Лекция №1 «Введение МК серии AVR»

## I Общие сведения о МК серии AVR

История компании Atmel начинается в 1984 году, когда Джон и Гюст Прелегос покидают компанию Seeq и создают на личные средства компанию, полное название которой звучит как Advanced Technology Memory and Logic, или сокращенно – Atmel. Первоначально продукцией компании были микросхемы энергонезависимой памяти всех разновидностей, как OTP EPROM так и EEPROM. В 1995 г. Два студента Норвежского университета науки и технологий в г. Тронхейме, Альф Боген и Вегард Воллен, выдвинули идею 8-разрядного RISC-ядра, которую предложили руководству Atmel. Идея понравилась руководству компании, и в 1996 году был основан исследовательский центр, а в конце года выпущен первый МК из серии AVR под названием AT90S1200. Во второй половине 1997г. Начинается серийное производство МК семейства AVR.

У AVR-контроллеров «с рождения» есть две особенности, которые отличают это семейство от остальных МК. Во-первых, система команд и архитектура AVR разрабатывалась совместно с фирмой-разработчиком компиляторов с языков программирования IAR Systems. В результате программы, написанные на языке С, практически не проигрывают в производительности программам, написанным на ассемблере.

Во-вторых, одним из существенных преимуществ AVR стало применение конвейера. В результате большинство команд выполняется за один такт.

Так же можно выделить такие особенности как:

- Производительность порядка 1MIPS/МГц
- Усовершенствованная RISC-архитектура – наличие простейшего двухступенчатого конвейера, наличие функций аппаратного умножения в подсемействе Mega.
- Разделенные шины данных и команд
- 32 регистра общего назначение (РОН).
- Flash – память программ
- Отдельная область энергонезависимой памяти (EEPROM)
- Встроенные устройства для обработки аналоговых сигналов
- Сторожевой таймер
- Последовательные интерфейсы SPI, TWI($I^2C$) и UART(USART)
- Таймеры счетчики

- Возможность работы в широком диапазоне частот от 0 Гц до 16-20Гц
- Диапазон напряжения питания от 2.7 до 5.5 В (в некоторых случаях от 1.8 или до 6.0 В)
- Многочисленные режимы энергосбережения
- Встроенный монитор питания

Общее устройство МК AVR представлено на рисунке 1.1.

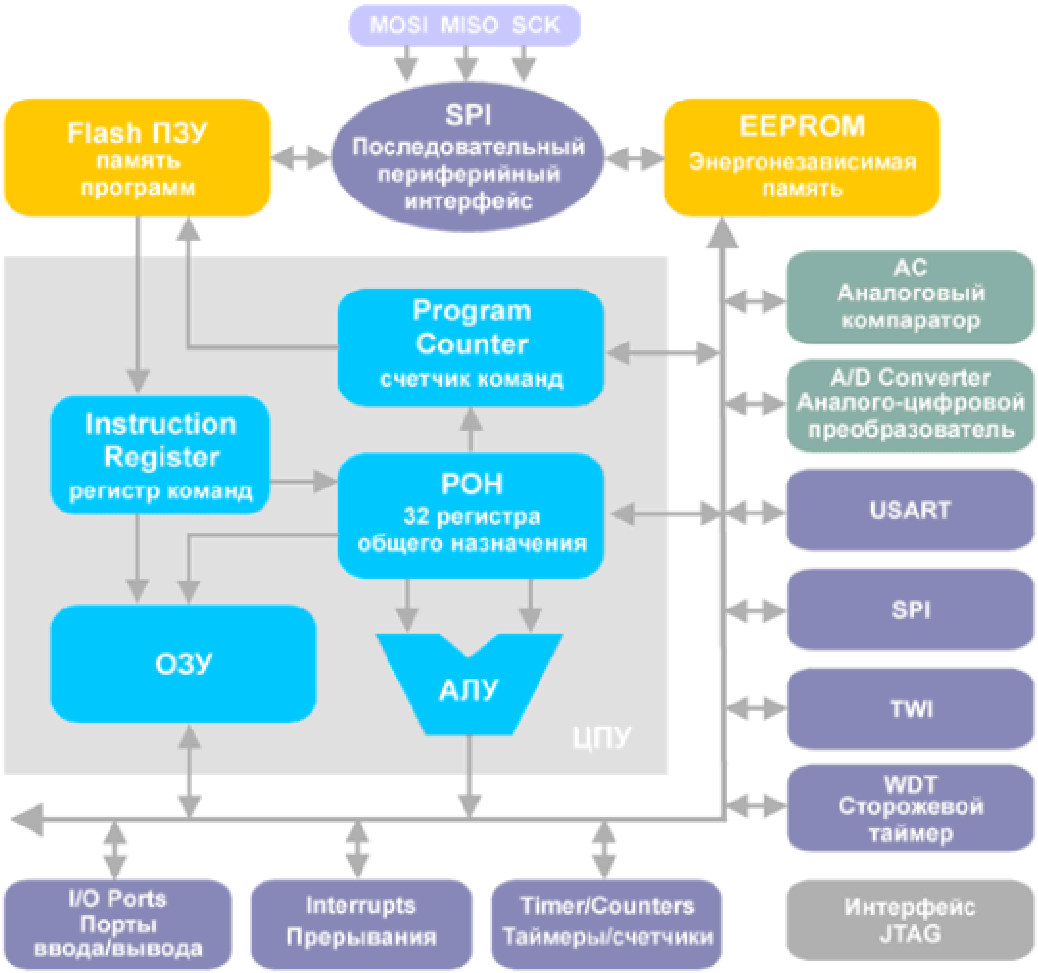



Рисунок 1.1. Общее устройство микроконтроллера

В рамках единой базовой архитектуры МК AVR делят на три семейства:

- Classic AVR
- Mega AVR
- Tiny AVR

Микроконтроллеры серии Classic уже не выпускаются. Поэтому рассмотрим семейства Tiny и Mega.

МК семейства Tiny имеют небольшие объемы памяти программ (1…2 Кбайта) и весьма ограниченную периферию. Практически все они выпускаются в 8-выводных корпусах и предназначены для создания интеллектуальных датчиков различного назначения, игрушки, зарядные устройства, различная бытовая техника и другие подобные устройства.

Микроконтроллеры семейства Mega, напротив, имеют наиболее развитую периферию, наибольшие среди всех МК AVR объемы памяти программ и данных. Они предназначены для использования в мобильных телефонах, контроллерах различного периферийного оборудования (сканеры, принтеры и т.п.), сложной офисной технике.

## II Архитектура и характеристики базовых моделей отдельных подгрупп

### 1.3. Характеристики процессора

Основными характеристиками центрального процессора микроконтроллеров рассматриваемого семейства являются:

- полностью статическая архитектура; минимальная тактовая частота равна нулю;
- АЛУ подключено непосредственно к регистрам общего назначения;
- большинство команд выполняются за один машинный цикл;
- многоуровневая система прерываний; поддержка очереди прерываний;
- 5...8 источников прерываний (из них до 2-х внешних);
- трехуровневый аппаратный стек.

### 1.4. Характеристики подсистемы ввода/вывода

Основными характеристиками подсистемы ввода/вывода являются:

- программное конфигурирование и выбор портов ввода/вывода;
- выводы могут быть запрограммированы как входные или как выходные независимо друг от друга;
- входные буферы с триггером Шмидта на всех выводах;
- возможность подключения к входам внутренних подтягивающих резисторов (сопротивление резисторов составляет 35... 120 кОм).

### 1.5. Периферийные устройства

Набор периферийных устройств, имеющихся в составе того или иного микроконтроллера, зависит от конкретной модели и может быть определен по сводной таблице, приведенной в Приложении 1. Вообще же в составе микроконтроллеров семейства встречаются следующие периферийные устройства:

- 8-разрядный таймер/счетчик с предделителем (таймер T0)*;
- второй 8-разрядный таймер/счетчик с предделителем (таймер T1)**;
- сторожевой таймер WDT;
- одноканальный генератор сигнала с ШИМ разрядностью 8 бит (один из режимов работы таймера T1);
- аналоговый компаратор;
- 10-разрядный АЦП (4 канала);
- аппаратный модулятор.

### 1.6. Архитектура ядра

Ядро микроконтроллеров AVR семейства Tiny выполнено по усовершенствованной RISC (enhanced RISC) архитектуре **(Рис.** 1.1), в которой используется ряд решений, направленных на повышение быстродействия микроконтроллеров.

Арифметико-логическое устройство (АЛУ), выполняющее все вычисления, подключено непосредственно к 32-м рабочим регистрам, объединенным в регистровый файл. Благодаря этому АЛУ выполняет одну операцию (чтение содержимого регистров, выполнение операции и запись результата обратно в регистровый файл) за один машинный цикл. Кроме того, в микроконтроллерах семейства Tiny каждая из команд занимает только одну ячейку памяти программ.

В микроконтроллерах AVR реализована Гарвардская архитектура, которая характеризуется раздельной памятью программ и данных, каждая из которых имеет собственные шины доступа к ним. 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, а также реализовать конвейеризацию. Конвейеризация заключается в том, что во время исполнения текущей команды производится выборка из памяти и дешифрация кода следующей команды.

В отличие от RISC-микроконтроллеров других фирм, в микроконтроллерах AVR используется 2-уровневый конвейер, а длительность машинного цикла составляет всего один период кварцевого резонатора. В результате, при более низкой тактовой частоте они могут обеспечивать ту же производительность, что и RISC-микроконтроллеры других фирм.

### 1.7. Цоколевка и описание выводов

В семейство Tiny входит в общей сложности 8 моделей микроконтроллеров, которые составляют 4 группы:

- ATtiny1 1, ATtiny 11L **(Рис.** 1.2) имеют FLASH-память программ объемом 1 Кбайт. Максимальное количество контактов ввода/вывода равно 6 (вывод 1 может использоваться только как входной);
- ATtiny12, ATtiny12L, ATtiny12V **(Рис.** 1.3) имеют FLASH-память программ объемом 1 Кбайт и EEPROM-память данных объемом 64 байт. Максимальное количество контактов ввода/вывода равно 6;
- ATtiny15L **(Рис.** 1.4) имеет FLASH-память программ объемом 1 Кбайт и EEPROM-память данных объемом 64 байт. Максимальное количество контактов ввода/вывода равно 6;
- ATtiny28L, ATtiny28V **(Рис.** 1.5) имеют FLASH-память программ 2 Кбайт. Количество контактов ввода/вывода равно 19 (из них 11 — контакты ввода/вывода общего назначения, а 8 — входные контакты).

Пока книга готовилась к печати, фирма «Atmel» выпустила еще две модели микроконтроллеров семейства — ATtiny26/ATtiny26L. Эти микроконтроллеры имеют FLASH-память программ объемом 2 Кбайт, ОЗУ объемом 128 байт и EEPROM-память данных объемом 128 байт. Количество контактов ввода/вывода в этих моделях равно 16. В данной книге эти модели по понятным причинам не описываются, однако основные их параметры приведены в Приложении 1.

Для сравнения в Табл. 1.1 приводятся основные параметры микроконтроллеров, такие, как объем памяти (программ и данных), количество контактов ввода/вывода, тип корпуса, диапазон рабочих частот и напряжение питания. Полная информация по каждой модели приведена в Приложении 1. Дополнительно следует отметить, что все микроконтроллеры семейства Tiny выпускаются как в коммерческом (диапазон рабочих температур 0...+70°C), так и в промышленном (диапазон рабочих температур — 40...+85°C) исполнениях.

*Таблица 1.1.* Основные параметры микроконтроллеров AYR семейства Tiny

| Обозначение | Память программ (FLASH) [Кбайт] | Память данных (EEPROM) [байт] | Количество линий ввода/ вывода | Напряжение питания[В] | Тактовая частота [МГц] | Тип корпуса |
|---|---|---|---|---|---|---|
| ATtinyc11 | 1 | - | 6 | 4.0...5.5 | 0...6 | DIP-8<br>SOIC-8 |
| ATtiny11L | 1 | - | 6 | 2.7... 5.5 | 0...2 | DIP-8<br>SOIC-8 |
| ATtiny12 | 1 | 64 | 6 | 4.0...5.5 | 0...8 | DIP-8<br>S01C-8 |
| ATtiny12L | 1 | 64 | 6 | 2.7... 5.5 | 0...4 | DIP-8<br>SOIC-8 |
| ATtiny12V | 1 | 64 | 6 | 18 55 | 0...1.2 | DIP-8<br>SOIC-8 |
| ATtiny15L | 1 | 64 | 6 | 2.7...5.5 | 0...1.2 | DIP-8<br>SOIC-8 |
| ATtiny28L | 2 | - | 19 | 2.7...5.5 | 0...4 | DIP-28<br>TQFP-32<br>MLF-32 |
| ATtiny28V | 2 | - | 19 | 18…55 | 0...1.2 | DIP-28<br>TQFP-32<br>MLF-32 |

В таблицах использованы следующие обозначения:

I - вход;
O - выход; I/O - вход/выход;
P - выводы питания.

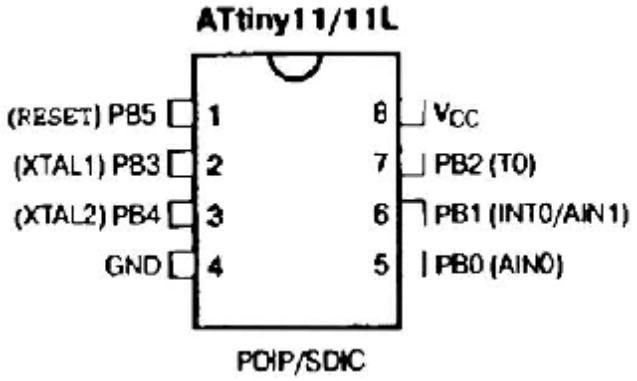



*Рис. 1.2.* Расположение выводов (вид сверху) моделей ATiny11/11L

*Таблица 1.2.* Описание выводов модели ATtiny11/11L

| Обозначение | Номер вывода | Тип вывода | Описание |
|---|---|---|---|
| PB0(AIN0) | 5 | I/O | 0-й разряд порта B (Положительный вход компаратора) |
| PBI (INT0/AIN1) | 6 | I/O | 1-й разряд порта B (Вход внешнего прерывания/ Отрицательный вход компаратора) |
| B2(T0) | 7 | 1/0 | 2-й разряд порта B (Вход внешнего тактового сигнала таймера/счетчика TO) |
| PB3(XTAL1) | 2 | I/O | 3-й разряд порта B (Вход тактового генератора) |
| PB4(XTAL2) | 3 | I/O | 4-й разряд порта B (Выход тактового генератора) |
| PB5(RESET) | 1 | I | 5-й разряд порта B (Вход сброса) |
| GND | 4 | P | Общий вывод |
| Vcc | 8 | P | Вывод источника питания |

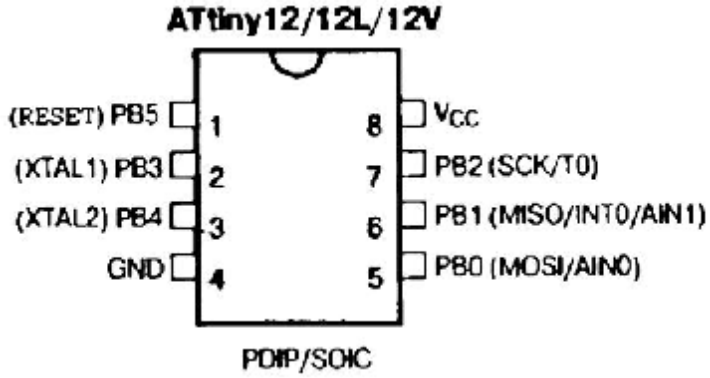



*Рис. 1.3.* Расположение выводов (вид сверху) моделей ATiny12/12L/12V

*Таблица 1.3.* Описание выводов модели ATtiny12/12L/12V

| Обозначение | Номер вывода | Тип вывода | Описание |
|---|---|---|---|
| PB0(MOSI/AIN0) | 5 | I/O | 0-й разряд порта B (Вход данных при программировании/Положительный вход компаратора) |
| PB1 (MISO/INT0/ AIN1) | 6 | I/O | 1-й разряд порта B (Выход данных при программировании/Вход внешнего прерывания/ Отрицательный вход компаратора) |
| PB2 (SCK/T0) | 7 | I/O | 2-й разряд порта B (Вход тактового сигнала при программировании/Вход внешнего тактового сигнала таймера/счетчика TO) |
| PB3 (XTAL1) | 2 | I/O | 3-й разряд порта B (Вход тактового генератора) |
| PB4 (XTAL2) | 3 | I/O | 4-й разряд порта B (Выход тактового генератора) |
| PB5 (RESET) | 1 | I/O | 5-й разряд порта B, тип выхода — открытый коллектор (Вход сброса) |
| GND | 4 | P | Общий вывод |
| Vcc | 8 | P | Вывод источника питания |

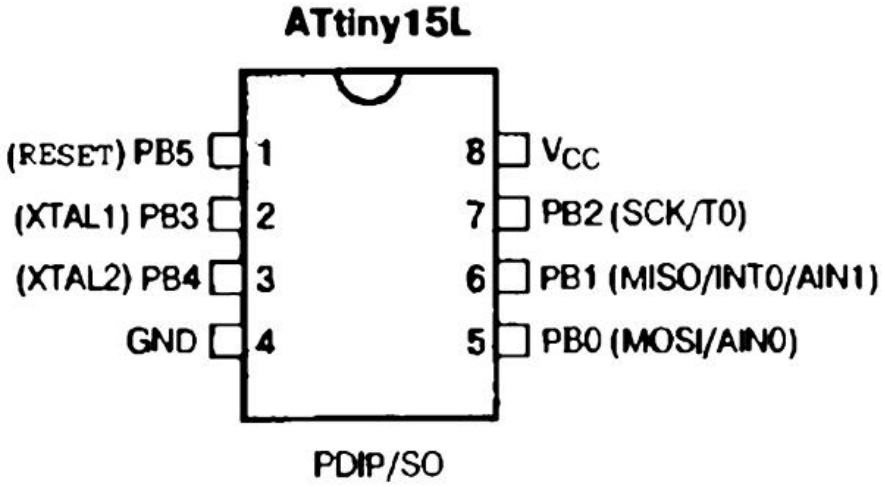



*Рис. 1.4.* Расположение выводов (вид сверху) модели ATtinyl5L

*Таблица 1.4.* Описание выводов модели ATtiny15L

| Обозначение | Номер вывода | Тип вывода | Описание |
|---|---|---|---|
| PB0(AIN0/ AREF/MOSI) | 5 | I/O | 0-й разряд порта B (Положительный вход компаратора/Вход опорного напряжения для АЦП/Вход данных при программировании) |
| PB1 (AIN1/ OC1A/MISO) | 6 | I/O | 1-Й разряд порта B (Отрицательный вход компаратора/Выход таймера/счетчика T1 (режимы Compare, PWM)/Выход данных при программировании) |
| PB2(ADC1/ TO/INTO/ SCK) | 7 | I/O | 2-й разряд порта B (Вход АЦП/Вход внешнего тактового сигнала таймера/счетчика T0/Вход внешнего прерывания/Вход тактового сигнала при программировании) |
| PB3(ADC2) | 3 | I/O | 3-й разряд порта B (Вход АЦП) |
| PB4(ADC3) | 2 | I/O | 4-й разряд порта B (Вход АЦП) |
| PB5(ADC0/ RESET) | 1 | I/O | 5-й разряд порта B (Вход АЦП/Вход сброса) |
| GND | 4 | P | Общий вывод |
| Vcc | 8 | P | Вывод источника питания |

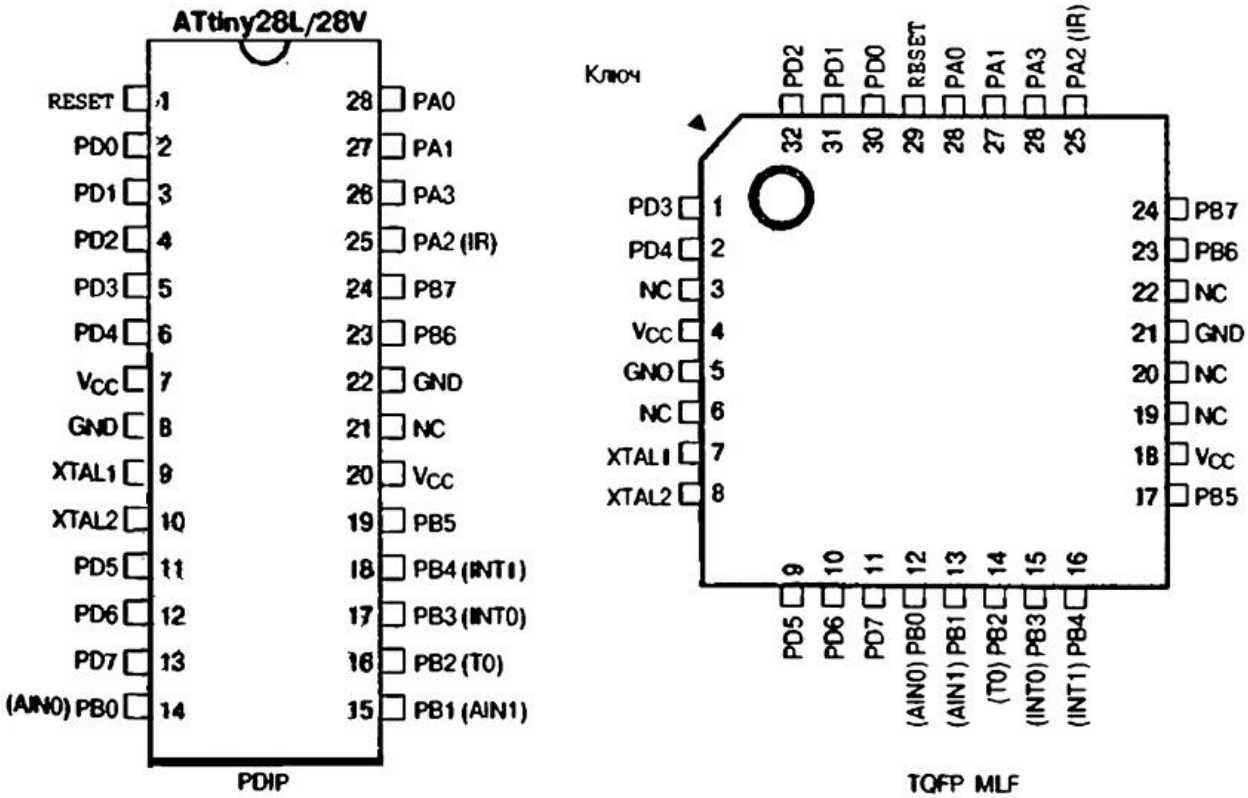



*Рис. 1.5.* Расположение выводов (вид сверху) моделей ATtiny28L/28V

*Таблица 1.5.* Описание выводов моделей ATtiny28L/28V

| Обозначение | Номер вывода | | Тип вывода | Описание |
|---|---|---|---|---|
| | DIP | TQFP,MLF | | |
| XTAL1 | 9 | 7 | I | Вход тактового генератора |
| XTAL2 | 10 | 8 | O | Выход тактового генератора |
| RESET | 1 | 29 | I | Вход сброса |
| PA0 | 28 | 28 | I/O | 0-й разряд порта A |
| PA1 | 27 | 27 | I/O | 1-й разряд порта A |
| PA2(IR) | 25 | 25 | O | 2-й разряд порта A (Выходной контакт с повышенной нагрузочной способностью) |
| PA3 | 26 | 26 | I/O | 3-й разряд порта A |
| PB0(AIN0) | 14 | 12 | I/O | 0-й разряд порта B (Положительный вход компаратора) |
| PB1 (AIN1) | 15 | 13 | I/O | 1-й разряд порта B (Отрицательный вход компаратора) |
| PB2(T0) | 16 | 14 | I/O | 2-й разряд порта B (Вход внешнего тактового сигнала таймера/счетчика T0) |
| PB3 (INT0) | 17 | 15 | I/O | 3-й разряд порта B (Вход внешнего прерывания) |
| PB4(INT1) | 18 | 16 | I/O | 4-й разряд порта B (Вход внешнего прерывания) |
| PB5 | 19 | 17 | I/O | 5-й разряд порта B |
| PB6 | 23 | 23 | I/O | 6-й разряд порта B |
| PB7 | 24 | 24 | I/O | 7-й разряд порта B |
| PD0 | 2 | 30 | I/O | 8-разрядный двунаправленный порт ввода/вывода D |
| PD1 | 3 | 31 | I/O | |
| PD2 | 4 | 32 | I/O | |
| PD3 | 5 | 1 | I/O | |
| PD4 | 6 | 2 | I/O | |
| PD5 | 1 | 9 | I/O | |
| PD6 | 12 | 10 | I/O | |
| PD7 | 13 | 11 | I/O | |
| GND | 8,22 | 5,21 | P | Общий вывод |
| Vcc | 7,20 | 4 | P | Вывод источника питания |
| NC | 21 | 3,6,19,20,22 | - | Не используются |

*Лекция №2 «Принципы работы, организация памяти и особенности выполнения команд для микроконтроллеров AVR»*

## I. Принципы работы

### 2.1 Способы тактирования

Канонический способ тактирования МК — подключение кварцевого резонатора к соответствующим выводам (рис. *2.1, а)*. Емкость конденсаторов С1 и С2 в типовом случае должна составлять 15-22 пФ. В большинстве моделей Tiny и Mega имеется специальный конфигурационный бит СКРОТ, отвечающий за усиление тактового сигнала. При установке этого бита в 1 (незапрограммированное состояние), размах колебаний генератора уменьшается, однако при этом сужается возможный диапазон частот и общая помехоустойчивость, поэтому задействовать этот режим не рекомендуется. Может быть также выбран низкочастотный кварцевый резонатор (например, "часовой" 32 768 Гц), при этом конденсаторы С1 и С2 могут отсутствовать, т.д. при установке СКРОТ в значение 0 подключаются имеющиеся в составе МК внутренние конденсаторы емкостью 36 пФ. Кварцевый резонатор можно заменить керамическим.

