данных в память программ?
- Как выполняются инструкции с косвенной адресацией данных? Какие регистры используются для косвенной адресации данных?

Лабораторная работа 4 «ИНСТРУКЦИИ ВЕТВЛЕНИЯ»

Инструкции ветвления изменяют содержимое программного счетчика. Они используются при организации переходов и подпрограмм.

Инструкции переходов изменяют содержимое программного счетчика. Они могут быть безусловными и условными. Инструкции условных переходов в процессе исполнения проверяют выполнение различных условий. Если условие не выполняется, программный счетчик просто инкрементируется. Такими условиями может быть равенство содержимого двух регистров, единичное или нулевое состояние заданных битов в регистрах общего назначения, регистрах ввода/вывода или в регистре состояния. Например, команда cpse (compare, skip if equal) проверяет равенство содержимого регистров, команда sbrc Rr, b (skip if bit in register cleared) осуществляет переход при условии равенства нулю содержимого бита b в регистре Rr, а инструкция brmi k (branch if minus) проверяет флаг отрицательного результата (N) в регистре состояния SREG.

Инструкции вызова подпрограмм изменяют содержимое программного счетчика, но при этом его предыдущее значение сохраняется в стеке. В конце подпрограммы обязательно должна быть инструкция reti, восстанавливающая содержимое программного счетчика из стека. При этом процессорное ядро

возвращается к решению отложенной задачи. Все инструкции вызова подпрограмм безусловные.

Время выполнения команд ветвления зависит от результата проверки. Оно для различных инструкций может меняться в пределах от одного до четырех тактов.

Все инструкции ветвления AVR-микроконтроллеров приведены в таблице 4.5.

Таблица 4.4 - Инструкции ветвления микроконтроллера ATmega16

Мнемоника	Операнды	Описание	Операция	Такт
rjmp	k	Relative Jump	PC <- PC + k + 1	2
ijmp		Indirect Jump to (Z)	PC <- Z	2
jmp	k	Jump	PC <- k	3
rcall	k	Relative Subroutine Call	PC <- PC + k + 1	3
call	k	Call Subroutine	PC <- k	4
icall		Indirect Call to (Z)	PC <- Z	3
ret		Subroutine Return	PC <- STACK	4
reti		Interrupt Return	PC <- STACK	4
cpse	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC <- PC + 2 or 3	1/ 2 / 3
sbrc	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC <- PC + 2 or 3	1 / 2
sbrs	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC <- PC + 2 or 3	1 / 2
sbic	P*, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC <- PC + 2 or 3	1 / 2
sbis	P*, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC <- PC + 2 or 3	1 / 2
brbs	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC <- PC+k + 1	1 / 2
brbc	s, k	Branch if Status Flag Cleared	if(SREG(s) = 0) then PC <- PC+k + 1	1 / 2
breq	k	Branch if Equal	if (Z = 1) then PC <- PC + k + 1	1 / 2
brcs	k	Branch if Carry Set	if (C = 1) then PC <- PC + k + 1	1 / 2
brne	k	Branch if Not Equal	if (Z = 0) then PC <- PC + k + 1	1 / 2
brcc	k	Branch if Carry Cleared	if (C = 0) then PC <- PC + k + 1	1 / 2
brsh	k	Branch if Same or Higher	if (C = 0) then PC <- PC + k + 1	1 / 2
brlo	k	Branch if Lower	if (C = 1) then PC <- PC + k + 1	1 / 2
brmi	k	Branch if Minus	if (N = 1) then PC <- PC + k + 1	1 / 2
brpl	k	Branch if Plus	if (N = 0) then PC <- PC + k + 1	1 / 2
brge	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC <- PC + k + 1	1 / 2
brlt	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC <- PC + k + 1	1 / 2
brhs	k	Branch if Half Carry Flag Set	if (H = 1) then PC <- PC + k + 1	1 / 2
brhc	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC <- PC + k + 1	
brts	k	Branch if T Flag Set	if (T = 1) then PC <- PC + k + 1	1 / 2
brtc	k	Branch if T Flag Cleared	if (T = 0) then PC <- PC + k + 1	1 / 2
brvs	k	Branch if Overflow Flag is Set	if (V = 1) then PC <- PC + k + 1	1 / 2
brvc	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC <- PC + k + 1	1 / 2
brie	k	Branch if Interrupt Enabled	if (I = 1) then PC <- PC + k + 1	1 / 2
brid	k	Branch if Interrupt Disabled	if (I = 0) then PC <- PC + k + 1	1 / 2

Программа работы

Составить схему алгоритма и программу на языке ассемблера для решения задачи в соответствии с вариантом.