Естественно, тактировать МК можно и от внешнего генератора (рис. 2.1, б). Особенно это удобно, когда требуется либо синхронизировать МК с внешними компонентами, либо получить очень точную частоту тактирования, выбрав соответствующий генератор (например, серии SG-8002 фирмы Epson).

Наоборот, когда точность не требуется, можно подключить внешнюю *RC*-цепочку (рис. 2.1, *в)*. В этой схеме емкость С1 должна быть не менее 22 пФ, а резистор R1 выбирается из диапазона 3,3-100 кОм. Частота при этом определяется по формуле *F=2/3 RC*. С1 можно не устанавливать вообще, если записать лог. 0 в конфигурационную ячейку СКРОТ, подключив тем самым внутренний конденсатор 36 пФ.

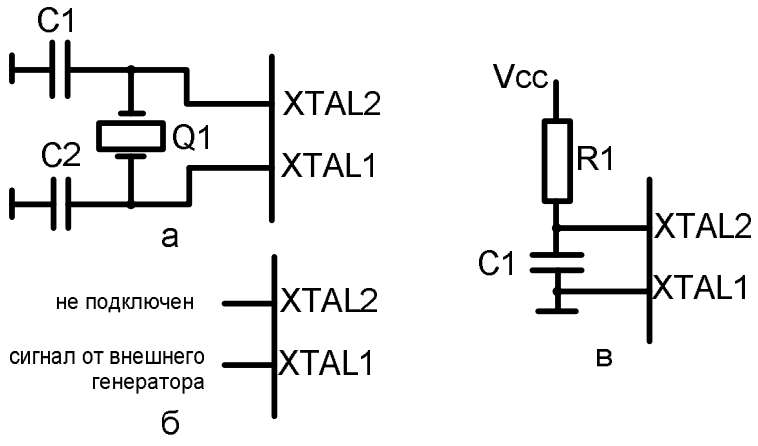



Рис. 2.3. Способы тактирования МК AVR с использованием: а — кварцевого резонатора; б — внешнего генератора; в — RC-цепочки

Наконец, можно вообще отказаться от внешних компонентов и обойтись встроенным RC-генератором, который способен работать на четырех приблизительных значениях частот (1, 2, 4 и 8 МГц). В ряде моделей предусмотрена возможность подстройки частоты этого генератора. Эту возможность наиболее целесообразно использовать в младших моделях Tiny, выпускающихся в 8-контактном корпусе — тогда выводы, предназначенные для подключения резонатора или внешнего генератора, можно задействовать для других целей, как обычные порты ввода-вывода.

По умолчанию МК семейств Tiny и Mega установлены в состояние для работы со встроенным генератором на частоте 1 МГц (CKSEL = 0001), поэтому для других режимов нужно соответствующим образом установить конфигурационные ячейки CKSEL (см. табл. 2.1). При этом следует учитывать, что состояние ячеек CKSEL = 0000 (зеркальное по отношению к наиболее часто употребляемому значению для кварцевого резонатора 1111) переводит МК в режим тактирования от внешнего генератора, и при этом его нельзя даже запрограммировать без подачи внешней частоты.

***Таблица 2.1.*** *Установка конфигурационных ячеек* CKSEL *в зависимости от режимов тактирования*

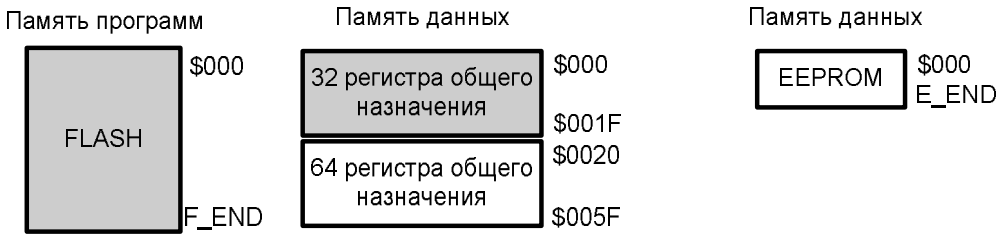
| CKSEL3... 0 | **Источник тактирования** | **Частота** |
|---|---|---|
| 0000 | Внешняя частота | 0... 16 МГц |
| 0001 | Встроенный RC-генератор | 1 МГц |
| 0010 | Встроенный RC-генератор | 2 МГц |
| 0011 | Встроенный RC-генератор | 4 МГц |
| 0100 | Встроенный RC-генератор | 8 МГц |
| 0101 | Внешняя RC-цепочка | < 0,9 МГц |
| 0110 | Внешняя RC-цепочка | 0,9... 3,0 МГц |
| 0111 | Внешняя RC-цепочка | 3,0... 8,0 МГц |
| 1000 | Внешняя RC-цепочка | 8,0... 12 МГц |
| 1001 | Низкочастотный резонатор | 32,768 кГц |
| 101x | Кварцевый резонатор | 0,4... 0,9 МГц |
| 110x | Кварцевый резонатор | 0,9... 3,0 МГц |
| 111x | Кварцевый резонатор | 3,0... 8,0 МГц |
| 1xxx(CKPOT=0) | Кварцевый резонатор | > 1,0 МГц |

# II. Организация памяти

## 2.2. Организация памяти

Организация памяти микроконтроллеров AVR семейства Tiny выполнена по схеме Гарвардского типа, в которой разделены не только адресные пространства памяти программ и памяти данных, но также и шины доступа к ним. Память данных состоит из двух областей: регистровая память и память на основе EEPROM. Каждая область расположена в своем адресном пространстве.

Обобщенная карта памяти микроконтроллеров AVR семейства Tiny приведена на **Рис. 2.2.**




| Модель | Память программ (FLASH) | | Память данных (EEPROM) | |
|---|---|---|---|---|
| | Верхняя граница (F_END) | Объём слов | Верхняя граница (F_END) | Объём слов |
| ATiny11/11L | $3FF | 1K | - | - |
| ATiny12/12L/12V | $3FF | 1K | $03F | 64 |
| ATiny15L | $3FF | 1K | $03F | 64 |
| ATiny28L/28V | $7FF | 1K | - | - |

*Рис. 2.2.* Карта памяти микроконтроллеров семейства Tiny

Обратите внимание на следующие моменты. Поскольку микроконтроллеры AVR имеют 16-разрядную систему команд, объем памяти программ на рисунке указан не в байтах, а в 16-битных словах. Символ «$» перед числом означает, что это число записано в шестнадцатеричной системе счисления. Карту памяти микроконтроллеров семейства Mega можно найти в официальном руководстве.

### 2.2.1. Память программ

Как следует из названия, память программ предназначена для хранения команд, управляющих функционированием микроконтроллера. В памяти программ хранятся также различные константы, не меняющиеся во время работы программы. Как уже было сказано, память программ представляет собой электрически стираемое ППЗУ (FLASH-ПЗУ). Поскольку все команды занимают в памяти по 16 бит, память программ имеет 16-разрядную организацию. Соответственно объем памяти микроконтроллеров семейства составляет 512...1024 16-битных слова.

Для адресации памяти программ используется счетчик команд (PC — Program Counter). Размер счетчика команд составляет 9 или 10 разрядов в зависимости от объема адресуемой памяти.

По адресу $000 памяти программ находится вектор сброса. После инициализации (сброса) микроконтроллера выполнение программы начинается с этого адреса (по этому адресу рекомендуется размещать команду относительного перехода к инициализационной части программы).

Начиная с адреса $001 располагается таблица векторов прерываний. Размер этой области зависит от модели микроконтроллера и для семейства tiny составляет от 4 (адреса $001...$004) до 8 (адреса $001...$008) векторов. При возникновении прерывания после сохранения в стеке текущего значения счетчика команд происходит выполнение команды, расположенной по адресу соответствующего вектора. Поэтому по этим адресам располагаются команды относительного перехода к подпрограммам обработки прерываний.

Если в программе прерывания не используются (запрещены), то основная программа может начинаться непосредственно с адреса $001.

Следует отметить, что память программ может использоваться не только для хранения кода программы, но также и для хранения различных констант. Для пересылки байта из памяти программ в память данных имеется специальная команда — LPM Rd, Z. Адрес, по которому производится чтение, перед использованием этой команды должен быть загружен в индексный регистр Z (см. далее). При этом старшие 15 разрядов содержимого регистра будут определять адрес слова, а младший разряд будет определять, какой из байтов будет прочитан: «О» — младший байт, «1» — старший байт (Рис. 2.3).

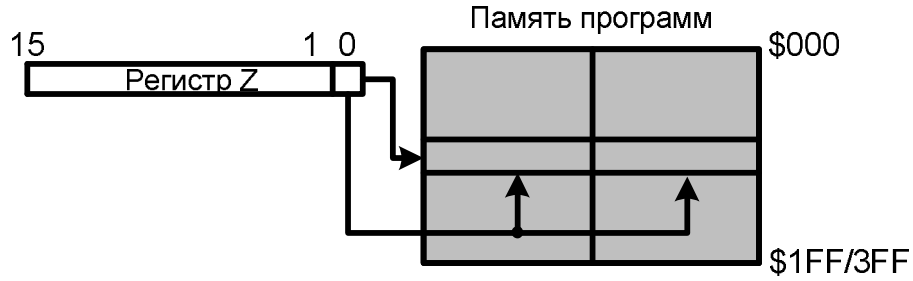



*Рис. 2.3.* Косвенная адресация памяти программ

В заключение следует отметить, что FLASH-ПЗУ, используемое в микроконтроллерах AVR, рассчитано как минимум на 1000 циклов стирания/записи.

### 2.2.2. Память данных

В отличие от памяти программ, адресное пространство памяти данных адресуется побайтно (а не пословно). Адресация полностью линейная, без какого-то деления на страницы, сегменты или банки, как это принято в

некоторых других системах. Младшие МК семейства Tiny (включая Tiny11 и Tiny28) памяти данных, как таковой, не имеют, ограничиваясь лишь регистровым файлом (РОН) и регистрами ввода-вывода (РВВ). В других моделях объем встроенной SRAM колеблется от 128 байт в представителях семейства Tiny (например, у ATtiny2313) до 4-8 кбайт у старших моделей Mega.

Адресное пространство статической памяти данных (SRAM) условно делится на несколько областей, показанных на рис. 2.4. Темной заливкой выделена часть, относящаяся собственно к встроенной SRAM, до нее по порядку адресов расположено адресное пространство регистров (первые 32 байта занимает РОН, еще 64— РВВ). Для старших моделей Mega со сложной структурой (например, ATmega128) 64-х регистров ввода-вывода может оказаться недостаточно, поэтому в них для дополнительных РВВ выделяется отдельное адресное пространство (от $60 до максимально возможного в байтовой адресации значения $FF, итого таких регистров может быть всего 160).

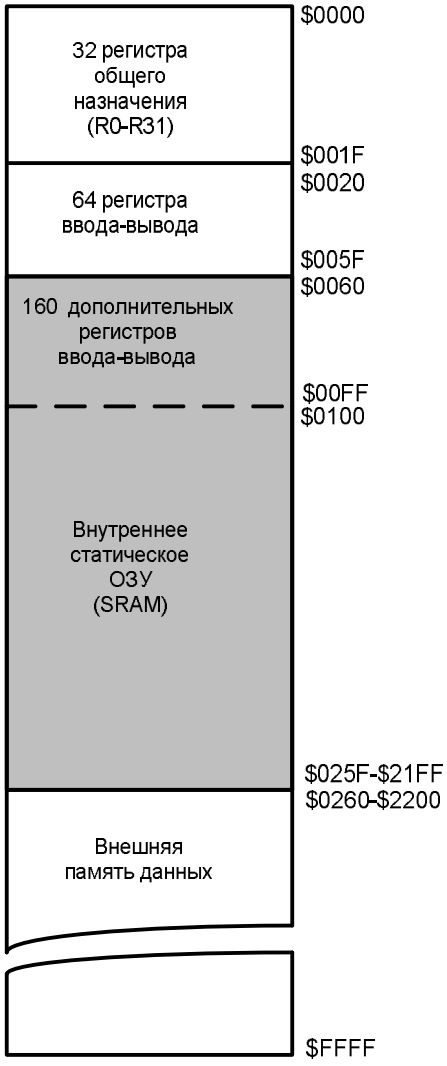



Рис. 2.4. Адресное пространство статической памяти данных (SRAM) микроконтроллеров AVR

Для некоторых моделей Mega (ATmega8515, ATmega162, ATmega128, ATmega2560 и др.) предусмотрена возможность подключения внешней памяти объемом до 64 кбайт, которая может быть любой статической разновидностью (SRAM, Flash или EEPROM) с параллельным интерфейсом.

Отметим, что адреса РОН и РВВ не отнимают пространство у ОЗУ данных (за исключением подключаемой внешней памяти у старших моделей Mega, максимальный адрес которой ограничен значением $FFFF): так, если в конкретной модели МК имеется 512 байт SRAM, а пространство регистров занимает первые 96 байт (до адреса $60), то адреса SRAM займут адресное пространство от $0060 до $025F (т. е. от 96-й до 607-й ячейки включительно). Конец встроенной памяти данных обозначается константой RAMEND.

Операции чтения/записи в память одинаково работают с любыми адресами из доступного пространства, и при работе с SRAM нужно быть внимательным: вместо записи в память вы легко можете "попасть" в какой-нибудь регистр. Например, команда загрузки значения регистра r16 в регистр r0 (mov r0,r16) равносильна записи в SRAM по нулевому адресу (sts $0000,r16). Адрес в памяти для РОН совпадает с его номером. В то же время для непосредственной записи в РВВ по его адресу в памяти к номеру регистра следует прибавить $20: так, регистр флагов SREG, который для большинства моделей располагается в конце таблицы РВВ по адресу $3F, в памяти имеет адрес $5F. Устанавливать РОН и РВВ прямой адресацией памяти неудобно: такая запись всегда отнимает два такта вместо одного, характерного для большинства других команд, хотя иногда это позволяет обойти ограничения на манипуляции с некоторыми РВВ. Но если имеется готовая программа, работающая с SRAM, то при замене моделей процессоров на более старшие нужно быть внимательным из-за того, что в них младшие адреса SRAM могут перекрываться дополнительными РВВ.

## III. Особенности выполнения команд

### 2.3.1. Функционирование конвейера и АЛУ

Одной из причин, обуславливающей большое быстродействие микроконтроллеров AVR, является использование двухуровневого конвейера при выполнении программы. Работа этого конвейера показана на **Рис. 2.5.**

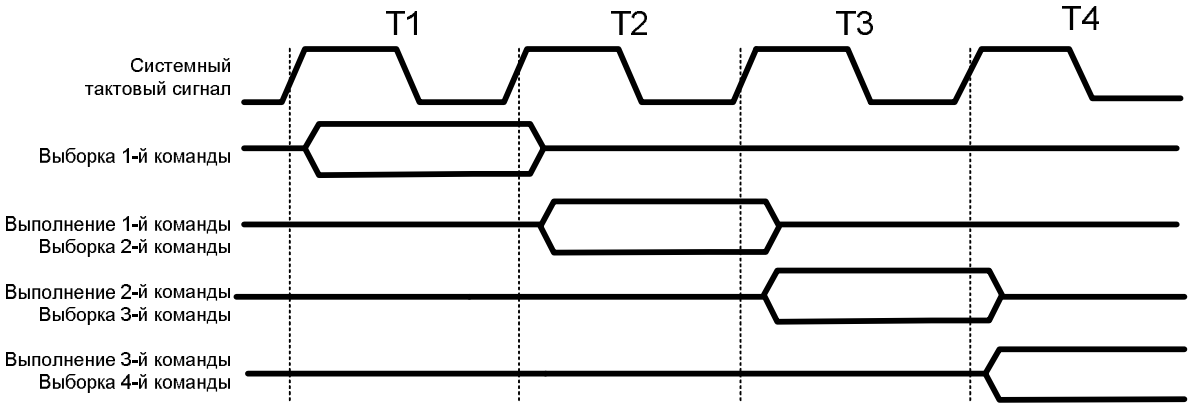



*Рис. 2.5.* Последовательность выполнения команд в конвейере

Во время первого машинного цикла происходит выборка команды из памяти программ и ее декодирование. Во время второго цикла эта команда выполняется, а параллельно происходит выборка и декодирование второй команды и т. д. В результате фактическое время выполнения каждой команды получается равным одному машинному циклу.

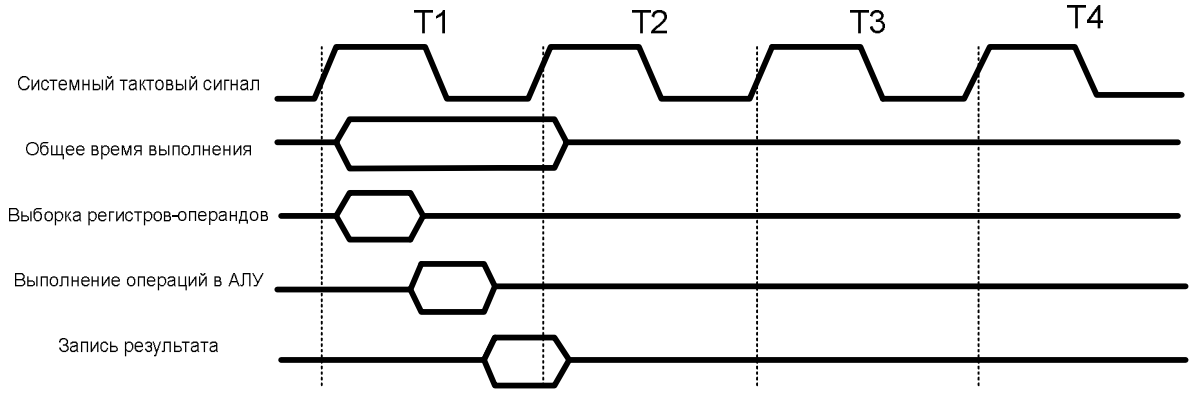



*Рис. 2.6.* Функционирование АЛУ

Благодаря подключению АЛУ непосредственно к регистровому файлу, оно выполняет одну команду (чтение содержимого двух регистров, выполнение операции и запись результата в регистр-приемник) за один такт (машинный цикл), как показано на **Рис. 2.6.**

### 2.3.2. Задержки в конвейере

При выполнении некоторых команд может происходить нарушение нормального функционирования конвейера. Наиболее ярким примером команд, вызывающих подобное нарушение, являются команды условного перехода, а также команды типа Test & Skip (проверка и пропуск следующей команды, если результат проверки положительный). В первом случае, если условие, проверяемое командой условного перехода, истинно, выполнение программы будет продолжено с некоторого адреса. А поскольку в конвейере уже произошла выборка команды, расположенной за командой перехода, время выполнения команды перехода увеличивается на один машинный цикл, во время которого происходит выборка команды, расположенной по требуемому адресу.

Во втором случае при выполнении команд типа Test & Skip следующая команда не выполняется в случае истинности проверяемого условия. Однако выборка пропускаемой команды уже произошла. Вследствие того что команда не выполняется, в конвейере образуется «дырка, которая заключается в пропуске одного или двух (в зависимости от пропускаемой команды) машинных циклов. Соответственно команды типа Test & Skip выполняются за один машинный цикл, если результат проверки условия отрицателен, и за два или три цикла, если результат проверки положителен.

Аналогично команды безусловного перехода RJMP (Relative Jump) и IJMP (Index Jump), команды вызова подпрограммы RCALL (Relative CALL) и ICALL (Index CALL) и команды возврата из подпрограмм RET (Return) и RETI (Return Interrupt) также изменяют содержимое счетчика команд PC (Program Counter), вызывая тем самым переход в памяти программ. В результате выполнения этих команд происходит «разрыв» в работе конвейера, а вследствие этого происходит задержка выполнения программы на несколько (2...4) машинных циклов. По той же причине нарушение нормального функционирования конвейера происходит и при возникновении прерывания. Минимальная задержка при этом составляет 4 машинных цикла.

### 2.3.3. Счетчик команд

Размер счетчика команд составляет 9 (модели ATtinyl lx, ATtinyl2x, ATtinyl5L) или 10 (модели ATtiny28x) разрядов. Напрямую (как регистр) счетчик команд из программы недоступен.

При нормальном выполнении программы содержимое счетчика команд автоматически увеличивается на 1 или на 2 (в зависимости от выполняемой команды) в каждом машинном цикле. Этот порядок нарушается при выполнении команд перехода, вызова и возврата из подпрограмм, а также при возникновении прерываний.

После включения питания, а также после сброса микроконтроллера в счетчик программ автоматически загружается значение $000. Как правило, по этому адресу располагается команда относительного перехода (RJMP) к инициализационной части программы.

При возникновении прерывания в счетчик команд загружается адрес соответствующего вектора прерывания ($001...$010). Если прерывания используются в программе, по этим адресам должны размещаться команды относительного перехода к подпрограммам обработки прерываний. В противном случае основная программа может начинаться непосредственно с адреса $001.

### 2.3.4. Команды типа «проверка/пропуск» (Test & Skip)

В командах этого типа производится проверка условия, результат которой влияет на выполнение следующей команды. Если условие истинно, следующая команда игнорируется. Например, команда SBRS Rd,b проверяет разряд b регистра Rd и игнорирует следующую команду, если этот разряд равен «1». В действительности переход к следующей инструкции производится увеличением счетчика команд на 1, а пропуск команды требует загрузки нового значения в счетчик команд. Следовательно, когда проверяемое условие истинно, в конвейере возникает задержка. Длительность задержки зависит от размера пропускаемой команды и составляет от одного до двух машинных циклов.

### 2 3.5. Команды условного перехода

В этих командах производится проверка условия, результат которой влияет на состояние счетчика команд. Если условие истинно, происходит переход по заданному адресу. Если же условие ложно, выполняется следующая команда.

Команды условного перехода имеют ограничение по области действия. В действительности новое значение счетчика команд получается прибавлением к нему или вычитанием из него некоторого смещения. А поскольку под значение смещения в слове команды отводится всего 7 разрядов, максимальная величина перехода составляет 128 слов (-64...+64).

Так как переход по заданному адресу осуществляется загрузкой нового значения в счетчик команд, то в случае истинности проверяемого условия в конвейере возникает задержка длительностью в один машинный цикл.

### 2.3.6. Команда безусловного перехода

Для безусловного перехода по требуемому адресу в памяти программ имеется только одна команда — команда относительного перехода RJMP. Процесс выполнения команды заключается в изменении содержимого счетчика команд путем прибавления к нему или вычитания из него некоторого значения, являющегося операндом команды, как показано на **Рис. 2.7.**

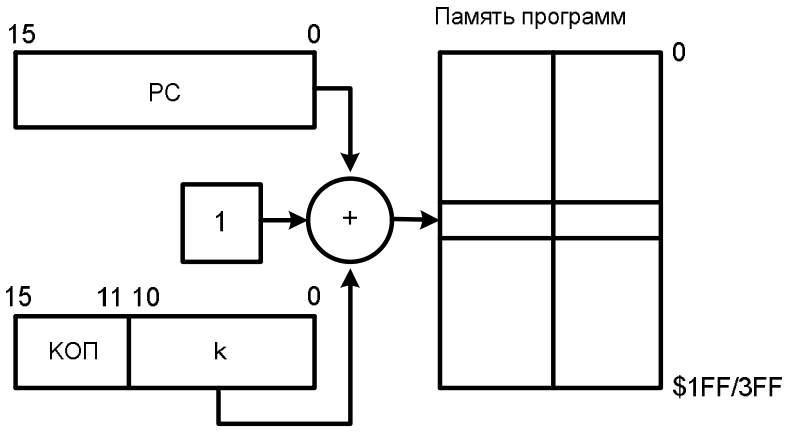



*Рис. 2.7.* Относительная адресация памяти программ

Положение разрядов k на рисунке показано условно. В программах в качестве операндов этой команды вместо констант используются метки. Ассемблер сам вычисляет величину перехода и подставляет это значение в слово команды. Проиллюстрируем сказанное следующим примером:

```
cpi     r16,$42         ;Сравниваем регистр R16 с числом $42
brne    error           ;Переход,   если R16 о  $42
rjmp    ok              ;Безусловный переход
error:  nop
ok:     nop             ;Место перехода по  команде RJMP
```

Поскольку команда относительного перехода изменяет содержимое счетчика команд, она выполняется за 2 машинных цикла.

### 2.3.7. Команда вызова подпрограмм

Как и для реализации безусловных переходов, для вызова подпрограмм имеется только одна команда — команда относительного вызова RCALL. Если не принимать во внимание некоторые отличия, описанные ниже, эта команда работает так же, как и команда относительного, безусловного перехода RJMP. Команда RCALL сохраняет в стеке значение счетчика команд. Затем содержимое счетчика команд увеличивается или уменьшается на некоторое значение, являющееся операндом команды **(Рис. 2.7).**

В программах в качестве операндов этой команды, как и в случае команды RJMR используются метки. Ассемблер сам вычисляет величину перехода и подставляет это значение в слово команды.

Команда относительного вызова подпрограмм выполняется за 3 машинных цикла, 2 из которых затрачиваются на сохранение в стеке двух байт счетчика команд.

### 2.3.8. Команды возврата из подпрограмм

В конце каждой подпрограммы обязательно должна находиться команда возврата из нее. В системе команд микроконтроллеров семейства имеется две

таких команды. Для возврата из обычной подпрограммы, вызываемой командой RCALL, используется команда RET. Для возврата из подпрограммы обработки прерывания используется команда RETI.

Обе команды восстанавливают из стека содержимое счетчика команд, сохраненное там перед переходом к подпрограмме. Команда возврата из подпрограммы **RETI** дополнительно устанавливает в «1» флаг общего разрешения прерываний I регистра SREG, сбрасываемый аппаратно при возникновении прерывания.

На выполнение каждой из команд возврата из подпрограммы требуется 4 машинных цикла.

### 2.3.9 Стек

В микроконтроллерах AVR семейства Tiny стек реализован аппаратно. Глубина стека равна трем уровням, а размер равен размеру счетчика команд (9 или 10 разрядов). Стек расположен в собственной области памяти и имеет LIFO-организацию (Last In First Out — последним вошел, первым вышел), т. е. значение, записанное последним, будет прочитано первым.

При вызове подпрограммы адрес команды, расположенной за командой RCALL, сохраняется в стеке. При возврате из подпрограммы этот адрес извлекается из стека и загружается в счетчик команд. Тоже 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.

Непосредственно из программы стек недоступен (для МК tiny), т. к. в наборе команд микроконтроллера отсутствуют команды занесения в стек и извлечения из стека. Указатель стека также недоступен из программы, т. е. он не может быть явно прочитан или модифицирован. Поэтому микроконтроллер сам управляет перемещением данных по стеку. Работа стека показана на **Рис. 2.8** и **Рис. 2.9**

Рассмотрим выполнение команды RCALL **(Рис. 2.8):** содержимое счетчика команд пересылается на 1-й уровень стека, а предыдущие значения предварительно «сползают» на один уровень (значение, находившееся на первом уровне, перемещается на второй уровень и т. д.). Заметим, что в результате этой операции будет потеряно значение, расположенное на 3-м уровне стека.

При выполнении команды возврата из подпрограммы RET или RETI **(Рис. 2.9)** значение, хранящееся на 1-м уровне стека, заносится в счетчик команд. Во время этой операции все значения «поднимаются» на один уровень вверх (значение, находившееся на втором уровне, перемещается на первый уровень и т. д.).

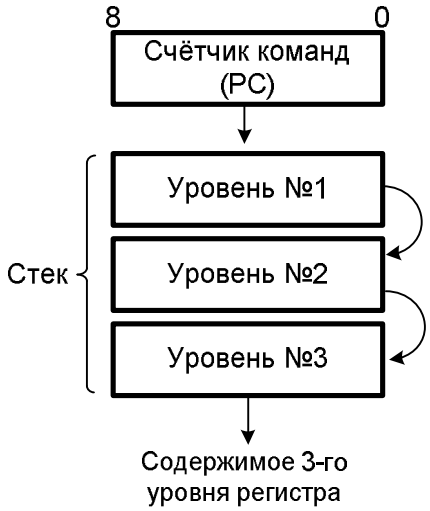



*Рис. 2.8.* Работа стека при выполнении команды RCALL

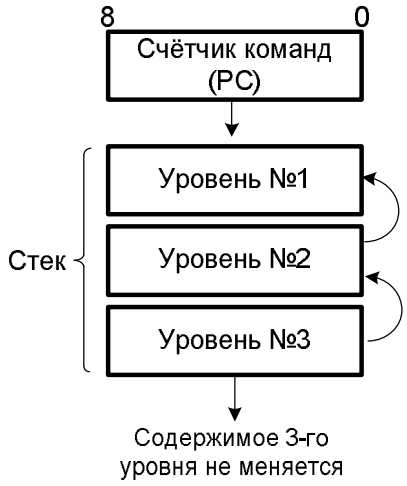



*Рис. 2.9.* Работа стека при выполнении команды RET

### 2.3.10- Регистры общего назначения

Все регистры общего назначения объединены в файл, структура которого показана на **Рис. 1.10.** Как уже было сказано, в микроконтроллерах AVR все 32 РОН непосредственно доступны АЛУ, в отличие от микроконтроллеров других фирм, в которых имеется только один такой регистр — рабочий регистр W (аккумулятор). Благодаря этому любой **РОН** может использоваться во всех командах и как операнд-источник, и как операнд-приемник. Исключение составляют лишь пять арифметических и логических команд, выполняющих действия между константой и регистром (SBCI, SUBI, CPI, ANDI, ORI), а также команда загрузки константы в регистр (LDI). Эти команды могут обращаться только ко второй половине регистров (R16 ... R31).

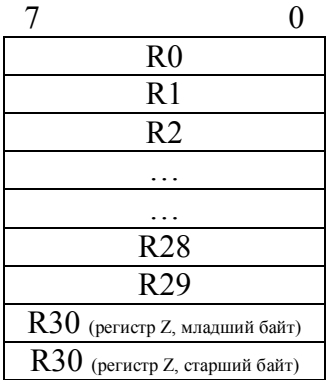



*Рис. 2.10.* Структура файла регистров общего назначения

Два старших регистра общего назначения формируют 16-разрядный индексный регистр Z, который используется в качестве указателя при косвенной адресации памяти программ и памяти данных. Так как объем адресуемой памяти данных составляет всего 32 байт, при обращении к ней используется только младший байт (регистр R30). Содержимое старшего байта индексного регистра (регистр R31) при косвенной адресации памяти данных автоматически очищается процессором.

### 2.3.11 Регистры ввода/вывода

Регистры ввода/вывода располагаются в так называемом пространстве ввода/вывода размером 64 байт. Все РВВ можно разделить на две группы: служебные регистры микроконтроллера и регистры, относящиеся к периферийным устройствам (в том числе порты ввода/вывода). Размер каждого регистра — 8 бит.

Распределение адресов пространства ввода/вывода зависит от конкретной модели микроконтроллера, т. к. разные модели имеют различный состав периферийных устройств и соответственно разное количество регистров. Размещение РВВ в адресном пространстве ввода/вывода для всех моделей семейства приведено в табл. 2.2. Прочерк в таблице означает, что для данной модели этот адрес зарезервирован, но запись по этому адресу запрещена.

*Таблица 2.2.* РВВ микроконтроллеров семейства Tiny

| Название | Функция | Адрес | ATtiny | | | |
|---|---|---|---|---|---|---|
| | | | 11x | 12x | 15L | 28x |
| SREG | Регистр состояния | $3F | + | + | + | + |
| GIMSK | Общий регистр маски прерываний | $3B | + | + | + | - |
| GIFR | Общий регистр флагов прерываний | $3A | + | + | + | - |
| TIMSK | Регистр маски прерываний от таймера/счетчика | $39 | + | + | + | - |
| TIFR | Регистр флагов прерываний от таймера/счетчика | $38 | + | + | + | - |
| MCUCR | Общий регистр управления микроконтроллером | $35 | + | + | + | - |
| MCUSR | Регистр состояния микроконтроллера | $34 | + | + | + | - |
| TCCR0 | Регистр управления таймером/счетчиком T0 | $33 | + | + | + | - |
| TCNT0 | Счетный регистр таймера/счетчика T0 (8-разрялный) | $32 | + | + | + | - |
| OSCCAL | Регистр калибровки тактового генератора | $31 | - | + | + | - |
| TCCRI | Регистр управления таймером/счетчиком T1 | $30 | - | - | + | - |
| TCNTI | Счетный регистр таймера/счетчика T1 | $2F | - | - | + | - |
| OCRIA | Регистр совпадения A таймера/счетчика T1 | $2E | - | - | + | - |
| OCRIB | Регистр совпадения B таймера/счетчика T1 | $2D | - | - | + | - |
| SFIOR | Регистр специальных функций | $2C | - | - | + | - |
| WDTCR | Регистр управления сторожевым таймером | $21 | + | + | + | - |
| EEAR | Регистр адреса EEPROM | $1E | - | + | + | - |
| EEDR | Регистр данных EEPROM | $1D | - | + | + | - |
| EECR | Регистр управления EEPROM | $1C | - | + | + | - |
| PORTA | Регистр данных порта A | $1B | - | - | - | + |
| PACR | Регистр управления порта A | $1A | - | - | - | + |
| PINA | Выводы порта A | $19 | - | - | - | + |
| PORTB | Регистр данных порта B | $18 | + | + | + | - |
| DDRB | Регистр направления данных порта B | $17 | + | + | + | - |
| PINB | Выводы порта B | $16 | + | + | + | + |
| PORTD | Регистр данных порта D | $12 | - | - | - | + |
| DDRD | Регистр направления данных порта D | $11 | - | - | - | + |
| PIND | Выводы порта D | $10 | - | - | - | + |
| ACSR | Регистр управления и состояния аналогового компаратора | $08 | + | + | + | + |
| ADMUX | Регистр управления мультиплексором АЦП | $07 | + | - | - | - |
| MCUCS | Регистр управления и состояния микроконтроллера | | - | - | - | + |
| ADCSR | Регистр управления и состояния АЦП | $06 | + | - | - | - |
| ICR | Регистр управления прерываниями | | - | - | - | + |
| ADCH | Регистр данных АЦП (старший байт) | $05 | + | - | - | - |
| IFR | Регистр флагов прерываний | | - | - | - | + |
| ADCL | Регистр данных АЦП (младший байт) | $04 | + | - | - | - |
| TCCR0 | Регистр управления таймером/счетчиком T0 | | - | - | - | + |
| TCNT0 | Счетный регистр таймера/счетчика T0 (8-разрядный) | $03 | - | - | - | + |
| MODCR | Регистр управления модулятором | $02 | - | - | - | + |
| WDTCR | Регистр управления сторожевым таймером | $01 | - | - | - | + |
| OSCCAL | Регистр калибровки тактового генератора | $00 | - | - | - | + |

К любому регистру ввода/вывода можно обратиться с помощью команд IN и OUT выполняющих пересылку данных между одним из 32-х РОН и пространством ввода/вывода. Кроме того, имеются 4 команды поразрядного доступа, использующие в качестве операндов регистры ввода/вывода: команды установки/сброса от цельного бита (SBI и CBI) и команды проверки состояния отдельного бита (SBIS и SBIC). Однако указанные команды могут обращаться только к 1-й половине регистров ввода/вывода (адреса $00...$ 1F).

**SREG (регистр состояния)**

Регистр состояния SREG (Status REGistr) располагается по адресу $3E. Этот регистр является, по сути дела, набором флагов, показывающих текущее состояние микроконтроллера. Эти флаги автоматически устанавливаются в «1» или в «0» при наступлении определенных событий (в соответствии с результатом выполнения команд). Все разряды этого регистра доступны как для чтения, так и для записи в любой момент времени; после сброса микроконтроллера все разряды регистра сбрасываются в «0». Содержимое этого регистра показано ниже на **Рис.** 2.11, а его описание приведено в табл. 2.3.

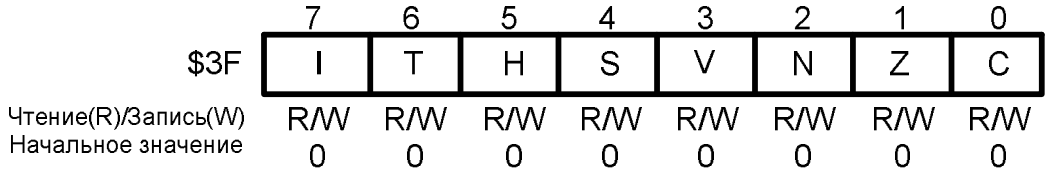



*Рис. 2.11.* Формат регистра состояния SREG

**GIMSK, TIMSK, GIFR, TIFR (регистры управления прерываниями)**

Эти четыре регистра предназначены для управления внешними прерываниями (регистры GIMSK и GIFR) и прерываниями от таймеров (регистры TIMSK и TIFR). Регистры масок GIMSK (General Interrupt MaSK — общий регистр маски прерываний) и TIMSK (Timer Interrupt MaSK — регистр маски прерываний от таймеров) используются для разрешения/запрещения отдельных прерываний, а регистры флагов GIFR (General Interrupt Flag Register — общий регистр флагов прерываний) и TIFR (Timer Interrupt Flag Register — регистр флагов прерываний от таймеров) содержат флаги, показывающие, произошло или нет соответствующее прерывание. В моделях ATtiny28x вместо описанных используются два других регистра: регистр управления прерываниями ICR (Interrupt Control Register) и регистр флагов прерываний IFR (Interrupt Flag Register).

*Таблица 1.7.* Разряды регистра состояния SREG

| Разряд | название | Описание |
|---|---|---|
| 7 | I | Общее разрешение прерываний. Для разрешения прерываний этот флаг должен быть установлен в «1». Разрешение/запрещение отдельных прерываний производится установкой или сбросом соответствующих разрядов регистров масок прерываний. Если флаг сброшен (0), то прерывания запрещены независимо от состояния этих регистров. Флаг сбрасывается аппаратно после входа в прерывание и восстанавливается командой RETI для разрешения обработки следующих прерываний. |
| 6 | T | Хранение копируемого бита. Этот разряд регистра используется в качестве источника или приемника командами копирования битов BLD (Bit LoaD) и BST (Bit STore). Заданный разряд любого **РОН** может быть скопирован в этот разряд командой BST или установлен в соответствии с содержимым данного разряда командой BID. |
| 5 | H | Флаг половинного **(Half)** переноса. Этот флаг устанавливается в «1», если имел место перс-нос из младшей половины байта (из 3-го разряда в 4-й), или заем из старшей половины байта при выполнении некоторых арифметических операций. |
| 4 | S | Флаг знака (Sign). Этот флаг равен результату операции «Исключающее ИЛИ» (XOR) между флагами N (отрицательный результат) и V (переполнение числа в дополнительном коде). Соответственно этот флаг устанавливается в «1», если результат выполнения арифметической операции меньше нуля. |
| 3 | V | Флаг переполнения (Overflow) дополнительного кода. Этот флаг устанавливается в «1» при переполнении разрядной сетки знакового результата. Используется при работе со знаковыми числами (представленными в дополнительном коде). |
| 2 | N | Флаг отрицательного (Negative) значения. Этот флаг устанавливается в «1», если старший (7-й) разряд результата операции равен «1». В противном случае флаг равен «0». |
| 1 | Z | Флаг нуля (Zero). Этот флаг устанавливается в «1», если результат выполнения операции равен нулю. |
| 0 | C | Флаг переноса (Cary). Этот флаг устанавливается в «1», если в результате выполнения операции произошел выход за границы байта. |

### MCUCR (регистр управления микроконтроллером)

Регистр управления микроконтроллером имеется во всех моделях, кроме ATtiny28x, и расположен по адресу $35. Этот регистр содержит ряд флагов, используемых для общего управления микроконтроллером. Состав флагов, размещенных в регистре MCUCR, несколько меняется от модели к модели. Неиспользуемые разряды регистра доступны только для чтения и содержат «0». Все используемые разряды регистра доступны как для чтения, так и для записи в любой момент времени. После сброса микроконтроллера во всех разрядах регистра записаны «0».

Формат этого регистра для различных моделей приведен на **Рис. 2.12,** а описание его разрядов приведено в табл. 2.4. При изменении состояния разрядов ISC01 и ISC00 возможна ложная генерация прерывания INT0. Чтобы этого избежать, рекомендуется на время изменения указанных разрядов запретить внешнее прерывание.

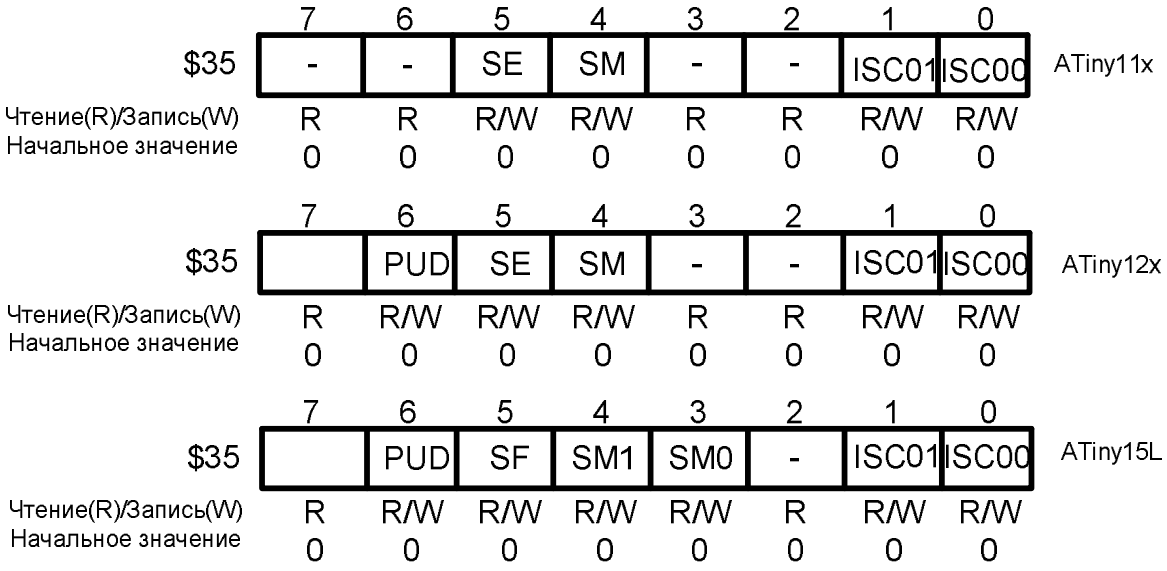



*Рис. 2.12.* Формат регистра управления микроконтроллером MCUCR

*Таблица 2.4.* Разряды регистра MCUCR моделей ATtiny11x/12x/15L

| Разряд | Название | Описание | Модель |
|---|---|---|---|
| 7 | - | Не используется, читается как «0» | Все модели |
| 6 | PUD | Запрещение использования внутренних подтягивающих резисторов порта B.<br>Если этот разряд установлен в «1», подключение внутренних подтягивающих резисторов порта B запрещено, если сброшен — разрешено | ATtinyl2x<br>ATtinyl5L |
| | - | Не используется, читается как «0» | ATtinyl lx |

<table>
<tr><td></td><td></td><td colspan="3"></td><td></td></tr>
<tr><td>5</td><td>SE</td><td colspan="3">Разрешение перехода в режим пониженного энергопотребления. Если этот разряд установлен в «1», то по команде «SLEEP» микроконтроллер переходит в «спящий» режим</td><td>Все модели</td></tr>
<tr><td>4</td><td>SM</td><td colspan="3">Выбор режима пониженного энергопотребления. Состояние этого разряда определяет, в какой режим перейдет микроконтроллер после выполнения команды «SLEEP». Если этот разряд установлен в «1», микроконтроллер переключится в режим «Power Down», если разряд сброшен — в режим «Idle».</td><td>ATtinyl lx<br>ATtinyl2x</td></tr>
<tr><td>3</td><td>-</td><td colspan="3">Не используется, читается как «0»</td><td>ATtinyl lx<br>ATtinvl2x</td></tr>
<tr><td>4,3</td><td>SMI,<br>SMO</td><td colspan="3">Выбор режима пониженного энергопотребления. Состояние этих разрядов определяет, в какой режим перейдет микроконтроллер после выполнения команды «SLEEP»</td><td></td></tr>
<tr><td></td><td></td><td>SM1</td><td>SM0</td><td>Режим</td><td>ATtinyl 5L</td></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td>Idle</td><td></td></tr>
<tr><td></td><td></td><td>0</td><td>1</td><td>Режим снижения шумов АЦП</td><td></td></tr>
<tr><td></td><td></td><td>1</td><td>0</td><td>Power Down</td><td></td></tr>
<tr><td></td><td></td><td>1</td><td>1</td><td>Зарезервировано</td><td></td></tr>
<tr><td>2</td><td>-</td><td colspan="3">Не используется, читается как «0»</td><td>Все модели</td></tr>
<tr><td>1,0</td><td>ISC01,<br>ISC00</td><td colspan="3">Условие генерации внешнего прерывания INTO</td><td>Все модели</td></tr>
<tr><td></td><td></td><td>ISC01</td><td>ISC00</td><td>Условие</td><td></td></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td>По НИЗКОМУ уровню на выводе INT0</td><td></td></tr>
<tr><td></td><td></td><td>0</td><td>1</td><td>При любом изменении сигнала на выводе INT0</td><td></td></tr>
<tr><td></td><td></td><td>1</td><td>0</td><td>По спадающему фронту сигнала на выводе INT0</td><td></td></tr>
<tr><td></td><td></td><td>1</td><td>1</td><td>По нарастающему фронту сигнала на выводе INT0</td><td></td></tr>
</table>

**MCUCS (регистр управления и состояния микроконтроллера)**

Этот регистр присутствует только в моделях ATtiny28x и используется для тех же целей, что и регистры MCUCR и MCUSR в остальных моделях семейства. Формат этого регистра приведен на Рис. **2.13** а описание его разрядов — в табл. 2.5.