1. Организовать временную задержку на 10 мкс с при тактовой частоте процессорного ядра 10 МГц;
2. Переписать данные из $100 ячеек памяти программ, начиная с адреса $30, в ячейки памяти данных, начиная с адреса $100.
3. Переписать данные из 100 ячеек памяти программ, начиная с адреса 30, в ячейки памяти данных, начиная с адреса 100. Данные расположить в обратном порядке.
4. Переписать данные из $70 ячеек памяти данных, начиная с адреса $60, в ячейки памяти данных, начиная с адреса $120.
5. Переписать данные из 60 ячеек памяти данных, начиная с адреса 300, в ячейки памяти данных, начиная с адреса 100. Данные расположить в обратном порядке.
6. Занести в $30 ячеек стека последовательность чисел $1,$2,$3……. Данные из стека в обратном порядке перенести в память данных, начиная с адреса $100.
7. Занести в 20 ячеек стека последовательность чисел 1,2,3……. Данные из стека в том же порядке перенести в память данных, начиная с адреса 100.
8. Занести в $50 ячеек памяти данных, начиная с адреса $120, убывающую последовательность чисел, $100, $101…. Одновременно все данные в обратном порядке записать в стек.
9. Расположить в обратном порядке данные в 30 ячейках памяти данных, начиная с адреса $250. При решении использовать инструкции с косвенной адресацией.
10. Расположить в обратном порядке данные в 45 ячейках памяти данных, начиная с адреса $250. При решении использовать стек.
11. Поменять местами 50 байт данных в двух областях памяти данных, начинающихся с адресов $60 и $160.
12. Заполнить область памяти данных с адресами $100 - $200 последовательностью четных десятичных чисел. При выполнении задачи использовать инструкции с косвенной адресацией.
13. Заполнить область памяти данных с адресами $200 - $300 убывающей последовательностью шестнадцатеричных чисел, начиная с числа $120. При выполнении задачи использовать стек.

14. Переписать данные из 30 ячеек памяти программ, начиная с адреса $300, в ячейки памяти данных, начиная с адреса $100. При решении задачи использовать стек.

Содержание отчета

Титульный лист. Схема алгоритма. Листинги программ. Скриншоты результатов. Таблицы результатов. Выводы.

Контрольные вопросы

- Как изменяется содержимое программного счетчика при выполнении инструкции jmp?
- Как изменяется содержимое программного счетчика при выполнении инструкции call?
- Как изменяется содержимое программного счетчика при выполнении инструкции ret?
- Какие флаги регистра статуса задействуются в командах условных переходов?
- Как изменяется содержимое указателя стека при выполнении инструкции jmp?
- Как изменяется содержимое указателя стека при выполнении инструкции call?
- Как изменяется содержимое указателя стека при выполнении инструкции ret?
- Какие регистры могут использоваться для задания условий переходов?

Библиографический список

1. Водовозов, А.М. Микроконтроллеры для систем автоматики: учебное пособие/ А.М.Водовозов.- Вологда: ВоГТУ, 2002. - 124 с.
2. Баранов, В.Н. Применение микроконтроллеров AVR. Схемы, алгоритмы, программы/ В.Н. Баранов. - М.: Додека XXI, 2006. – 288 с.
3. Ревич, Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера/ Ю.В.Ревич. - С.Пб.: БХВ-Петербург, 2011.- 352 с.
4. Гребнев, В.В. Микроконтроллеры семейства AVR фирмы Atmel/ В.В. Гребнев. - М.: РадиоСофт, 2002. - 174 с.

ПРИЛОЖЕНИЕ

Регистры ввода/вывода микроконтроллера ATmega16

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	–	–	–	–	–	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register							
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
$35 ($55)	MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)							
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register							
	OCDR	On-Chip Debug Register							
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte							
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)							
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register							
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
$1F ($3F)	EEARH	–	–	–	–	–	–	–	EEAR8
$1E ($3E)	EEARL	EEPROM Address Register Low Byte							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR	USART I/O Data Register							
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte							
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
$05 ($25)	ADCH	ADC Data Register High Byte							
$04 ($24)	ADCL	ADC Data Register Low Byte							
$03 ($23)	TWDR	Two-wire Serial Interface Data Register							
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register							

СОДЕРЖАНИЕ

Подписано в печать 6.03.2014.	Усл. печ. л. 2,44	Тираж экз.
Печать офсетная.	Бумага писчая.	Заказ №____.

Отпечатано: РИО ВоГУ, г. Вологда, ул. Ленина, 15