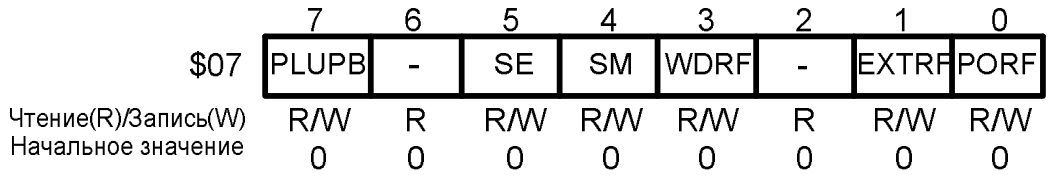



*Рис. 2.13.* Формат регистра MCUCS модели ATtiny28x

*Таблица 2.5.* Разряды регистра MCUCS моделей ATtiny28x

| Разряд | Название | Описание |
|---|---|---|
| 7 | PLUPB | Включение внутренних подтягивающих резисторов порта B. Если этот разряд установлен в «1», внутренние подтягивающие резисторы на всех входах порта B включены, если сброшен — выключены. Если используется любая из альтернативных функций порта B, подтягивающие резисторы отключаются независимо от состояния разряда PLUPB |
| 6 | - | Не используется, читается как «0» |
| 5 | SE | Разрешение перехода в режим пониженного энергопотребления. Если этот разряд установлен в «1», то по команде «SLEEP» микроконтроллер переходит в «спящий» режим |
| 4 | SM | Выбор режима пониженного энергопотребления. Состояние этого разряда определяет, в какой режим перейдет микроконтроллер после выполнения команды «SLEEP». Если этот разряд установлен в «1», «спящим» режимом является режим «Power Down». Если этот разряд сброшен — режим «Idle». Более полную информацию см. в разделе 3.3 |
| 3 | WDRF | Флаг сброса сторожевого таймера. Этот флаг устанавливается, если сброс микроконтроллера произошел из-за переполнения сторожевого таймера. Разряд сбрасывается в результате сброса по питанию или непосредственной записью в него лог. 0 |
| 2 | - | Не используется, читается как «0» |
| 1 | EXTRF | Флаг аппаратного сброса. Этот флаг устанавливается, если произошел аппаратный сброс микроконтроллера (по внешнему сигналу). Разряд сбрасывается в результате сброса по питанию или непосредственной записью в него лог. 0 |
| 0 | PORF | Флаг сброса по питанию. Этот флаг устанавливается в «1» в результате сброса по питанию. Разряд сбрасывается только непосредственной записью в него лог. 0 |

### *2.3.12. Способы адресации памяти данных*

Микроконтроллеры AVR семейства Tiny поддерживают только 4 способа адресации я доступа к различным областям памяти данных. Причем 3 способа из 4-х являются всего лишь разновидностями прямой адресации.

Обратите внимание, что на рисунках этого параграфа, а также далее в книге будет встречаться аббревиатура КОП. Эта аббревиатура обозначает часть (или части) слова команды, содержащую значение Кода ОПерации.

**Прямая адресация**

При прямой адресации адреса операндов содержатся непосредственно в слове команды. Микроконтроллеры семейства поддерживают следующие

разновидности прямой адресации: прямая адресация одного РОН, прямая адресация двух РОН, прямая адресация РВВ.

**Прямая адресация одного регистра общего назначения**

Этот способ адресации используется в командах, оперирующих с одним из регистров общего назначения. При этом адрес регистра-операнда (его номер) содержится в разрядах 8...4 (5 бит) слова команды **(Рис. 2.14).** Положение разрядов d на рисунке показано условно.

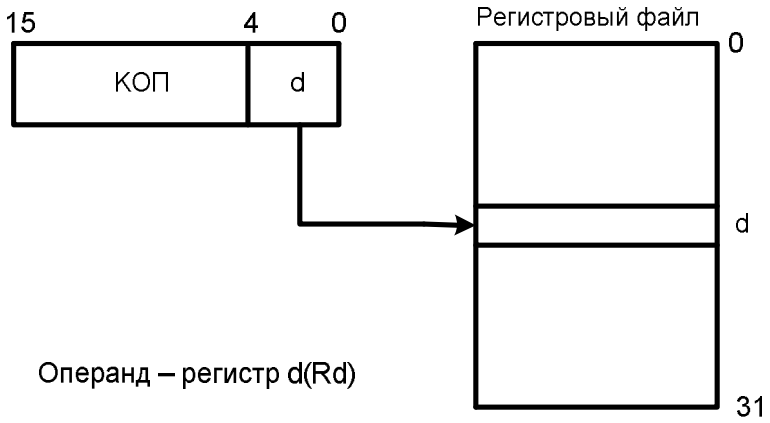



*Рис. 2.14.* Прямая адресация одного регистра общего назначения

Примером команд, использующих этот способ адресации, являются команды работы со стеком (PUSH, POP), команды инкремента (INC), декремента (DEC), а также некоторые команды арифметических операций.

**Прямая адресация двух регистров общего назначения**

Этот способ адресации используется в командах, оперирующих одновременно с двумя регистрами общего назначения. При этом адрес регистра-источника содержится в разрядах 9, 3...0 (5 бит), а адрес регистра-приемника в разрядах 8...4 (5 бит) слова команды (Рис. 2.15). Положение разрядов r и d на рисунке показано условно.

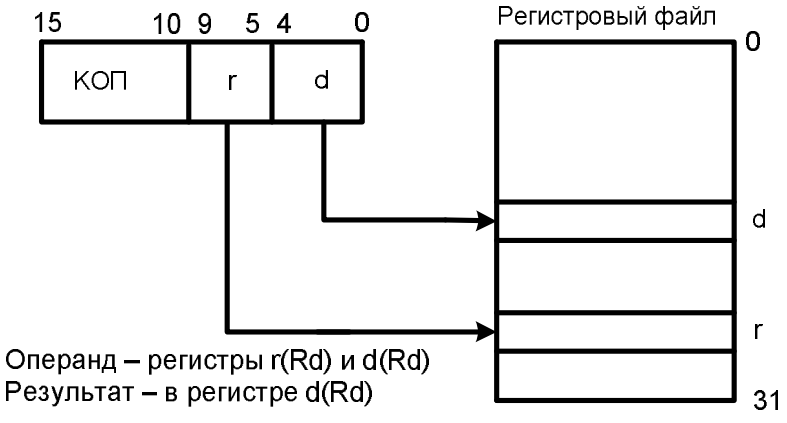


*Рис. 2.15.* Прямая адресация двух регистров общего назначения

К командам, использующим этот способ адресации, относятся команда пересылки данных из регистра в регистр (MOV), а также большинство команд арифметических операций. Кроме того, этот способ адресации используют даже некоторые команды, имеющие только один регистр-операнд. При этом источником и приемником является один и тот же регистр. В качестве примера можно привести команду очистки регистра (CLR Rd), которая в действительности выполняет операцию «Исключающее ИЛИ» регистра с самим собой (EOR Rd,Rd).

**Прямая адресация регистра ввода/вывода**

Данный способ адресации используется командами пересылки данных между регистром ввода/вывода и регистровым файлом —IN и OUT. В этом случае адрес регистра ввода/вывода содержится в разрядах 10,9,3...0 (6 бит), а адрес РОН — в разрядах 8...4 (5 бит) слова команды (Рис. 2.16). Положение разрядов r/d и P на рисунке показано условно.

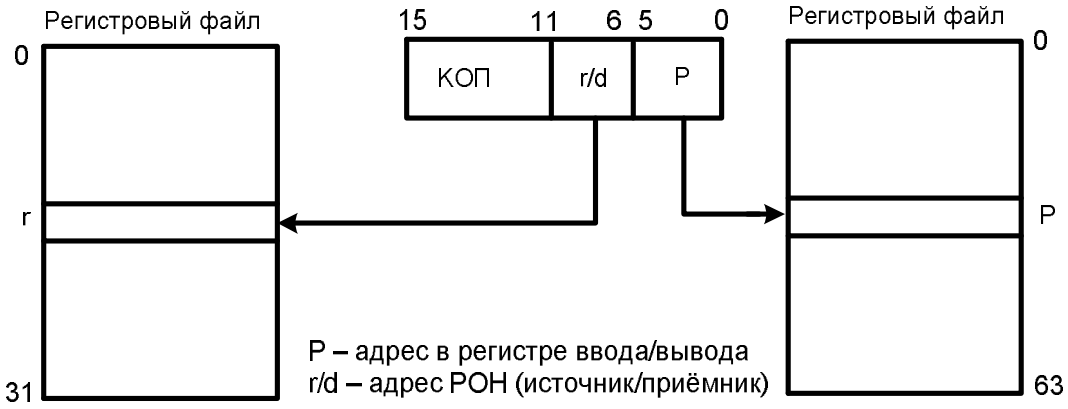



*Рис. 2.16.* Прямая адресация регистра ввода/вывода

***Лекция №3 «Организация обмена с внешними устройствами, память, прерывания для микроконтроллеров PIC и AVR»***

## 1.1 Порты ввода/вывода

Портов ввода/вывода в разных моделях может быть от 1 до 7. Номинально порты восьмиразрядные, в некоторых случаях разрядность ограничена числом выводов корпуса и может быть меньше восьми. Порты обозначаются буквами A, B, C, D и далее, причем необязательно по порядку.

Для сокращения числа контактов корпуса в подавляющем большинстве случаев внешние выводы, соответствующие портам, кроме своей основной функции (двунаправленного ввода-вывода) несут также дополнительную.

Отличительной особенностью портов микроконтроллеров AVR при использовании их в качестве цифровых портов ввода/вывода общего назначения является реализация истинной функциональности вида «чтение/модификация/запись». Благодаря этому можно выполнять операции над любым выводом (с помощью команд SBI и CBI), не влияя на другие выводы порта. Это относится к изменению режима работы контакта ввода/вывода, к изменению состояния выходного буфера (для выходов) и к изменению внутреннего подтягивающего резистора (выходов).

Каждый вывод порта может быть индивидуально сконфигурирован как вход или выход, причем в случае функционирования в качестве входа к нему может быть по выбору подключено подтягивающее сопротивление.

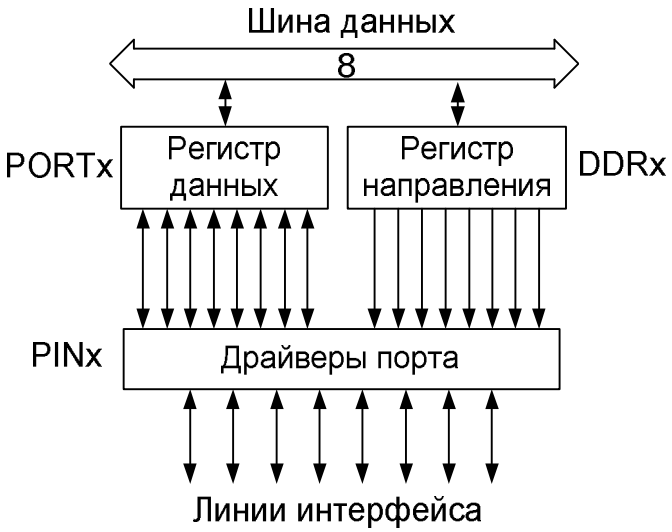



Рисунок 3.1 – Структура порта PortX

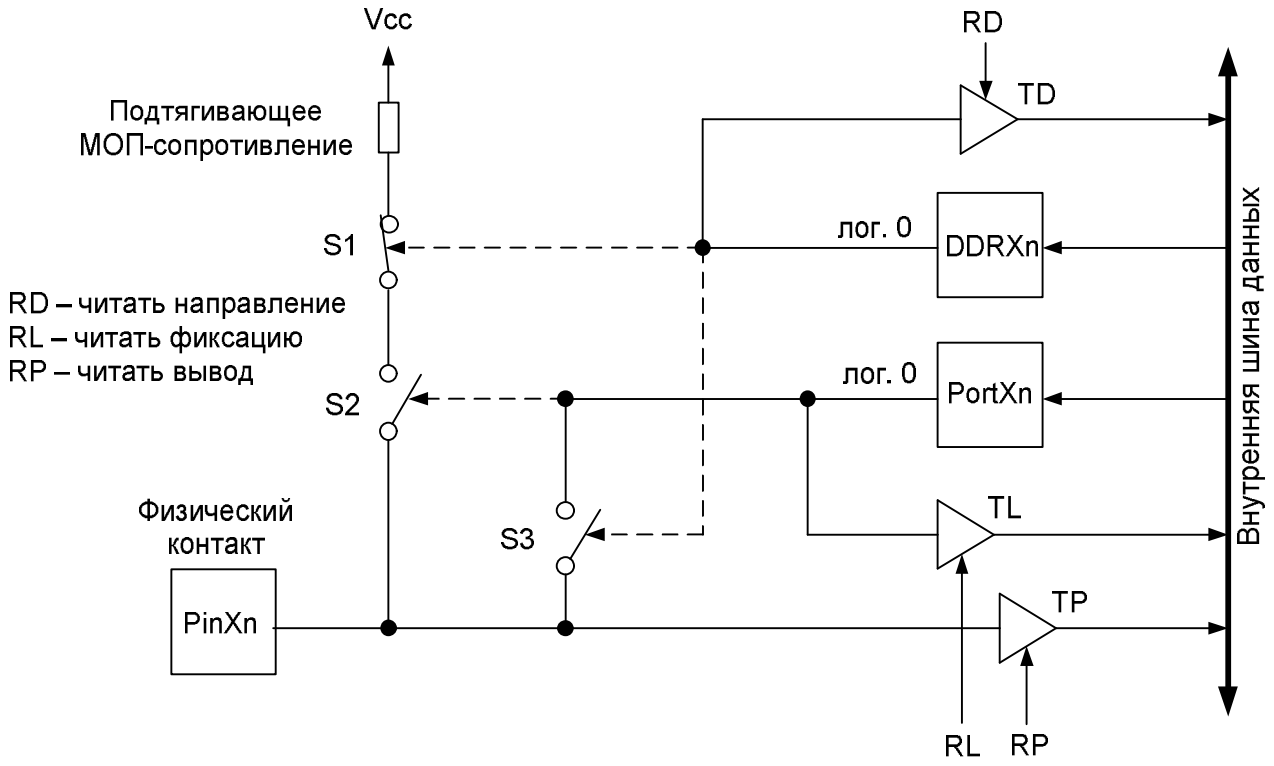



Рисунок 3.2 – Принципиальная схема вывода порта

На рисунке 3.2 показана схема, по которой построены порты микроконтроллеров семейства AVR. Буквой “X” в обозначении вывода регистра порта и регистра направления передачи данных используется вместо обозначения порта, то есть, вместо нее можно поставить буквы “A”, “B”, “C” или “D”. Номера разрядов 0-7 внутри регистра представлены буквой “n”.

Положение ключей S1-S3, показанные на рис. 3.2, соответствуют сигналам низкого уровня на выходах регистров DDRXn и PORTXn.

Обращение к каждому из портов осуществляется по трем различным адресам в области ввода/вывода:

1. Регистр направления передачи данных (DDX на рис3.2) определяет назначение вывода порта: вход или выход.
    - Если разряд “n” в регистре DDXn содержит сигнал низкого уровня, то соответствующий вывод сконфигурирован как вход. Ключ S3 (замыкающий контакт) в этом случае разомкнут и отделяет регистр PortXn от вывода PinXn. Ключ S1 (размыкающий контакт) при этом закрыт и подключает подтягивающее сопротивление к выводу PinXn, если ключ S2 замкнут посредством логической 1 на выходе регистра PortXn.
    - Если разряд “n” в регистре DDXn содержит сигнал высокого уровня, то соответствующий вывод сконфигурирован как выход. Ключ S3 в этом случае замкнут и отделяет регистр PortXn от вывода PinXn. Ключ S1 при этом разомкнут и отделяет подтягивающее сопротивление от вывода PinXn, независимо от положения ключа S2 и значения на выходе порта PortXn.
2. В регистр порта PortX (см. рис 3.2) в том случае, если вывод выполняет роль выхода, записывается значение, подлежащее выводу. В том случае, когда вывод выполняет роль входа, ключ S2 замкнут, и подтягивающее сопротивление может быть подключено к входному контакту.
3. Логический уровень на выводе PinXn может быть считан по третьему адресу. Схема управления в этом случае вырабатывает сигнал считывания RP (Read Pin – читать вывод), который через драйвер TP подключает вывод непосредственно к внутренней шине данных.

Все возможные конфигурации для направления передачи данных и подтягивающего сопротивления порта перечислены в табл. 3.2

Таблица 3.2 – Конфигурация направления передачи данных и подтягивающего сопротивления

| Регистр DDXn | Регистр PortXn | Ввод/ вывод | Подтягивающее сопротивление | Описание |
|---|---|---|---|---|
| 0 | 0 | Ввод | Нет | Высокоомный вход |
| 0 | 1 | Ввод | Да | Подтяг. сопротивление нагружает вход |
| 1 | 0 | Вывод | Нет | Двухтактный выход: лог. 0 |
| 1 | 1 | Вывод | Нет | Двухтактный выход: лог. 1 |

Регистр направления передачи данных и регистр порта доступны для чтения и записи, а информация на выводах аппаратного обеспечения может быть только считана. Для того чтобы выполнить операцию чтения схема управления выдает сигналы RD (Read Direction – читать направление), RL (Read Latch – читать фиксацию) и, соответственно, RP, которые через драйверы TD, TL, и TP передают соответствующее значение на внутреннюю шину данных.

Входной усилитель-формирователь порта микроконтроллера семейства AVR при низком уровне сигнала на выходе в состоянии принимать токи силой до 20 мА и благодаря этому, например, напрямую управлять светодиодами, подключенными к питающему напряжению.

Большинство портов альтернативно применяются для выполнения каких либо особых функций, например, для мультиплексированной шины адресов и данных внешней памяти RAM, для входов таймеров, прерывании, аналоговых компараторов, последовательных интерфейсов SPI, интерфейсов UART, выходов таймеров и т.д. Эти функции подробно описаны в руководстве того или иного модуля.

При пуске и перезапуске микроконтроллера все разряды регистров DDRx и PORTx сбрасываются в нулевое состояние, вследствие чего выводы работают в режиме входа и находятся в Z-состоянии (высокоимпедансном).

### 1.2 Таймеры

В большинстве МК AVR два или три таймера счетчика, один из которых 16-разрядный, а оставшиеся один или два 8-разрядные. Все счетчики имеют возможность предварительной загрузки значений и могут работать от тактовой частоты (Ф) процессора непосредственно, или поделенной на 8, 64, 256 или 1024 (в отдельных случаях еще на 16 и 32), а также от внешнего сигнала.

В архитектуре AVR 8-разрядным счетчикам таймерам присвоены номера 0 и 2, а 16-разрядным 1, 3 и далее. Некоторые 8-разрядные счетчики (обычно Timer2, если их два) могут работать в асинхронном режиме от отдельного тактового генератора, причем продолжать функционировать даже в случае «спящего» состояния всей остальной части МК, что позволяет использовать их в качестве часов реального времени.

При работе счетчиков таймеров, как обычных счетчиков внешних импульсов, частота подсчитываемых импульсов не должна превышать половины частоты тактового генератора МК. Это обусловлено тем, что при счете внешних импульсов их фронты обнаруживаются синхронно (в моменты положительного перепада тактового сигнала). Кроме того следует учитывать, что задержка обновления содержимого счетчика после прихода внешнего импульса может составлять до 2,5 периода тактовой частоты.

При переполнении счетчика возникает событие, которое может вызвать соответствующее прерывание. 8-разрядный счетчик Timer 0 в большинстве случаев этой функцией и ограничивается. Счетчик Timer 2 может также вызывать прерывание по совпадению подсчитанного значения с некоторой заранее заданной величиной. 16-разрядные счетчики могут вызывать прерывания по совпадению с двумя независимо заданными числами A и B. При этом счетчики могут обнуляться или продолжать счет, а на специальных выводах при этом – генерироваться импульсы (аппаратно, без участия программы).

Кроме того, 16-разрядные счетчики имеют возможность осуществлять «захват» (capture) внешних одиночных импульсов на специальном выводе. При этом может вызываться прерывание, а содержимое счетчика помещаться в

некий регистр. Сам счетчик может обнуляться и начинать счет заново или просто продолжать счет.

Все счетчики-таймеры допускают работу в т.н. режимах PWM, т.е. в качестве 8-, 9-, 10- или 16-битовых широтно-импульсных модуляторов (ШИМ), причем независимо друг от друга, что позволяет осуществлять многоканальный ШИМ.

В случае использования T/C0 и T/C1 в качестве таймера, используется в качестве тактового сигнала (напрямую или через делитель частоты) разделенный такт системной синхронизации Ф. Коэффициент делителя частоты может настраиваться индивидуально для каждого из двух таймеров с помощью мультиплексора.

На рисунке 3.3 показана схема управления таймером в микроконтроллерах базовой серии семейства AVR. В режиме работы “Таймер” тактовые частоты для T/C0 и T/C1 могут быть настроены для каждого отдельно с помощью предварительного делителя частоты и мультиплексоров. Для этой цели в микроконтроллер интегрирован 10-ступенчатый делитель $Q_0,\ldots,Q_9$, выводы которого $Q_2$ (Ф/8), $Q_5$ (Ф/64), $Q_7$ (Ф/256), $Q_9$ (Ф/1024), наряду с тактом системной синхронизации Ф, могут быть подключены напрямую через соответствующий мультиплексор Mux0 и, равным образом, Mux1 к тактовому входу T/C0 и, соответственно, T/C1.

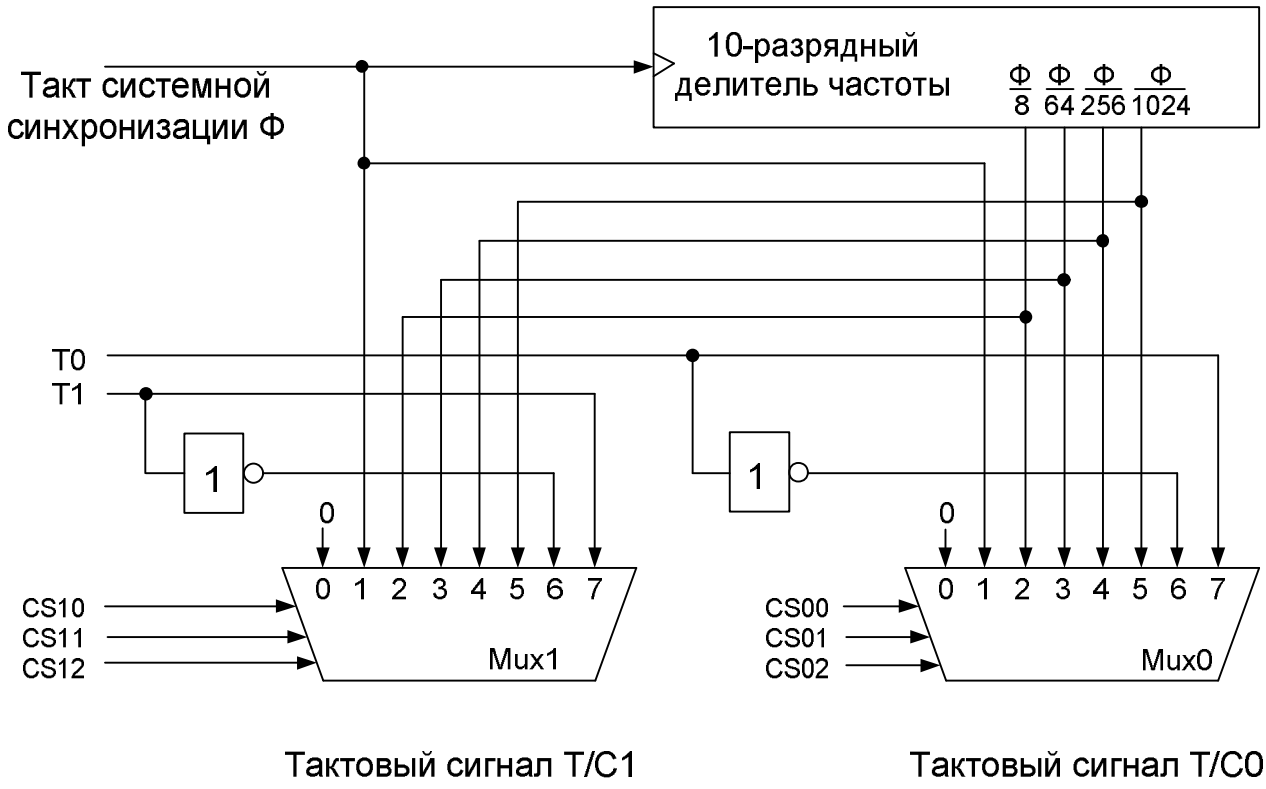



Рисунок 3.3 – Схема управления для подачи тактовых сигналов на таймер/счетчик микроконтроллеров базовой серии семейства AVR

Если требуется режим работы “Счетчик”, в качестве активного фронта с помощью мультиплексора может быть выбран ниспадающий или нарастающий фронт внешнего импульса на входах T0 и T1. При таком режиме работы внешний сигнал T0 (как и T1) синхронизируется с тактом системной синхронизации Ф внутреннего кварцевого осциллятора. Для этого внешний сигнал проверяется в течение каждого нарастающего фронта Ф.

Если в таймер/счетчике нет необходимости, то имеется возможность через мультиплексный адрес 0b00 установить на его тактовом входе лог.0 и таким образом остановить счет. В таблице 3.4 перечислены различные комбинации управляющих сигналов для мультиплексора Mux0 [Mux1] для выбора источника тактирования T/C0 [T/C1].

Таблица 3.4 – Выбор входного такта для T/C0 и T/C1 через регистр TCCR0

| CS02 [CS12] | CS01 [CS11] | CS00 [CS10] | Описание [значения в квадратных скобках относятся к T/C1] |
|---|---|---|---|
| 0 | 0 | 0 | Задержка T/C0 [T/C1] |

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | Режим “Таймер”, такт счета = Ф |
| 0 | 1 | 0 | Режим “Таймер”, такт счета = Ф/8 |
| 0 | 1 | 1 | Режим “Таймер”, такт счета = Ф/64 |
| 1 | 0 | 0 | Режим “Таймер”, такт счета = Ф/256 |
| 1 | 0 | 1 | Режим “Таймер”, такт счета = Ф/1024 |
| 1 | 1 | 0 | Режим “Счетчик”, такт счетчика = внешний импульс на выводе T0 [T1], активный ниспадающий фронт |
| 1 | 1 | 1 | Режим “Счетчик”, такт счетчика = внешний импульс на выводе T0 [T1], активный нарастающий фронт |

Восьмиразрядный таймер/счетчик T/C0

Восьмиразрядный таймер/счетчик T/C0 присутствует во всех микроконтроллерах базовой серии семейства AVR. Для T/C0 имеют значение только четыре регистра из области ввода/вывода (рис.3.4).

Кроме, собственно, счетного регистра TCNT0, используются:

- регистр управления TCCR0 для настройки мультиплексора;
- регистр флагов прерываний TIFR с флагом переполнения T/C0 TOV0;
- уведомление о переполнении TCNT0 (переход из состояния $FF в $00);
- регистр TIMSK для разрешения/запрета прерываний T/C0 с помощью флага TOIE0.

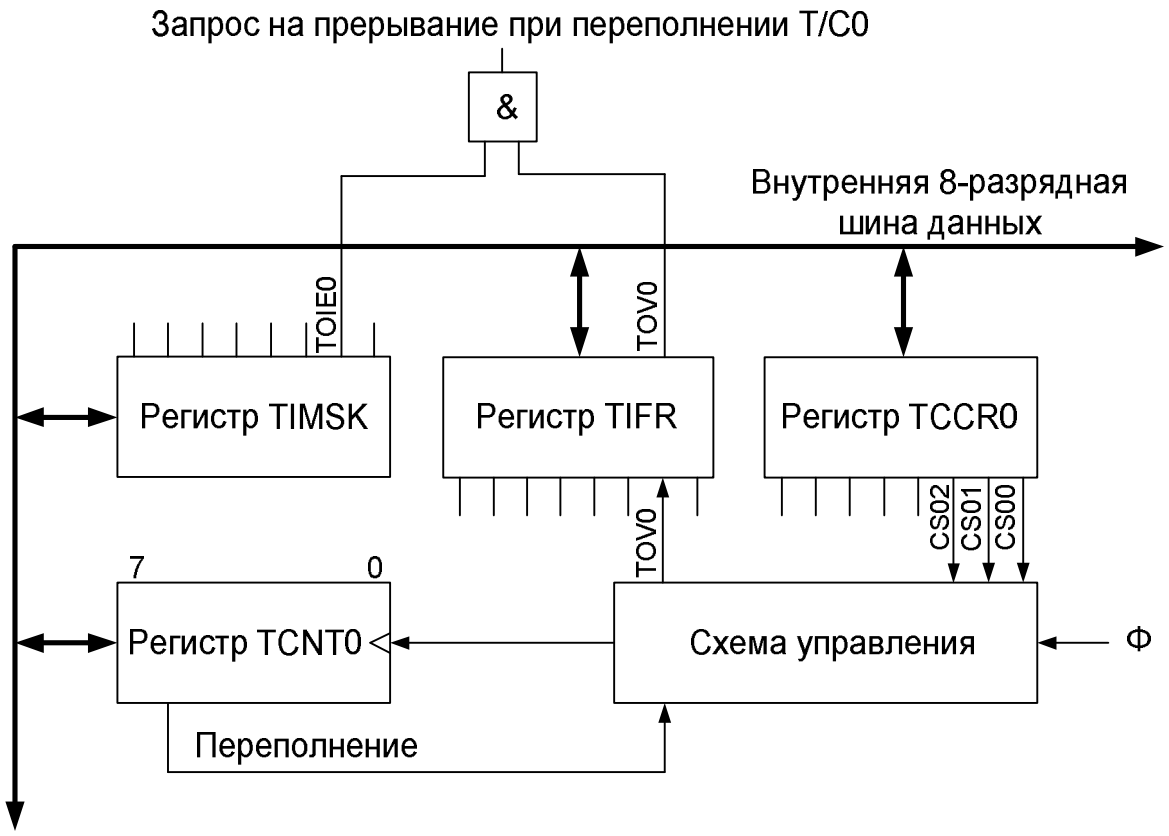



Рисунок 3.4 – Блок-схема таймера/счетчика T/C0

Как только с помощью разрядов CS00, CS01 и CS02 регистра управления TCCR0 для мультиплексора Mux0 будет выбран адрес, отличный от 0b000, T/C0 по каждому импульсу, поступающему на тактовый вход, начинает увеличивать на единицу содержимое регистра TCNT0.

Когда состояние счетчика в регистре TCNT0 изменится с $FF на $00, в регистре TIFR устанавливается флаг переполнения TOV0 таймера/счетчика T/C0. Для обработки этого переполнения у программиста есть две возможности.

- Постоянно опрашивать флаг TOV0 в цикле. Как только он установлен, требуемый промежуток времени истек, и выполнение программы может быть продолжено. Конечно, перед этим флаг TOV0 должен быть опять сброшен посредством записи лог. 1 в разряд 1 регистра TIFR.
- Когда будет установлен флаг I для глобального разрешения прерываний (разряд 7 в регистре состояния SREG), а также флаг TOIE0 для разрешения

прерываний от T/C0 (разряд 1 в регистре TIMSK), то после установки флага TOV0 будет возникать прерывание T/C0. Необходимо учитывать, что время от возникновения условия прерывания TOV0 до выполнения первой команды подпрограммы обработки прерывания, составляет минимум четыре тактовых цикла (сохранение адреса возврата в стеке и относительный переход к подпрограмме обработки прерывания). Если во время поступления запроса на прерывание выполняется какая-либо команда, для которой требуется более одного тактового цикла, то она в любом случае будет завершена. Это время также должно учитываться при определении начального значения регистра TCNT0.

Таймер/счетчик T/C0 очень хорошо подходит для оценки временных интервалов. Для этого в ходе выполнения программы в регистр TCNT0 записывается исходное значение. Затем может быть запущен таймер/счетчик T/C0 с требуемым тактом, выбранного с помощью мультиплексора Mux0. Программа или ожидает в цикле появление флага переполнения TOV0 в регистре TIFR (режим опроса), или разрешает прерывание T/C0. Флаг TOV0 указывает на то, что требуемое время задержки истекло, и выполнение программы может быть продолжено.

### 3.3 Память данных

Микроконтроллеры семейства AVR имеют память типа EEPROM, которая организована в виде отдельного запоминающего устройства. Запись и чтение могут выполняться по одному байту.

Память данных типа EEPROM предназначена для запоминания энергонезависимых данных и констант. Это очень удобно, к примеру, при калибровке измерительных приборов, работающих под управлением микроконтроллеров AVR, у которых в памяти EEPROM в процессе настройки сохраняются параметры корректировки.

Для программирования памяти EEPROM микроконтроллеров AVR нет необходимости применять внешнее программирующее устройство. Каждая

ячейка памяти EEPROM может быть запрограммирована непосредственно во время выполнения пользовательской программы.

Для программирования памяти используется три регистра памяти EEPROM: регистр адреса EEAR, регистр данных EEDR и регистр управления EECR. Во всех трех случаях речь идет о 8-разрядном регистре, исключением являются микроконтроллеры, объем EEPROM которых более 256 байт, регистр адреса у них делится на два – EEARH (старший байт) и EEARL (младший байт).

Регистр адреса EEAR памяти EEPROM имеет длину в один байт или два байта. Он расположен в области ввода/вывода по адресам $1E (младший байт) и $1F (старший байт). Эти байты доступны для чтения и записи. После подачи сигнала сброса они инициализируются нулями. Для программирования или чтения байта данных памяти EEPROM в регистр адреса EEAR должен быть записан соответствующий адрес.

Регистр данных EEDR памяти EEPROM находится в области ввода/вывода по адресу $1D. После подачи сигнала сброса он инициализируется нулями. Байт EEDR доступен для чтения и записи. В процессе записи в память EEPROM подлежащий программированию байт загружается в регистр EEDR. В процессе чтения из памяти EEPROM в регистр EEDR записывается содержимое соответствующей строки EEPROM. Адрес в памяти EEPROM в обоих случаях определяется по содержимому регистра EEAR.

Регистр управления EECR памяти EEPROM находится в области ввода/вывода по адресу $1C. После подачи сигнала сброса он инициализируется нулями.

Таблица 3.5 – Биты регистра EECR памяти EEPROM

| Разряд | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| $1C | - | - | - | - | EERIE | EEMWE | EEWE | EERE |

Разряд EERIE – разрешение прерывания от EEPROM. Данный разряд управляет генерацией прерывания, возникающего при завершении цикла записи в EEPROM. Если этот разряд установлен в «1», прерывания разрешены (если флаг I регистра SREG также установлен в «1»).

Для управления процессом чтения используется разряд EERE (EEPROM Read Enable – память EEPROM готова к чтению). После записи корректного адреса в регистр EEAR процесс чтения может быть активирован установкой разряда EERE в регистре управления. По окончанию считывания разряда EERE аппаратное обеспечение считывает требуемый байт в регистр EEDR.

Перед началом операции чтения программа пользователя должна постоянно опрашивать разряд EEWE и ждать появления лог.0. Если во время программирования памяти EEPROM в соответствующий регистр ввода/вывода памяти EEPROM будут записаны новые адреса или данные, то еще продолжающийся процесс программирования будет прерван и результат будет неопределенным.

Разряд EEWE (EEPROM Write Enable – память EEPROM готова к записи) – это разряд управления процессом записи. Для записи байта в память EEPROM разряд EEWE необходимо установить в лог. 1, если в регистре EEAR находится корректный адрес памяти EEPROM, а в регистре EEDR – байт данных, подлежащий программированию. Для предотвращения непреднамеренной записи в память EEPROM разряд может быть установлен только в том случае, если установлен также и разряд EEMWE.

Для программирования памяти EEPROM должны быть выполнены следующие действия.

1. Дождаться окончания процесса программирования памяти EEPROM (если он активен), то есть, пока разряд EEWE возвратится в состояние лог.0.
2. Записать новый адрес в регистр EEAR памяти EEPROM (1 или 2 байта).
3. Записать требуемый байт данных в регистр EEDR памяти EEPROM.

4. Установить разряд EEMWE в лог.1.
5. На протяжении следующих четырех периодов системного такта после установки разряда EEMWE в разряд EEWE должна быть записана лог.1. Тем самым будет запущен процесс программирования.

По окончанию процесса программирования разряд EEWE аппаратно автоматически сбрасывается в лог.0. Программа пользователя должна непрерывно опрашивать этот разряд, ожидая появления лог.0, прежде чем приступить к программированию следующего байта.

При записи одного байта в память EEPROM должен быть также установлен в лог.1 разряд EEMWE (EEPROM Master Write Enable – общее разрешение записи в память EEPROM). После того как разряд EEMWE установлен, уровень лог.1 в нем сохраняется в течение четырех периодов такта системной синхронизации, а затем будет выполнен аппаратный сброс в лог.0. Программа пользователя может программировать байт посредством записи лог.1 в разряд EEWE только на протяжении этих четырех тактов системной синхронизации. Если будет установлен разряд EEWE, но без установки также и разряда EEMWE, то процесс программирования начат не будет.

Пример подпрограмм записи и чтения памяти EEPROM.

```
EEWrite:                            ;Подпрограмма “Запись в EEPROM”
        sbic    EECR,EEWE           ;Если EEWE не лог. 0, то
        rjmp    EEWrite             ;       ожидать дальше
        ldi     Temp,High(AdrWr)    ;Старший байт адреса записи в EEPROM -
        out     EEARH,Temp          ;       в регистр адреса
        ldi     Temp,Low(AdrWr)     ; Младший байт адреса записи в EEPROM -
        out     EEARL,Temp          ;       в регистр адреса
        out     EEDR,EEdwr          ;байт данных в регистр данных
        sbi     EECR,EEMWE          ;разряд EEMWE разрешает программирование
        sbi     EECR,EEWE           ;Разряд EEWE установлен:
                                    ;       начало программирования
        ret

EERead:                             ; Подпрограмма “чтение EEPROM”
```

```
sbic    EECR, EEWE      ; Если EEWE не лог. 0, то
rjmp    EERead          ;       ожидать дальше
ldi     Temp,High(AdrRd) ; Старший байт адреса чтения EEPROM -
out     EEARH,Temp      ;       в регистр адреса
ldi     Temp,Low(AdrRd) ; Младший байт адреса чтения EEPROM -
out     EEARL,Temp      ;       в регистр адреса
sbi     EECR, EERE      ;Установить разряд EERE – начать процесс чтения
in      EEdrd, EEDR     ;Чтение байта данных
ret
```

### 3.4 Организация прерываний

В различных моделях AVR существует от 6-8 (младшие Tiny) до нескольких десятков прерываний (например, в ATmega2560 их 57). Все прерывания делятся на два вида: внутренние и внешние.

Внутренние прерывания возникают от любого устройства, которое является дополнительным по отношению к ядру системы: таймеров, аналогового компаратора, последовательного порта и т.п.

Внешних прерываний у большинства МК AVR два или три. Естественно, они могут возникать независимо друг от друга. Внешнее прерывание это событие, которое наступает при наличии сигнала на одном из входов, специально предназначенных для этого (например, INT0 и INT1).

Выделяют три вида событий, вызывающих такое прерывание, и их можно различить программно: это может быть низкий уровень напряжения, а также положительный или отрицательный фронт на соответствующем выводе.

Для разрешения прерываний используется бит I регистра флагов SREG, который разрешает (если установлен в логическую единицу) или запрещает (если установлен в логический ноль) аппаратные прерывания вообще. Для общего разрешения (запрещения) прерываний предусмотрены специальные команды sei (разрешить) и cli (запретить), устанавливающие этот бит в нужное состояние. По умолчанию бит I регистра флагов SREG сброшен.

Кроме общего флага прерываний, для каждого конкретного прерывания имеется свой разрешающий/запрещающий бит, расположенный в соответствующем регистре. Для таймеров это регистры TIMSK (бит TOIE0 – разрешение прерывания по переполнению таймера/счетчика T0, бит TOIE1 – разрешение прерывания по переполнению таймера/счетчика T1) или ETIMSK, для внешних прерываний – GIMSK, в последних моделях получивший название GIGR и т. п. При использовании прерываний эти биты необходимо устанавливать в состояние логической единицы, в противном случае автоматического вызова прерывания не произойдет.

Обработка прерываний осуществляется следующим образом. При возникновении прерывания флаг I регистра SREG аппаратно сбрасывается, запрещая обработку других прерываний. Когда обработка прерываний заканчивается (по команде reti) он автоматически устанавливается опять. При необходимости этот флаг можно «вручную» установить в подпрограмме обработчике (напрямую или командой sei), разрешив вложенные прерывания. После сброса флага I контроллер определяет, запрос на обработку какого именно прерывания произошел, - это делается по флагу конкретного прерывания, который также автоматически устанавливается при возникновении прерывания (например, для таймеров эти флаги находятся в регистре TIFR или ETIFR, для внешних прерываний – в регистре GIFR или EIFR и т.п.). Эти регистры при инициализации МК рекомендуется очищать.

После определения типа прерывания контроллер автоматически вычисляет адрес соответствующего вектора прерывания (векторы располагаются по начальным адресам памяти программ, например $002…$008 для МК семейства ATtiny). На этом месте должна располагаться команда rjmp (для контроллеров с памятью 16 кбайт и более – команда jmp), которая содержит адрес (вектор) процедуры-обработчика.

Перед тем как перейти по вектору прерывания, МК сбрасывает флаг прерывания и автоматически сохраняет содержимое счетчика команд в стеке. Заканчиваться процедура-обработчик должна командой выхода из прерывания

reti. Если во время обработки прерывания произошли еще какие-то прерывания, то их флаги окажутся установленными, и процедуры их обработки начнут выполняться незамедлительно, в том порядке, в каком они расположены в таблице векторов прерывания.

Если обработка прерываний может мешать выполнению каких-то длинных процедур, которые нельзя прерывать (например, во время записи в EEPROM), то прерывания запрещают командой cli.

В таблице 3.6 указаны примеры устройств, в которых расположены источники запросов прерываний, приведены в виде дроби имена маскирующих (маскирующий разряд показывает, что прерывание разрешено) и флажковых разрядов (флажковый разряд устанавливается в единичное состояние при появлении запроса прерывания) в числителе, и регистров ввода-вывода, в которых они расположены в знаменателе.

Таблица 3.6

| Устройство | Запрос прерывания | MASK | FLAG |
|---|---|---|---|
| CPU, WDT | RESET | | |
| Внешние | INT0<br>INT1 | INT0/GIMSK<br>INT1/GIMSK | INTF0/GIFR<br>INTF1/GIFR |
| | INT0-INT3<br>INT4-INT7 | INT0-INT3/EIMSK<br>INT4-INT7/EIMSK | INTF4-INTF7/EIFR |
| | I/O PINS<br>LLI PINS | PCIE/GIMSK<br>LLIE/ICR | PCIF/GIFR |
| T/C2 | T/C2 COMP<br>T/C2 OVF | OCIE2/TIMSK<br>TOIE2/TIMSK | OCF2/TIFR<br>TOV2/TIFR |
| T/C1 | T/C1 CAPT<br>T/C1 COMPA<br>T/C1 COMPB<br>T/C1 OVF | TICIE1/TIMSK<br>OCIE1A/TIMSK<br>OCIE1B/TIMSK<br>TOIE1/TIMSK | ICF1/TIFR<br>OCF1A/TIFR<br>OCF1B/TIFR<br>TOV1/TIFR |
| T/C0 | T/C0 COMP<br>T/C0 OVF | OCIE0/TIMSK<br>TOIE0/TIMSK | OCF0/TIFR<br>TOV0/TIFR |
| EEPROM | EE RDY | EERIE/EECR | |

***Лекция №4 «Специальные функции и система команд микроконтроллеров PIC и AVR»***

## I. Специальные функции.

Микроконтроллеры AVR имеют набор специальных функций расширяющих возможности системы.

Сброс.

Сброс всегда происходит при включении питания. Кроме этого, источниками сброса могут быть следующие события: аппаратный сброс, т.е. подача низкого уровня напряжения на вход RESET; окончание отсчета установленного интервала сторожевого таймера; срабатывание схемы BOD (Brown-out Detection). Значение четырех младших битов регистра состояния MCUCSR должно сигнализировать о том, от какого источника производился сброс предыдущий раз.

Встроенная схема BOD обеспечивает время срабатывания порядка микросекунд с задержкой на возврат в рабочее состояние после восстановления напряжения, определяющейся теми же установками, что и задержка сброса (ячейки CKSEL0 и SUT1..0). для выбора режима работы BOD служат три конфигурационные ячейки BODLEVEL2..0, имеющие следующие состояния:

111 (установка по умолчанию) – схема BOD выключена;

101 – включает BOD при пороге срабатывания 2,7 В;

100 – соответствует порогу 4,0 В.

Две конфигурационные ячейки SUT1..0 позволяют задать задержку сброса при подаче высокого уровня на вывод /RESET. Установленный с их помощью режим зависит еще от состояния ячеек CKSEL. Значения по умолчанию CKSEL=1 и SUT1..0=10, для кварцевого генератора с частотой 1 МГц и более оно будет соответствовать варианту, когда кварцу предоставляется время на «разгон» (16384 периодов тактовой частоты) плюс собственно задержка сброса, равная 4 мс.

Сторожевой таймер.

Основная функция сторожевого таймера – защита устройства от сбоев. Сторожевой таймер имеет независимый тактовый генератор и работает даже в режиме Power Down. Типовые значения частот этого генератора равны: 1 МГц при $V_{cc}$=5.0 В, 350 кГц при $V_{cc}$=3.0 В и 110 кГц при $V_{cc}$=2.0 В.

Структурная схема сторожевого таймера приведена на рисунке 4.1.

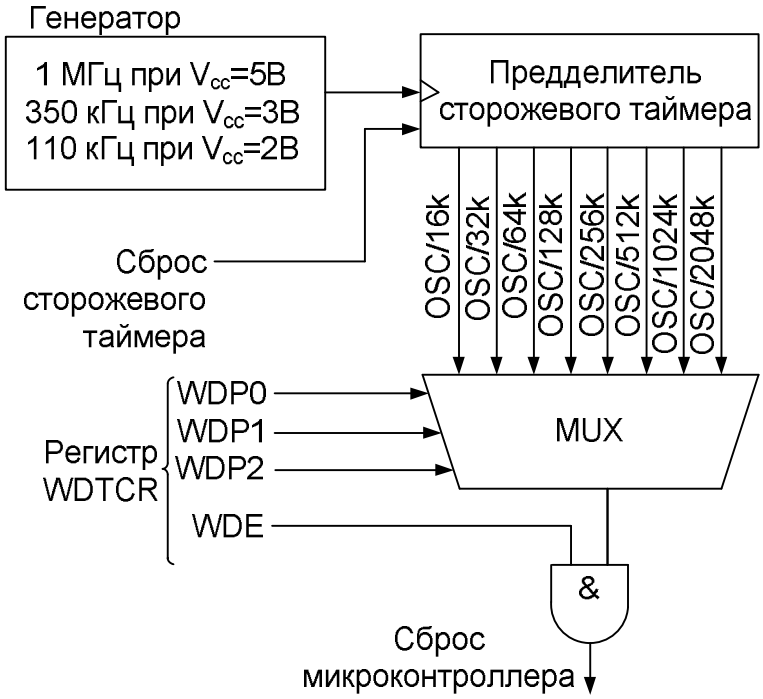



Рисунок 4.1 – Структурная схема сторожевого таймера

Если сторожевой таймер включен, то через определенные промежутки времени (при наступлении тайм-аута) выполняется сброс микроконтроллера. При нормальном выполнении программы сторожевой таймер должен периодически (через промежутки времени, меньшие его периода) сбрасываться командой WDR.

Для управления сторожевым таймером, предназначен регистр WDTCR (Watch Dog Timer Control Register). Формат этого регистра приведен в таблице 4.1.

Таблица 4.1 – Разряды регистра WDTCR

| Разряд | Название | Описание |
|---|---|---|
| 7…5 | - | Зарезервированы |
| 4 | WDTOE | Разрешение выключения сторожевого таймера |

| | | |
|---|---|---|
| 3 | WDE | Разрешение сторожевого таймера («1» - включен) |
| 2…0 | WDP2…WDP0 | Коэффициент деления предделителя сторожевого таймера |

Чтобы избежать непреднамеренного выключения сторожевого таймера, предназначен разряд WDTOE (Watch Dog Timer On Enable). Выключение сторожевого таймера (сброс разряда WDE) можно осуществить при установленном разряде WDTOE, который аппаратно сбрасывается через 4 машинных цикла после установки «1».

Режимы энергосбережения.

В различных моделях МК AVR имеется от 2-3 (семейства Classic и Tiny) до 5-6 (старшие Mega) режимов энергосбережения. Два базовых режима общие для всех моделей. Idle mode и Power Down mode. Режимы энергосбережения нельзя вызывать из процедуры прерывания – только из основной программы.

В Idle mode (режим ожидания) останавливается GPU (а также устройство управления выборкой команд из памяти). Все периферийные устройства – таймеры АЦП, порты – продолжают функционировать. Поэтому потребление снижается лишь на 30-50%.

В режиме Power Down mode останавливаются все узлы МК, за исключением сторожевого таймера, системы обработки внешних асинхронных прерываний и модуля TWI. Выход из этого режима возможен либо по сбросу МК, либо от прерывания TWI (иное наименование стандартизованного фирмой Philips интерфейса $I^2C$), либо от внешнего прерывания (причем, только того, которое обнаруживается асинхронно). Потребление в этом режиме может составить до нескольких десятков миллиампер. Особенность Power Down Mode – в этом режиме останавливается тактовый генератор. Это означает, что при выходе из спящего режима тактовый генератор потратит время на «раскрутку», причем этот интервал будет тем же самым, что задается конфигурационными ячейками SUT1..0 для задержки сброса.

Выбор типа генератора.

МК AVR могут тактироваться как от внешних источников (кварцевый (керамический) резонатор, RC-цепь), так и от внутреннего RC-генератора. Внутренний RC-генератор способен работать на четырех приблизительных значениях частот (1, 2, 4 и 8 МГц). В ряде моделей предусмотрена возможность подстройки частоты этого генератора.

Семейство Classic встроенного RC-генератора не имеет. По умолчанию МК семейств Tiny и Mega установлены в состояние работы со встроенным генератором на частоте 1 МГц (CKSEL=0001), поэтому для других режимов нужно соответствующим образом установить конфигурационные ячейки CKSEL (см табл. 4.2).

Таблица 4.2 – Установка конфигурационных ячеек CKSEL

| CKSEL | Источник тактирования | Частота |
|---|---|---|
| 0000 | Внешняя частота | 0…16 МГц |
| 0001 | Встроенный RC-генератор | 1 МГц |
| 0010 | Встроенный RC-генератор | 2 МГц |
| 0011 | Встроенный RC-генератор | 4 МГц |
| 0100 | Встроенный RC-генератор | 8 МГц |
| 0101 | Внешняя RC-цепочка | <0,9 МГц |
| 0110 | Внешняя RC-цепочка | 0,9…3,0 МГц |
| 0111 | Внешняя RC-цепочка | 3,0…8,0 МГц |
| 1000 | Внешняя RC-цепочка | 8,0…12 МГц |
| 1001 | Низкочастотный резонатор | 32,768 кГц |
| 101x | Кварцевый резонатор | 0,4…0,9 МГц |
| 110x | Кварцевый резонатор | 0,9…3,0 МГц |
| 111x | Кварцевый резонатор | 3,0…8,0 МГц |
| 1xxx(CKPOT=0) | Кварцевый резонатор | >1,0 МГц |

Защита от кода считываний.

Содержимое FLASH-памяти (памяти программ), а также содержимое EEPROM-памяти (память данных) может быть защищено от записи и/или

чтения посредством программирования ячеек защиты (Lock Bits) LB1 и LB2. Возможные режимы защиты: защита кода и данных отключена (LB1=1, LB2=1), последующая запись FLASH и EEPROM запрещена (LB1=0, LB2=1), запрещены запись и чтение FLASH и EEPROM (LB1=0, LB2=0).

В режимах 1 и 2 запрещается также изменение конфигурационных ячеек. Поэтому включение защиты следует выполнять в самую последнюю очередь, после программирования остальных областей памяти.

В исходном (незапрограммированном) состоянии во всех ячейках защиты содержится «1». Стирание ячеек (запись в них лог. 1) можно произвести только при выполнении команды «Стирание кристалла», уничтожающей также содержимое FLASH- и EEPROM-памяти.

Биты идентификации.

Все микроконтроллеры фирмы «Atmel» имеют три 8-разрядные ячейки, содержимое которых позволяет идентифицировать устройство. Ячейки идентификатора расположены в отдельном адресном пространстве, доступ к которому возможен только в режиме программирования. Эти ячейки доступны только для чтения. Их содержимое: код производителя, код объема FLASH-памяти, код устройства.

Последовательное программирование.

В общей сложности микроконтроллеры семейств Tiny и Mega поддерживают следующие режимы программирования: последовательное программирование при высоком напряжении (под «высоким» напряжением понимается управляющее напряжение 12 В, подаваемое на вывод /RESET микроконтроллера для перевода последнего в режим программирования), последовательное программирование при низком напряжении (по интерфейсу SPI), параллельное программирование при высоком напряжении, программирование по интерфейсу JTAG.

Режим программирования по последовательному каналу поддерживается всеми микроконтроллерами семейства Mega. В этом режиме программирование памяти программ и данных осуществляется по последовательному интерфейсу

SPI. Для подключения программатора к устройству используются три линии интерфейса: SCK (тактовый сигнал), MOSI (вход данных) и MISO (выход данных).

Биты конфигурации.

Конфигурационные ячейки (Fuse Bits) определяют некоторые параметры конфигурации микроконтроллера. Эти ячейки расположены в отдельном адресном пространстве, доступном только при программировании. Все конфигурационные ячейки сгруппированы в несколько байтов (от одного до трех в зависимости от модели), а состав этих ячеек зависит от конкретной модели микроконтроллера.

Краткое назначение всех конфигурационных ячеек приведено в табл. 4.3

Таблица 4.3 – Назначение конфигурационных ячеек

| **Название** | **Назначение** |
|---|---|
| FSTRT | Определяет длительность задержки сброса $t_{TOUT}$ |
| RSTDISBL | Определяет функционирование вывода микроконтроллера, совмещенного с выводом аппаратного сброса (0 - контакт порта ввода/вывода, 1 – вывод сброса) |
| CKSEL | Определяют режим работы тактового генератора, а также длительность задержки сброса $t_{TOUT}$ |
| BODLEVEL | Определяет порог срабатывания схемы BOD |
| BODEN | Разрешает/запрещает функционирование схемы BOD (0 – разрешено, 1 - запрещено) |
| SPIEN | Разрешает/запрещает программирование по интерфейсу SPI (0 – разрешено, 1 - запрещено) |
| INTCAP | Определяет состояние внутренних конденсаторов, продключаемых между выводами XTAL1, XTAL2 и общим проводом (0 – подключены, 1 - отключены) |
| SUT | Определяет длительность задержки сброса $t_{TOUT}$ |
| WDTON | Определяет режим сторожевого таймера (0 – всегда включен, 1 – может быть включен программно) |
| CKPOT | Определяет функционирование тактового генератора, действие зависит от установок ячеек CKSEL |

| | |
|---|---|
| EESAVE | Определяет влияние команды «Стирание кристалла» на EEPROM-память (0 – не стирает, 1 - стирает) |
| BOOTSZ | Определяет размер секции загрузчика |
| BOOTRST | Определяет положение вектора сброса |
| S8515C | Включает/выключает режим совмести с микроконтроллерами AT90S4414/8515 семейства Classic (0 – включен, 1 - выключен) |
| OCDEN | Разрешает/запрещает внутрисхемную отладку (0 – разрешена, 1 - запрещена) |
| JTAGEN | Разрешает/запрещает использование интерфейса JTAG (0 – разрешен, 1 - запрещен) |
| CDIV16 | Определяет начальное состояние делителя системного тактового сигнала |
| CKOUT | Определяет состояние выходного буфера системного тактового сигнала (0 – подключен к выводу микроконтроллера, 1 - отключен) |
| M161C | Включает/выключает режим совмести с микроконтроллерами ATmega161xx семейства Mega (0 – включен, 1 - выключен) |
| M103C | Включает/выключает режим совмести с микроконтроллерами ATmega103x семейства Mega (0 – включен, 1 - выключен) |

Для изменения содержимого конфигурационных ячеек используются специальные команды программирования. Команда «Стирание кристалла» на состояние этих ячеек не влияет.

## II. Система команд

### 2.1 Перечень и формат команд

Всего для AVR насчитывается от 90 до 133 команд (в зависимости от контроллера)

Все множество команд микроконтроллеров AVR можно разбить на несколько групп: арифметические и логические команды, команды операций с битами, команды сравнения, команды передачи управления, команды переноса данных и команды управления системой.

В таблицах 4.4…4.9 указаны команды, которыми располагают микроконтроллеры семейств Tiny и Mega. В таблицах приведены основные сведения о командах, такие как мнемоническое обозначение команды, ее

описание, число машинных циклов, необходимых для ее выполнения, а также флаги регистра SREG, на которые воздействует эта команда. При использовании команд следует обращать внимание на то, что некоторые из них могут быть применены только к определенным регистрам, а константы или адреса иногда имеют ограниченный диапазон.

В таблицах приняты следующие обозначения:

РОН – регистр общего назначения, обозначается Rd (приемник) или Rr (источник), где d или r – номер регистра;

РВВ – регистр ввода-вывода обозначается P;

PC – счетчик команд (программный счетчик, Program Counter);

K – константа (в том числе адрес);

b или n – номер бита;

s – произвольный флаг в регистре флагов SREG;

c – флаг переноса в регистре флагов SREG (устанавливается при возникновении переноса при арифметических операциях);

z – флаг нуля (устанавливается по равенству операндов при сравнении);

X – пара регистров R27:R26;

Y – пара регистров R29:R28;

Z – пара регистров R31:R30;

A – означает, что участвует любой из двубайтовых регистров R27:R26 (X), R29:R28 (Y) или R31:R30 (Z).

Таблица 4.4 – Арифметические и логические команды

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|
| ADD Rd,Rr | Сложение двух РОН без учета переноса | Rd←Rd+Rr<br>d=0..31; r=0..31 | 1 | Z,C,N,V,H |
| ADC Rd,Rr | Сложение двух РОН с учетом переноса | Rd←Rd+Rr+c<br>d=0..31; r=0..31 | 1 | Z,C,N,V,H |
| ADIW Rd,K | Сложение регистровой пары с константой | Rd+1:Rd←Rd+1:Rd+K<br>d=24,26,28,30<br>K=0..63 | 2 | Z,C,N,V,S |
| SUB Rd,Rr | Вычитание двух РОН без учета переноса | Rd←Rd-Rr<br>d=0..31; r=0..31 | 1 | Z,C,N,V,H |
| SBC Rd,Rr | Вычитание двух РОН с учетом переноса | Rd←Rd-Rr-c<br>d=0..31; r=0..31 | 1 | Z,C,N,V,H |
| SBIW Rd,K | Вычитание константы из | Rd+1:Rd←Rd+1:Rd-K | 2 | Z,C,N,V,S |

| | | | | |
|---|---|---|---|---|
| | регистровой пары | d=24,26,28,30<br>K=0..63 | | |
| SUBI Rd,K | Вычитание константы из регистра | Rd←Rd-K<br>d=16..31; K=0..255 | 1 | Z,C,N,V,H |
| SBCI Rd,K | Вычитание константы из регистра с учетом переноса | Rd←Rd-K-c<br>d=16..31; K=0..255 | 1 | Z,C,N,V,H |
| INC Rd | Увеличить на единицу | Rd←Rd+1<br>d=0..31 | 1 | Z,N,V |
| DEC Rd | Уменьшить на единицу | Rd←Rd-1<br>d=0..31 | 1 | Z,N,V |
| CLR Rd | Очистить регистр (операция «исключающее или» регистра с самим собой) | Rd←Rd ⊕Rd<br>d=0..31 | 1 | Z,N,V |
| SER Rd | Установить регистр | Rd←$FF<br>d=0..31 | 1 | - |
| AND Rd,Rr | Логическое И | Rd←Rd∧Rr<br>d=0..31; r=0..31 | 1 | Z,N,V |
| ANDI Rd,K | Логическое И с константой | Rd←Rd∧K<br>d=16..31; K=0..255 | 1 | Z,N,V |
| OR Rd,Rr | Логическое ИЛИ | Rd←Rd∨Rr<br>d=0..31; r=0..31 | 1 | Z,N,V |
| ORI Rd,K | Логическое ИЛИ с константой | Rd←Rd∨K<br>d=16..31; K=0..255 | 1 | Z,N,V |
| EOR Rd,Rr | Логическое исключающее ИЛИ | Rd←Rd⊕Rr<br>d=0..31; r=0..31 | 1 | Z,N,V |
| COM Rd | Побитная инверсия | Rd←$FF-Rd<br>d=0..31 | 1 | Z,C,N,V |
| NEG Rd | Дополнительный код (инверсия знака) | Rd←$00-Rd<br>d=0..31 | 1 | Z,C,N,V,H |

Таблица 4.5 – Команды операций с битами

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|
| SBR Rd,K | Установить биты РОН по маске (то же, что и команда ORI) | Rd←Rd∨K<br>d=16..31; K=0..255 | 1 | Z,N,V |
| CBR Rd,K | Сбросить биты РОН по маске | Rd←Rd∧($FF-K)<br>d=16..31; K=0..255 | 1 | Z,N,V |
| SBI P,b | Установить бит в РВВ | P(b) ←1<br>P=0..31; b=0..7 | 2 | - |
| CBI P,b | Очистить бит в РВВ | P(b) ←0<br>P=0..31; b=0..7 | 2 | - |
| LSL Rd | Логический сдвиг влево (с установкой переноса) | Rd(n+1) ←Rd(n),<br>Rd(0) ←0,c←Rd(7)<br>d=0..31 | 1 | Z,C,N,V |
| LSR Rd | Логический сдвиг вправо (с установкой переноса) | Rd(n) ←Rd(n+1),<br>Rd(7) ←0,c←Rd(0)<br>d=0..31 | 1 | Z,C,N,V |

| | | | | |
|---|---|---|---|---|
| ROL Rd | Логический сдвиг влево через перенос | Rd(0) ←c,<br>Rd(n+1) ←Rd(n),<br>c←Rd(7)<br>d=0..31 | 1 | Z,C,N,V |
| ROR Rd | Логический сдвиг вправо через перенос | Rd(7) ←c,<br>Rd(n) ←Rd(n+1),<br>c←Rd(0)<br>d=0..31 | 1 | Z,C,N,V |
| ASR Rd | Арифметический сдвиг вправо | Rd(n)←Rd(n+1),<br>n=0,…,6 | 1 | Z,C,N,V |
| CLI | Запретить все прерывания (очистить флаг прерывания I в SREG) | I←0 | 1 | I |
| SEI | Разрешить все прерывания (установить флаг прерывания I в SREG) | I←1 | 1 | I |
| CLC | Очистить бит переноса c | c←0 | 1 | C |
| SEC | Установить бит переноса c | c←1 | 1 | C |
| BCLR s | Очистить флаг s в регистре SREG | SREG(s) ←0 | 1 | SREG(s) |
| BSET s | Установить флаг s в регистре SREG | SREG(s) ←1 | 1 | SREG(s) |
| SWAP Rd | Перестановка тетрад | Rd(3..0) ←Rd(7..4),<br>Rd(7..4) ←Rd(3..0) | 1 | - |

Таблица 4.6 – Команды сравнения

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|
| CP Rd,Rr | Сравнение двух регистров | Rd-Rr | 1 | Z,N,V,C,H |
| CPC Rd,Rr | Сравнение двух регистров с учетом переноса | Rd-Rr-c | 1 | Z,N,V,C,H |
| CPI Rd,K | Сравнение регистра с константой | Rd-K | 1 | Z,N,V,C,H |
| TST Rd | Проверка на 0 или отрицательное значение (операция «логическое ИЛИ» регистра с самим собой) | Rd∧Rd | 1 | Z,N,V |

Таблица 4.7 – Команды передачи управления

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|
| **Команды безусловного перехода и вызова подпрограмм** | | | | |
| CALL K | Абсолютный вызов подпрограммы | STACK←PC+2<br>PC←K<br>K=0..655361 | 4 | - |
| RCALL K | Относительный вызов подпрограммы | STACK←PC+1<br>PC←PC+K+1<br>K= -2048..2048 | 3 | - |

| | | | | |
|---|---|---|---|---|
| JMP K | Абсолютный переход | PC←K<br>K=0..655361 | 3 | - |
| RJMP K | Относительный переход | PC←PC+K+1<br>K= -2048..2048 | 2 | - |
| RET | Возврат из подпрограммы | PC←STACK | 4 | - |
| RETI | Возврат из подпрограммы обработки прерывания | PC←STACK | 4 | I |
| *Для устройств с максимально возможным объемом памяти программ до 64 К слов (128 кбайт) | | | | |
| **Команды проверка-пропуск и команды условного перехода** | | | | |
| SBRC Rr,b | Пропустить следующую команду, если разряд РОН сброшен | Если Rr(b)=0 то PC←PC+2(3)<br>r=0..31; b=0..7 | 1/2/3 | - |
| SBRS Rr,b | Пропустить следующую команду, если разряд РОН установлен | Если Rr(b)=1 то PC←PC+2(3)<br>r=0..31; b=0..7 | 1/2/3 | - |
| SBIC P,b | Пропустить следующую команду, если разряд РВВ сброшен | Если A(b)=0 то PC←PC+2(3)<br>P=0..31; b=0..7 | 1/2/3 | - |
| SBIS P,b | Пропустить следующую команду, если разряд РВВ установлен | Если A(b)=1 то PC←PC+2(3)<br>P=0..31; b=0..7 | 1/2/3 | - |
| CPSE Rd,Rr | Пропустить, если равно | Если Rd=Rr то PC←PC+2(3)<br>d=0..31; r=0..31 | 1/2/3 | - |
| BRNE K | Перейти, если не равно | Если z=1 то PC←PC+k+1<br>K= -64..63 | 1/2 | - |
| BREQ K | Перейти, если равно | Если z=0 то PC←PC+k+1<br>K= -64..63 | 1/2 | - |
| BRSH K | Перейти, если больше или равно | Если c=0 то PC←PC+k+1<br>K= -64..63 | 1/2 | - |
| BRLO K | Перейти, если меньше | Если c=1 то PC←PC+k+1<br>K= -64..63 | 1/2 | - |
| BRCC K | Перейти, если нет переноса | Если c=0 то PC←PC+k+1<br>K= -64..63 | 1/2 | - |
| BRBS s,K | Перейти, если флаг в SREG установлен | Если s=1 то PC←PC+k+1<br>s=0..7; K= -64..63 | 1/2 | - |
| BRBC s,k | Перейти, если флаг в SREG очищен | Если s=0 то PC←PC+k+1<br>s=0..7; K= -64..63 | 1/2 | - |

Таблица 4.8 – Команды переноса данных

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| MOV Rd,Rr | Перенос данных между РОН | Rd←Rr<br>d=0..31; r=0..31 | 1 | - |
| LDI Rd,K | Загрузка константы в РОН | Rd←K<br>d=16..31; K=0..255 | 1 | - |
| LD Rd,A | Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в A | Rd←(A)<br>d=0..31 (исключая A) | 2 | - |
| LD Rd,A+ | Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в A, с постинкрементом адреса | Rd←(A), A=A+1<br>d=0..31 (исключая A)<br>A=X,Y,Z | 2 | - |
| LD Rd,-A | Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в A, с преддекрементом адреса | A=A-1, Rd←(A)<br>d=0..31 (исключая A)<br>A=X,Y,Z | 2 | - |
| ST A,Rr | Запись значения в память данных (SRAM) по адресу, содержащемуся в A | (A) ←Rd<br>r=0..31 (исключая A)<br>A=X,Y,Z | 2 | - |
| ST A+,Rr | Запись значения в память данных (SRAM) по адресу, содержащемуся в A с постинкрементом адреса | (A) ←Rd, A=A+1<br>r=0..31 (исключая A)<br>A=X,Y,Z | 2 | - |
| ST –A,Rr | Запись значения в память данных (SRAM) по адресу, содержащемуся в A с преддекрементом адреса | A=A-1, (A) ←Rd<br>r=0..31 (исключая A)<br>A=X,Y,Z | 2 | - |
| LPM | Загрузка данных из памяти программ в регистр R0 по адресу (байтовому), находящемуся в регистре Z | R0←(Z) | 3 | - |
| IN Rd,P | Загрузка значения PBB в РОН | Rd←(P)<br>r=0..31; P=0..63 | 1 | - |
| OUT P,Rr | Вывод значения РОН в PBB | (P) ←Rr<br>r=0..31; P=0..63 | 1 | - |
| PUSH Rr | Сохранить значение РОН в стеке | STACK←Rr<br>r=0..31 | 2 | - |
| POP Rd | Извлечь значение верхушки стека в РОН | Rd←STACK<br>d=0..31 | 2 | - |

Таблица 4.9 – Команды управления системой

| **Мнемоника** | **Описание** | **Операция** | **Циклы** | **Флаги** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| NOP | Нет операции | - | 1 | - |
| SLEEP | Переход в «спящий» режим | - | 3 | - |
| WDR | Сброс сторожевого таймера | - | 1 | - |

Команды логических операций позволяют выполнять стандартные логические операции над байтами, такие как, логическое умножение (И), логическое сложение (ИЛИ), операцию «исключающее ИЛИ». Операции производятся между регистрами общего назначения либо между регистром и константой. Результат сохраняется в РОН. Все логические команды выполняются за один машинный цикл.

Команды арифметических операций позволяют выполнять такие базовые операции, как сложение, вычитание, инкремент и декремент. В микроконтроллерах семейства Mega также имеются команды, позволяющие осуществлять умножение 8-разрядных значений. Все операции производятся только над регистрами общего назначения. При этом микроконтроллеры AVR позволяют оперировать как знаковыми, так и беззнаковыми числами, а также работать с числами, представленными в дополнительном коде. Почти все команды арифметических операций выполняются за один машинный цикл. Команды умножения и команды, оперирующие двухбайтовыми значениями, выполняются за два цикла.

К командам операций с битами относятся команды, выполняющие установку или сброс заданного разряда РОН или РВВ. Следует помнить, что в командах CBR и SBR операндом является битовая маска, а не номер разряда. Этими командами можно за один раз установить хоть все биты в регистре – в этом отличие команд sbr/cbr от sbi/cbi. Также к группе битовых операций относят команды сдвига.

В командах сравнения с регистрами выполняются те же действия, что и соответствующих арифметических и логических операциях, однако результат никуда не помещается (и, соответственно, операнды не портятся), лишь устанавливаются флаги в регистре флагов. Значением этих флагов в

дальнейшем определяется работа тех команд условного перехода, которые употребляются в паре с командами сравнения.

Команды передачи управления делятся на команды безусловного перехода и похожие на них команды вызова подпрограмм (последние от первых отличаются тем, что автоматически размещают в стеке содержимое счетчика команд для последующего возврата из подпрограммы), и на команды условного перехода, т.е. нарушения последовательности выполнения операторов по какому-то условию. Большинство таких команд оперируют с адресом в памяти оператора, на который производится переход. В тесте ассемблерных программ абсолютные или относительные числа, обозначающие адрес, в команды передачи управления не подставляются, вместо них указывают метки, которые затем компилятор интерпретирует, как абсолютный адрес. Команды предполагают предварительный вызов одной из команд, модифицирующих флаги z или c (обычно это команды сравнения).

Команды пересылки данных переносят данные из одной области памяти в другую (память рассматривается в широком смысле этого слова, и в нее включаются также регистры). Команда ldi загружает в регистр непосредственно число константу (но действует только для регистров, начиная с r16). Команда mov загружает в регистр значение другого регистра. Команды загрузки и чтения SRAM – ld и st. С этими командами связаны такие понятия как «прямая» и «косвенная» адресация.

Во всех случаях чтения и записи SRAM требуются регистры X, Y и Z - т.е. пары r27:r26, r29:r28 и r31:r30 соответственно. Если обмен данными производится между памятью и другим регистром общего назначения, то достаточно одной из этих пар, если же между областями памяти – целесообразно задействовать две. Режимы с преддекрементом и постинкрементом используются, когда нужно прочесть/записать целый фрагмент из памяти.

Команда переноса данных lpm позволяет прочесть произвольный байт их памяти программ. Хранение в памяти тех констант, которые никогда не будут

изменяться, прямо рекомендуется разработчиками AVR, т.к. существует проблема безопасного хранения данных в EEPROM.

Команд управления системой всего три: nop (no operation, пустая команда), sleep (перевод МК в режим энергосбережения) и wdr – сброс сторожевого таймера.

Операция nop служит для заполнения ячеек памяти пустыми значениями, если это зачем-то требуется: например, чтобы выровнять адрес процедуры по определенному адресу в памяти. Встретив эту команду, процессор выполнит единственную операцию инкрементирования содержимого счетчика команд.

При использовании команды sleep предварительно режим энергосбережения должен быть разрешен и, если потребуется, установлен его тип. Если энергосбережение не разрешено, то команда sleep нечего не делает (и притом сработает, только если она вызывается из основной программы, а не из прерывания).

## I. Особенности программирования на AVR

Микроконтроллеры AVR имеют гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 8-битных регистра общего назначения, объединенных в регистровый файл. В отличие от «идеального» RISC, регистры не абсолютно ортогональны:

Три «сдвоенных» 16-битных регистра-указателя X (r26:r27), Y (r28:r29) и Z (r30:r31)

Некоторые команды работают только с регистрами r16…r31.

Результат умножения (в тех моделях, в которых есть модуль умножения) всегда помещается в r0:r1

## II. Разработка программного кода

Как известно, наиболее эффективные программы получаются при использовании языка Ассемблер Для микроконтроллеров семейства AVR имеется свободно распространяемый транслятор ассемблера — wavrasm. Также одновременно с транслятором ассемблера устанавливается программа для отладки программ на языке ассемблера для микроконтроллеров семейства AVR. Однако она заметно уступает свободно распространяемому фирмой Atmel отладчику AVR Studio.

Транслятор ассемблера работает в среде Microsoft Windows 3.11, Microsoft Windows 95/98 и NT. Кроме того, имеется версия ассемблера, работающая из командной строки MS-DOS. Она устанавливается одновременно с версией для Windows. Версия для Windows имеет встроенный полноэкранный текстовый редактор и справочную систему на английском языке.

### 2.1 AVR Ассемблер фирмы Atmel

Программа на языке ассемблера представляется в виде одного или нескольких текстовых файлов. Файл состоит из одного или нескольких сегментов. Программа может иметь в своем составе сегменты трех типов — CSEG, ESEG и DSEG. Сегмент типа CSEG является обязательным.

Запись программы выполняется построчно. Строка может содержать до 120 символов (буквы, цифры, знаки, пробелы) или может быть пустой.

В строке выделяются три поля:

- метки;

- мнемокод;

- комметнарии.

В процессе ассемблирования мнемокоды коды команд, директивы ассемблера и метки используются при формировании программы в машинных кодах.

**Метки**

Поле для записи метки находится слева. Метка представляет собой последовательность из букв, цифр и некоторых знаков и начинается с буквы. Запись метки заканчивается знаком ":". Поле метки может быть пустым.

**Мнемокод**

Поле для записи мнемокодов команд и директив ассемблера находится справа от поля метки. Поле может быть пустым.

Ниже в таблице приведены операции, выполняемые по командам базовой системы, и приведены мнемокоды команд

| № | Операция | Мнемокод команды | Признаки результата | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | H | S | V | N | Z | C |
| 1 | $00→$R_d$(($R_d$)⊕ ($R_d$)→$R_d$) | CLR $R_d$ | | 0 | 0 | 0 | 1 | |
| 2 | ($R_d$)→$R_d$(($R_D$)^($R_d$)→$R_d$) | TST $R_d$ | | + | 0 | + | + | |
| 3 | $\overrightarrow{R_d}$→$R_d$($FF-($R_d$)→$R_d$) | COM $R_d$ | | + | 0 | + | + | 1 |
| 4 | -($R_d$)→$R_d$($00-($R_d$)→$R_d$) | NEG $R_d$ | + | + | + | + | + | + |
| 5 | ($R_d$)+1→Rd | INC $R_d$ | | + | + | + | + | |
| 6 | ($R_d$)-1→$R_d$ | DEC $R_d$ | | + | + | + | + | |
| 7 | 0→($\overrightarrow{R_d}$)→C | LSR $R_d$ | | + | + | 0 | + | + |
| 8 | C←($\overrightarrow{R_d}$)←0 | LSL $R_d$ | + | + | + | + | + | + |
| 9 | C→($\overrightarrow{R_d}$)→C | ROR $R_d$ | | + | + | + | + | + |
| 10 | C← ($\overrightarrow{R_d}$)←C | ROL $R_d$ | + | + | + | + | + | + |
| 11 | $R_d$.7→($R_d$.6-$R_d$.0)→C | ASR $R_d$ | | + | + | + | | |
| 12 | $R_d$.4-7↔$R_d$.0-3 | SWAP $R_d$ | | | | | | |
| 13 | (Rr)→$R_d$ | MOV $R_d$, $R_r$ | | | | | | |
| 14 | ($R_d$)+($R_r$)→$R_d$ | ADD $R_d$, $R_r$ | + | + | + | + | + | + |
| 15 | ($R_d$)+($R_r$)+C→$R_d$ | ADC $R_d$, $R_r$ | + | + | + | + | + | + |
| 16 | (Rd)-($R_r$)→Rd | SUB $R_{d>}$ $R_r$ | + | + | + | + | + | + |
| 17 | ($R_d$)-(Rr)-C→-$R_d$ | SBC $R_d$, $R_r$ | + | + | + | + | + | + |
| 18 | (Rd)-(Rr) | CP $R_d$, $R_r$ | + | + | + | + | + | + |
| 19 | ($R_d$)-($R_r$)-C | CPC $R_d$, $R_r$ | + | + | + | + | + | + |
| 20 | (Rd)^(Rr)→Rd | AND $R_d$, $R_r$ | | + | 0 | + | + | + |
| 21 | ($R_d$)v($R_r$)→$R_d$ | OR $R_d$, $R_r$ | | + | 0 | + | + | |
| 22 | ($R_d$)⊕($R_r$)→$R_d$ | EOR $R_d$, $R_r$ | | + | 0 | + | + | |

d = 0—31; r = 0—31

| № | Операция | Мнемокод команды | Признаки результата | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | H | S | V | N | Z | C |
| 23 | $FF→$R_d$ | SER $R_d$ | | | | | | |
| 24 | K→$R_d$ | LDI $R_d$, K | | | | | | |
| 25 | ($R_d$) - K→$R_d$ | SUBI $R_d$, K | + | + | + | + | + | + |
| 26 | ($R_d$)-K-C→$R_d$ | SBCI $R_d$, K | + | + | + | + | + | + |
| 27 | ($R_d$) – K | CPI $R_d$, K | + | + | + | + | + | + |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 28 | $(R_d) \wedge K \rightarrow R_d$ | ANDI $R_d$, K | | + | 0 | + | + | |
| 29 | $(R_d) \wedge \overline{K} \rightarrow R_d$ | CBR $R_d$, K | | + | 0 | + | + | |
| 30 | $(R_d) \vee K \rightarrow R_d$ | ORI $R_d$, K | | + | 0 | + | + | |
| 31 | $(R_d) \vee K \rightarrow R_d$ | SBR $R_d$, K | | + | 0 | + | + | |

d = 16—31 (!);K = 0—255

| № | Операция | Мнемокод команды | Признаки результата | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | H | S | V | N | Z | C |
| 32 | $(R_{d+1}, R_d)+K \rightarrow R_{d+1}, R_d$ | ADIW $R_d$, K | | + | + | + | + | + |
| 33 | $(R_{d+1}, R_d)-K \rightarrow R_{d+1}, R_d$ | SBIW $R_d$, K | | + | + | + | + | + |

d = 24, 26, 28, 30; K = 0—63

| № | Операция | Мнемокод команды | | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|---|
| 34 | (Ячк)→$R_d$ | LDS $R_d$, k | | 34 | $(R_r)$→Ячк | STS k, $R_r$ |

d, r = 0—31; к — адрес из адресного пространства SRAM

| № | Операция | Мнемокод команды | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|
| 36 | (Яч(X))→$R_d$ | LD $R_d$, X | 37 | $(R_r)$→Яч(X) | ST X, $R_r$ |
| 38 | (Яч(Y))→$R_d$ | LD $R_d$, Y | 39 | $(R_r)$→Яч(Y) | ST Y, $R_r$ |
| 40 | (Яч(Z))→$R_d$ | LD $R_d$, Z | 41 | $(R_r)$→Яч(Z) | ST Z, $R_r$ |
| 42 | 1 (Яч(X))→$R_d$<br>2. (X)+1→X | LD $R_d$, X+ | 43 | 1. $(R_r)$→Яч(X)<br>2. (X)+1→X | ST X+, $R_r$ |
| 44 | 1 (Яч(Y))→$R_d$<br>2. (Y)+1→Y | LD $R_d$, Y+ | 45 | 1. $(R_r)$→Яч(Y)<br>2. (Y)+1→Y | ST Y+, $R_r$ |
| 46 | 1 (Яч(Z))→$R_d$<br>2. (Z)+1→Z | LD $R_d$, Z+ | 47 | 1. $(R_r)$→Яч(Z)<br>2. (Z)+1→Z | ST Z+, $R_r$ |
| 48 | 1. (X)-1→X<br>2. (Яч(X))→$R_d$ | LD $R_d$, -X | 49 | 1. (X)-1-X<br>2. $(R_r)$-fl4(X) | ST -X, $R_r$ |
| 50 | 1. (Y)-1→Y<br>2. (Яч(Y))→$R_d$ | LD $R_d$, -Y | 51 | 1. (Y)-1-*Y<br>2. $(R_r)$-fl4(Y) | ST -Y, $R_r$ |
| 52 | 1. (Z)-1→Z<br>2. (Яч(Z))→$R_d$ | LD $R_d$, -Z | 53 | 1. (Z)-1→Z<br>2. $(R_r)$→ Яч(Z) | ST -Z, $R_r$ |
| 54 | (Яч(Y)+q)→$R_d$ | LDD $R_d$, Y+**q** | 55 | $(R_r)$ → Яч(Y)+**q** | STD Y+**q,** $R_r$ |
| 56 | (Яч(Z)+q)→$R_d$ | LDD $R_d$, Z+**q** | 57 | $(R_r)$ → Яч(Z)+**q** | STD Z+**q,** $R_r$ |
| 58 | 1. (SP)+1→SP<br>2. (CTEK) →$R_d$ | POP $R_d$ | 59 | 1. $(R_r)$ →CTEK<br>2. (SP)-1 →SP | PUSH $R_r$ |

d, r - 0—31; q - 0—63

| № | Операция | Мнемокод команды | | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|---|
| 60 | (РгР)→$R_d$ | IN $R_d$, P | | 34 | $(R_r)$→ РгР | OUT P, $R_r$ |

d, r = 0—31; P = 0—63 = $00—$3F

| № | Операция | Мнемокод команды | |
|---|---|---|---|
| 62 | (Яч(Z)) →R0 | LPM | (Z.15-1) - адрес в FlashROM<br>Z.0 = 0 - мл. байт; Z.0 = 1 — ст. байт |

| № | Операция | Мнемокод команды | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|
| 63 | T→$R_d$.b | BLD $R_d$, b | 64 | $R_r$.b→T | BST $R_r$, b |
| 65 | 0→PrP.b | CBI P, b | 66 | 1→PrP.b | SBI P, b |
| 67 | 0→SREG.b | BCLR b | 68 | 1→SREG.b | BSET b |

d, r - 0—31; P =0—31 (!); b = 0—7

| № | Операция | Мнемокод команды | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|
| 69 | 0→I | CLI | 70 | 1→I | SEI |
| 71 | 0→T | CLT | 72 | 1→T | SET |
| 73 | 0→H | CLH | 74 | 1→H | SEH |
| 75 | 0→S | CLS | 76 | 1→S | SES |
| 77 | 0→V | CLV | 78 | 1→V | SEV |
| 79 | 0→N | CLN | 80 | 1→N | SEN |
| 81 | 0→Z | CLZ | 82 | 1→Z | SEZ |
| 83 | 0→C | CLC | 84 | 1→C | SEC |

| № | Операция | Мнемокод команды | № | Операция | Мнемокод команды |
|---|---|---|---|---|---|
| 85 | (PC)+1+k→PC | RJMP k | 86 | (Z) →PC | IJMP |
| 87 | 1. (PC)+1→CTEK<br>2. (SP)-2→SP | RCALL k | 88 | 1. (PC)+1→CTEK<br>2. (SP)-2→SP | ICALL |

| | | | | | |
|---|---|---|---|---|---|
| | 3. (PC)+1+k→PC | | | 3. (Z) →PC | |
| 89 | 1. (SP)+2→SP<br>2. (CTEK) →PC | RET | 90 | 1. (SP)+2→SP<br>2. (CTEK) →PC<br>3. 1→1 | RETI |

k = -2048 — +2047

| № | Условие | Мнемокод команды | № | Условие | Мнемокод команды |
|---|---|---|---|---|---|
| 91 | I = 0 | BRID k | 92 | I = 1 | BRIE k |
| 93 | T = 0 | BRTC k | 94 | T = 1 | BRTS k |
| 95 | H = 0 | BRHC k | 96 | H = 1 | BRHS k |
| 97 | S = o | BRGE k | 98 | S = 1 | BRLT k |
| 99 | V = o | BRVC k | 100 | V = 1 | BRVS k |
| 101 | N = 0 | BRPL k | 102 | N = 1 | BRMI k |
| 103 | Z = 0 | BRNE k | 104 | Z = 1 | BREQ k |
| 105 | C = 0 | BRCC k | 106 | C = 1 | BRCS k |
| 107 | C = 0 | BRSH k | 108 | C = 1 | BRLO k |
| 109 | SREG.b = 0 | BRBC b, k | 110 | SREG.b=1 | BRBS b, k |

к = -64 — +63; b =0-7

| № | Условие | Мнемокод команды | № | Условие | Мнемокод команды |
|---|---|---|---|---|---|
| 111 | $R_r$.b = 0 | SBRC $R_r$, b | 112 | $R_r$.b = 1 | SBRS $R_r$, b |
| 113 | PrP.b = 0 | SBIC P, b | 114 | PrP.b = 1 | SBIS P, b |
| 115 | ($R_d$) = ($R_r$) | CPSE $R_d$, $R_r$ | | | |

d, г = 0-31; P =-31 (!); b=0-7

| № | Операция | Мнемокод команды |
|---|---|---|
| 116 | МК→режим энергосбережения | SLEEP |
| 117 | Перезапуск WDT | WDR |
| 118 | Нет | NOP |

## Директивы

Директивы ассемблера выполняют следующие функции:

- определяют тип сегмента (директивы .CSEG, .ESEG, .DSEG);

- распределяют память микроконтроллера (директивы .ORG, .DB, .DW, .BYTE);

- присваивают имена и значения (директивы .DEF, .EQU, .SET);

- управляют процессом ассемблирования (директивы .DEVICE, .INCLUDE, .EXIT, .MACRO, .ENDMACRO);

- управляют формированием листинга (директивы .NOLIST, .LIST, .LISTMAC).

Сегмент тип CSEG начинается с директивы .CSEG. Если запись программы начинается с сегмента этого типа, директива может отсутствовать. В сегменте типа CSEG записываются мнемокоды команд и могут записываться все директивы ассемблера, кроме директивы .BYTE.

Метка является символическим адресом в адресном пространстве FlashROM. Реальный адрес вычисляется в процессе ассемблирования. Исходное значение задается с помощью директивы .ORG, записанной в начале сегмента после директивы CSEG. В программе могут находиться несколько сегментов типа CSEG. При отсутствии директивы .ORG в первом из них в качестве исходного принимается нулевой адрес в FlashROM.

В сегменте типа CSEG директива .DB определяет байт или группу байтов, которые должны быть записаны в FlashROM в качестве констант, начиная с адреса, обозначенного меткой перед директивой.

*Пример 1.*

*.CSEG .ORG $140 NUM: .DB $12, $34, $56, $78*

*Байт $12 записывается в FlashROM по адресу NUM = $140 в младшую половину ячейки, байт $34 — по этому же адресу в старшую половину ячейки, байт $56 — по адресу NUM+1 = $141 в младшую половину ячейки, байт $78 — по адресу NUM+1 в старшую половину ячейки. Запись выполняется при программировании микроконтроллера.*

Директива .DW определяет слово или группу слов, которые должны быть записаны в FlashROM в качестве констант, начиная с адреса, обозначенного меткой перед директивой.

*Пример 2.*

*NUMS: .DW $1234, $5678, $9ABC*

*Слово $1234 записывается в FlashROM по адресу NUMS, слово $5678 — по адресу NUMS+1, слово $9ABC — по адресу NUMS+2. Реальные адреса вычисляются в процессе ассемблирования. Запись в FlashROM выполняется при программировании микроконтроллера.*

Директива .DEF присваивает символическое имя регистру общего назначения. Это имя может указываться в мнемокодах команд вместо стандартного обозначения *($R_d$, $R_T$)*.

*Пример 3.*

*.DEF TEMP = R16 LDI TEMP, $F0*

*В регистр общего назначения Rl 6 заносится байт $F0.*

Директивы .EQU и .SET присваивают значение имени, введенному программистом.

*Пример 4.*

*.EQU DDRCINIT - $3B*

*LDI TEMP, DDRCINIT OUT $14. TEMP*

*Коду, который должен быть занесен в регистр DDRC (№$14), программист присвоил имя DDRCINIT. При выполнении программы в регистр DDRC будет*

*занесен двоичный код 00111011, который определяет направление передачи через выводы порта С.*

*Пример 5.*

*.DEF MASTEST = R20 . EQU KTEST = 6*

*SBRC MASTEST. KTEST*

*Выполнение команды, условного перехода (команда №111) зависит от состояния шестого разряда в регистре общего назначения R20.*

*Пример 6.*

*. EQU STACKINIT = $025F*

*Коду, предназначенному для занесения в регистр-указатель стека, программист присвоил имя и задал значение.*

*Значение, присвоенное директивой .EQU, остается неизменным для всей программы. Значение, присвоенное директивой .SET, может быть изменено в другом месте программы.*

Директива .DEVICE определяет тип микроконтроллера, для которого составляется программа. По этой директиве в ассемблере выбирается служебный файл, в котором присвоены имена регистрам ввода-вывода и их разрядам, указана емкость запоминающих устройств и отмечены другие особенности микроконтроллера используемого типа.

*Пример 7:*

*.DEVICE AT90S8515*

Директива .INCLUDE указывает имя дополнительного файла, который должен быть использован в процессе ассемблирования. Пример записи директивы:

*.INCLUDE "UART.ASM",*

*где UART.ASM — имя дополнительного файла.*

Директива .EXIT указывает конец используемой при ассемблировании части файла, если не требуется использовать весь файл.

Директивы .MACRO и .ENDMACRO обрамляют участок программы (макроопределение), который должен ассемблироваться каждый раз, когда в поле для мнемокодов команд и директив ассемблера появляется имя, заданное директивой .MACRO.

Директива LIST MAC, записанная перед именем, которое было указано в директиве .MACRO, включает в листинг команды, обрамленные директивами MACRO и ENDMACRO, и помечает их знаком "+".

Директива .DB определяет байт или группу байтов, которые должны быть записаны в ячейки EEPROM, начиная с адреса, обозначенного меткой перед директивой .DB.

*Пример 8.*

*.ESEG .ORG $20*

*.EMAS:.DB $02. $13, $24*

*Байт $02 будет записан в EEPROM по адресу EMAS = $20, байт $13 — по адресу EMAS+1 = $21, байт $24 — по адресу EMAS+2. Запись в EEPROM выполняется в процессе программирования микроконтроллера.*

Директива .DW определяет слово или группу слов, которые должны быть записаны в пары ячеек EEPROM, начиная с адреса, обозначенного меткой перед директивой .DW.

*Пример 9.*

*. ESEG*

*EMASW:.DW $1357, $2468*

*Слово $1357 будет записано в EEPROM побайтно по адресам EMASW и EMASW+1, слово $2468 - по адресам EMASW+2 и EMASW+3.*

Директива .BYTE определяет количество ячеек в SRAM, к которым будет производиться обращение для записи и чтения байтов, начиная с адреса, обозначенного меткой перед директивой .BYTE.

*Пример 10.*

*.DSEG .ORG $60 DMAS:.BYTE 3 .CSEG*

*MST: STS DMAS+1, R5*

*В SRAM зарезервированы три ячейки, к которым обращаются по адресам DMAS = $60, DMAS+1 = $61 и DMAS+2 = $62.*

*По команде, записанной в FlashROM по адресу MST, выполняется пересылка байта из регистра общего назначения R5 в ячейку SRAM по адресу $61.*

**Операнды**

Могут быть использованы следующие операнды:

- определенные программистом метки, имеющие значение счетчика, в зависимости от места своего расположения;

- переменные, определенные с помощью директивы SET;

- константы, определенные с помощью директивы EQU;

- целые константы:

а) десятичные (по умолчанию)

б) шестнадцатеричные (два вида записи)

в) двоичные

Пример записи числа:
75 = $4B = 0X4B = 0b0l00l0l1.

- коды символов ASCII: 'A', 'a';

- строки ASCII (без нуля в конце строки): «String»;

- PC — текущее значение счетчика команд в памяти программ.

**Функции**

LOW (выражение) — возвращает младший байт выражения;

HIGT(выражение) — возвращает старший байт выражения;

BYTE2(выражение) — возвращает 2 байта выражения;

BYTE3(выражение) — возвращает 3 байта выражения;

BYTE4(выражение) — возвращает 4 байта выражения;

LWRD(выражение)— возвращает биты 0—15 выражения;

HWRD(выражение) — возвращает биты 16—31 выражения;

PAGE(выражение) — возвращает биты 16—21 выражения;

EXP2(выражение) — возвращает $2^{Л}$ выражения;

LOG2(выражение).

**Операции**

Ассемблер поддерживает различные операторы, описанные ниже. При их использовании можно применять скобки.

*Логическое НЕ*

Обозначение: !

Описание: унарный оператор, возвращает 1, если выражение равно нулю, и 0, если выражение было не равно нулю. Приоритет: 14.

Пример: ldi r16,!0xf0 ; Загрузить в П6 0x00

*Побитовое НЕ*

Обозначение: ~

Описание: унарный оператор, который возвращает исходное выражение со всеми инвертированными битами. Приоритет: 14.

Пример: ldi r16.~0xf0 ; Загрузить в r16 OxOf

*Унарный минус*

Обозначение: —

Описание: возвращает число с измененным на противоположный знаком.

*Умножение*

Обозначение: *

Описание: возвращает результат умножения двух чисел. Приоритет: 13.

Пример: ldi r30,label*2 ; Загрузить в регистр r30 label*2

*Деление*

Обозначение: /

Описание: возвращает целую часть от деления левого параметра на правый.

Приоритет: 13.

Пример: ldi r30, label/2 ; Загрузить в регистр r30 label/2 .

*Сложение*

Обозначение: +

Описание: возвращает сумму двух чисел. Приоритет: 12.

Пример: ldi r30,c1+c2 Загрузить в регистр r30 c1+c2

*Вычитание*

Обозначение: -

Описание: возвращает результат вычитания правого числа из левого. Приоритет: 12.

Пример: ldi r17,c1-c2 ; Загрузить в регистр r30 c1-c2

*Сдвиг влево*

Обозначение: «

Описание: возвращает значение левого числа, сдвинутое влево на число раз, равное правому числу. Приоритет: 11.

Пример: ldi r17,1«3 ; Загружает в регистр r17 число 1,

; сдвинутое влево на 3 бита

*Сдвиг вправо*

Обозначение: »

Описание: возвращает значение левого числа, сдвинутое вправо на число раз, равное правому числу. Приоритет: 11.

Пример: ldi r17,1»2 ; Загружает в регистр r17 число 1, ; сдвинутое вправо на 2 бита

*Меньше*

Обозначение: <

Описание: возвращает 1, если первое число меньше второго, иначе — 0.

Приоритет: 10.

Пример: ori r18,bitmask*(c1<c2)+1

*Меньше или равно*

Обозначение: <=

Описание: возвращает 1, если первое число меньше второго или равно ему, иначе — 0. Приоритет: 10.

Пример: ori r18,bitmask*(c1<=c2)+1

*Больше*

Обозначение: >

Описание: возвращает 1, если первое число больше второго, иначе – 0.

Приоритет: 10.

Пример: ori r18,bitmask*(c1>c2)+1

*Больше или равно*

Обозначение: >=

Описание: возвращает 1, если первое число больше второго или ! равно ему, иначе — 0.

Приоритет: 10.

Пример: ori r18,bitmask«(c1>=c2)+1

*Равно*

Обозначение: =

Описание: возвращает 1, если первое число равно второму, иначе—0.

Приоритет: 9

Пример: andi r19, bitmask*(c1==c2)+1

*Не равно*

Обозначение: !=

Описание: возвращает 1, если первое число не равно второму, [ иначе — 0.

Приоритет: 9.

Пример: .SET flag=(c1=c2)

*Побитовое И*

Обозначение: &

Описание: возвращает результат побитной операции «И» между операндами.

Приоритет: 8.

Пример: ldi r18,High(c1&c2)

*Побитовое исключающее ИЛИ*

Обозначение: ^

Описание: возвращает результат побитной операции «исключающее ИЛИ» между операндами. Приоритет: 7.Пример: ldi r18,Low(c1^c2)

*Побитовое ИЛИ*

Обозначение: |

Описание: возвращает результат побитной операции «ИЛИ» между операндами.

Приоритет: 6.

Пример: ldi r18,Low(c1|c2)

*Логическое И*

Обозначение: &&

Описание: возвращает 1, если оба выражения не равны нулю, иначе — 0.

Приоритет: 5.

Пример: ldi r18,Low(c1&&c2)

*Логическое ИЛИ*

Обозначение: ||

Описание: возвращает 0, если оба выражения равны нулю, иначе — 0.

Приоритет: 4.

Пример: ldi r18, Low(c1||c2)

**Коментарии**

В конце  находится поле для записи комментария. Запись комментария начинается со знака ";". Поле комментария может быть пустым. Комментарии игнорируются. Они используются программистами в качестве пояснений к программе.

**Средства разработки**

Существуют следущие средства разработки для AVR:

AVRStudio — IDE + ассемблер + отладчик страница AVRStudio

IAR Embedded Workbench for Atmel AVR — компилятор C/C++ сайт разработчика

CodeVisionAVR — компилятор C + генератор начального кода сайт разработчика

ICC AVR — компилятор C сайт разработчика

AtmanAvr — компилятор C + отладчик + генератор начального кода сайт разработчика

E-LAB AVRco — компилятор Pascal

Algorithm Builder - визуальная среда разработки программ для AVR в виде блок-схем включает также эмулятор и программатор. Используемый язык программирования - псевдоассемблер.

ForthInc Forth-Compiler компилятор языка Forth

MPE Forth-Compiler компилятор языка Forth

VMLAB — симулятор AVR

Proteus — симулятор AVR

AVReal — программатор подключение LPT, совместим с CodeVisionAVR сайт разработчика

PonyProg — программатор подключение COM port (LPT) поддерживает МК AVR, PIC и др сайт разработчика

**Транслятор ассемблера WAVRASM**

Теоретически нет ограничений на количество одновременно открытых файлов исходных текстов. Размер каждого файла не должен превышать примерно 28 Кб. Для работы с файлами большего размера следует использовать версию ассемблера, работающую из командной строки MS-DOS — avrasm. Также можно разбить всю программу на несколько файлов и объединить их с помощью директивы INCLUDE.

Для каждого открытого файла создается окно с его текстом.

Для создания нового файла следует выполнить команду меню File»New (быстрая комбинация клавиш: Alt-F N). Для открытия существующего файла следует выполнить команду меню File»Open (быстрая комбинация клавиш: Alt-F O).

*Перемещение по тексту программы*

Для перемещения по тексту программы можно пользоваться следующими командами:

вправо — стрелка вправо; влево — стрелка влево; вверх — стрелка вверх; вниз — стрелка вниз; в начало строки — Home; в конец строки — End; в начало файла — Ctrl+Home; в конец файла — Ctrl+End.

*Редактирование текста*

Для редактирования текста следует пользоваться клавишами:

— вставить пробел — пробел;

— завершить строку — Enter;

— удалить символ слева от курсора — Backspace;

— удалить символ справа от курсора — Del.

Для разбиения строки на две следует установить курсор на место разбиения и нажать Enter.

Для объединения двух строк следует установить курсор в начало второй строки и нажать клавишу Backspace.

Выделение текста, операции копирования, перемещения и удаления осуществляются так же, как в любой программе для Windows.

*Установка опций программы*

Некоторые установки транслятора ассемблера могут быть изменены. Для этого следует выполнить команду меню Options.

В этом окне можно установить расширение файла, содержащего листинг программы и файла с оттранслированным кодом. Менять их не рекомендуется.

Также здесь можно указать, какого типа должен генерироваться выходной файл. Имеется три типа файлов: Generic, Motorola S-record и Intel HEX.

Обратите внимание, что объектный файл (который используется отладчиком) всегда имеет расширение obj. Также, если в программе инициализируются значения в памяти EEPROM, генерируется файл с расширением eep, используемый программатором для прошивки в микроконтроллер в процессе программирования. Этот файл генерируется в формате Generic.

Опция Wrap relative jumps — разрешить относительную адресацию переходов. Эта опция полезна для использования с микроконтроллерами, имеющими 4 К слов памяти программ.

Опция Save before assemble — сохранять исходный текст программы каждый раз перед ее ассемблированием.

**Пакет Project-AVR**

Пакет Project-AVR - набор программно-аппаратных средств, предназначенный для разработки и отладки систем на базе микроконтроллеров семейства AVR фирмы Atmel.

Концепция Project-AVR - объединение внутрисхемного эмулятора, программного отладчика-симулятора, компиляторов, текстового редактора, менеджера проектов и программатора в рамках единой интеллектуальной среды разработки.

При наличии одного из программаторов PicProg+, ChipProg, ChipProg+ пакет поддерживает работу и с программатором. Программный интерфейс пакета унифицирован и поддерживает все этапы разработки программного обеспечения - от написания исходного текста программы до ее компиляции и отладки.

Полная конфигурация пакета называется Project-AVR/ESA и включает в себя:

- менеджер проектов;
- кросс-компиллятор языка ассемблер MCA-AVR;
- отладчик-симулятор PDS-AVR;
- внутрисхемный эмулятор PICE-AVR.

**Отладчик-симулятор микроконтроллеров семейства AVR**

PDS-AVR - это интегрированный комплекс профессиональных средств для разработки систем на базе семейства микроконтроллеров AVR фирмы Atmel, включающий среду разработки, макроассемблер, отладчик-симулятор, примеры программ и проектов, мощную систему контекстной помощи, электронные гипертекстные руководства по всем компонентам пакета, а также краткое руководство пользователя в печатном виде. PDS-AVR работает в среде Windows-95/98/ME/NT/2000/XP.

С помощью PDS-AVR можно эффективно разрабатывать и отлаживать программы, используя не только входящий в комплект макроассемблер MCA-AVR,

но и кросс-средства фирм IAR Systems и ImageCraft Creations, для которых также предоставляется возможность разработки программ на уровне ведения проектов. Пользователю предоставляется обширный сервис по выполнению отлаживаемой программы в различных режимах, манипуляции различными типами точек останова, просмотру и модификации состояния ресурсов микроконтроллера. Поддерживается отладка программ по исходному тексту, а также просмотр и изменение значений сложных объектов языка высокого уровня - массивов, структур, указателей.

Среда разработки программ PDS-AVR интегрирует в себе средства, используемые при разработке программ для микроконтроллеров AVR. Обеспечивается интерактивная поддержка всех этапов разработки от написания исходного текста до зашивки готовой программы в ПЗУ микроконтроллера, а именно:

- написание исходных текстов программ с помощью встроенного многооконного редактора;
- настройка опций кросс-средств, используемых для компиляции программы (ассемблера, компилятора Си, линкера, библиотекаря). Настройка производится с помощью диалогов, снабженных контекстной справочной информацией;
- компиляция и линковка программы. Если компилятор обнаруживает ошибки в исходном тексте программы, то строка с ошибкой в окне редактора подсвечивается и ошибки можно сразу же исправить;
- отладка программы;
- "зашивка" программы в ПЗУ микроконтроллера.

"Интегрированность" среды PDS-AVR проявляется в том, что перечисленные этапы разработки связываются в одно целое. Самые трудоемкие этапы, а именно компиляция/линковка с диагностикой и исправлением ошибок, максимально упрощены. PDS-AVR самостоятельно следит за изменениями, которые Вы вносите в исходные тексты своих программ. Например, исправив ошибку в исходном тексте, Вы можете нажатием одной кнопки "выполнить программу до курсора" заставить PDS-AVR перетранслировать изменившиеся модули, загрузить полученную программу в память отладчика и запустить ее до указанной строки. Переход от отладки к редактированию происходит так же прозрачно и быстро.

Возможности PDS-AVR:

- отслеживание выполнения программы по ее исходному тексту;
- просмотр и изменение значений любых переменных;
- встроенный анализатор эффективности программного кода;
- точки останова по сложному условию;
- неограниченное количество точек останова по доступу к ячейкам памяти;
- просмотр стека вызовов подпрограмм и функций;
- встроенный строчный ассемблер;
- возможность выполнения программы "назад" на большое количество шагов, а также в непрерывном режиме. При этом состояние модели микроконтроллера полностью восстанавливается;
- точный подсчет интервалов времени и многое другое.

**AVR Studio**

Для программирования AVR-микроконтроллеров существует немало средств разработки, однако, наиболее популярным, несомненно, следует признать пакет AVR Studio. Есть ряд причин такой популярности – это бесплатный пакет, разработанный фирмой ATMEL, он объединяет в себе текстовый редактор, ассемблер и симулятор. Пакет AVR Studio также используется совместно с аппаратными средствами отладки.

При программировании в среде AVR Studio надо выполнить стандартную последовательность действий:

- создание проекта
- загрузка файла
- компиляция
- симуляция
- загрузка hex-кода в микроконтроллер

Создание проекта начинается с выбора строки меню Project\New Project. В открывшемся окне «Create new Project» указывается имя проекта и имя файла инициализации. После нажатия кнопки «Next» открывается окно «Select debug platform and device», где выбирается отладочная платформа (симулятор или эмулятор) и тип микроконтроллера.

Можно выбрать один из предлагаемых внутрисхемных эмуляторов, заметим, что у каждого эмулятора свой список поддерживаемых микросхем. После нажатия

кнопки «Finish» предстают рабочие окна пакета AVR Studio, пока пустые. В первое окно помещается исходный текст программы. Это можно сделать двумя способами, либо набрать весь текст непосредственно в окне редактора, либо загрузить уже существующий файл.

Компиляция проекта производится командой \Project\Build или нажатием кнопки F7. Процесс компиляции отображается в окне «Output», расположенной в \View\Output.

Пакет AVR Studio содержит мощные средства для просмотра и редактирования состояния внутренних регистров и портов ввода/вывода отлаживаемого микроконтроллера, а также время, выполнения программы. Доступ к ним осуществляется через окно “I/O”.

Есть возможность просмотра программы в пошаговом режиме и видеть изменение текущих состояний этих регистров в поле Bits. Есть возможность оперативного изменения состояния любого бита регистров порта, причем это можно делать либо записью нового кода в поле Value, либо непосредственно, щелкнув мышью на нужном бите регистра.

Еще одна из многих характеристик пакета AVR Studio - возможность подключения внешних программ. Например, для обеспечения вызова оболочки внутрисхемного программатора AS2 нужно выполнить следующие операции:

- в меню Tools главного окна AVR Studio надо выбрать пункт Customize;
- в окне Customize выбрать пункт Tools;
- двойным нажатием кнопки мыши или нажав Insert на клавиатуре, добавить новую команду в список и назвать ее "Программатор AS2";
- указать путь к исполняемому файлу программатора, введя его непосредственно в поле для ввода "Command", или нажав на кнопку "…" справа от этого поля;
- теперь в меню Tools появился пункт "Программатор AS2".

# Лекция №7 Макет микропроцессорной системы и программирование простейших задач для микроконтроллеров AVR

Макет схемы для программы для программ:

1. Считывания кнопок и вывода на светоидный индикатор при определенных условиях.
2. Программы для борьбы с дребезгом контактов

Макет схемы представлен на рисунке 7.1

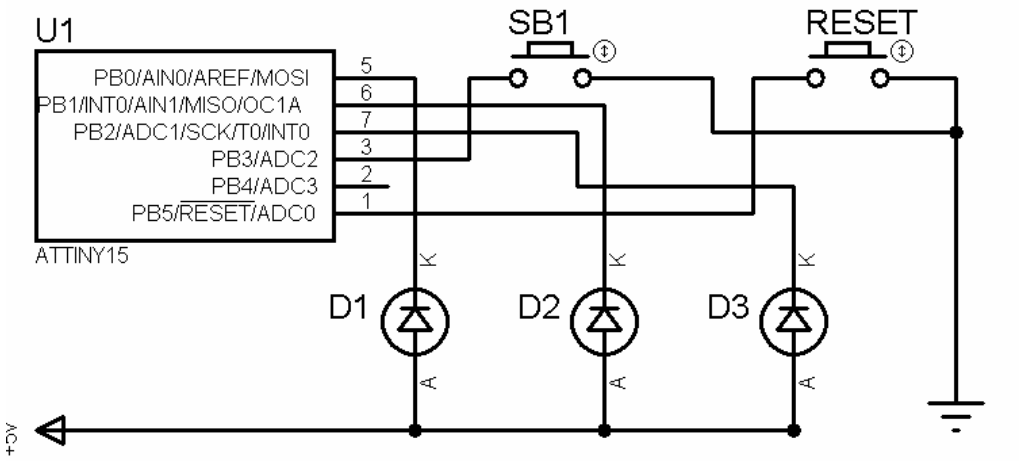



Рис. 7.1 Макет схемы

Три светоида D1, D2, D3 подключены к портам PB0, PB1, PB2 микроконтроллера AtTiny15. Кнопки SB1 и RESET к портам PB3 и $\overline{RESET}$ соответственно. В схеме применяется аппаратный сброс. При нажатие кнопки SB1 программа начинает свое выполнение, три светоида работают как двоичный счетчик увеличивающий свое значение при каждом нажатии кнопки SB1. Данная функциональность реализуется следующей программой:

```
.include "tn15def.inc"          ;Подключения файла с определением констант
                                ;и адрессов для tiny15
.def temp = r18

CLR  temp                       ;Отчистка регистра для правильной
                                ;обработки аппаратного сброса

RJMP Reset                      ;Вектор сброса и переход на метку Reset
```

```
;Подпрограммы
Button:                 ;Метка начала подпрограммы
OUT  DDRB, temp         ;Вывод в порт значения регистра temp
INC   temp              ;Увеличение регистра на 1

CPI    temp, 0b00001000 ;Ограничиваем максимальное значение
BREQ Clear              ;Если мы до него дошли то выполняется
                        ; условный  переход на метку Clear

RJMP Reset              ;Переход  на метку Reset

Clear:                  ;Подпрограмма отчистки индикаторов
CLR  temp               ;Обнуляем переменную
RJMP Reset              ;Переход  на метку Reset

;Начало основной программы
Reset:
;Ожидание нажатия кнопки
SBIC PinB,3             ;Если разряд 3 порта B чист пропускаем
                        ;команду
RJMP Button             ;Переход  на метку Button

RJMP Reset              ;Переход  на метку Reset
```

Директивой .include мы определяем тип микроконтроллера в данном случае это AtTiny15, далее определяем регистру r18 имя temp, который будет использоваться для хранения различных временных величин. После отчистки регистра temp командой CLR выполняется вектор сброса и происходит переход к метке начала программы – Reset. В данной части программы происходит  ожидание нажатия кнопки SB1, команда SBIC относится к группе команд Test&Skip и пропускает следующую команду пока разряд PBB сброшен, в нашем случае это 3 разряд регистра порта B(PinB,3).