МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ (МАДИ)

О.И. МАКСИМЫЧЕВ, В.А. ВИНОГРАДОВ

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Методические указания
к лабораторным работам по дисциплинам
«Микропроцессорные системы»,
«Аппаратно-программные комплексы»,
«Проектирование микропроцессорных систем»

МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(МАДИ)

Кафедра «Автоматизированные системы управления»

Утверждаю
Зав. кафедрой профессор
_____ А.Б. Николаев
«___» _____ 20 ___ г.

О.И. МАКСИМЫЧЕВ, В.А. ВИНОГРАДОВ

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Методические указания
к лабораторным работам по дисциплинам
«Микропроцессорные системы»,
«Аппаратно-программные комплексы»,
«Проектирование микропроцессорных систем»

по направлениям подготовки
«Информатика и вычислительная техника»,
«Электрооборудование автомобилей и тракторов»

МОСКВА
МАДИ
2015

УДК 004.382.7:004.43
ББК 32.973.26-04-018
М171

Максимычев, О.И.

М171 Программирование микроконтроллеров: методические указания к лабораторным работам по дисциплинам «Микропроцессорные системы», «Аппаратно-программные комплексы», «Проектирование микропроцессорных систем» / О.И. Максимычев, В.А. Виноградов. – М.: МАДИ, 2015. – 88 с.

Рассмотрены вопросы, связанные с программированием микроконтроллеров (однокристальных микро-ЭВМ), применяемых в системах автоматического управления. Методические указания предназначены для студентов, обучающихся по специальностям «Информатика и вычислительная техника» 09.03.01 Бакалавр, «Электрооборудование автомобилей и тракторов» 13.04.02 Бакалавр, 13.03.02 Магистр. Может быть использовано инженерами и аспирантами, занимающимися вопросами программирования микроконтроллеров.

УДК 004.382.7:004.43
ББК 32.973.26-04-018

Учебное издание

МАКСИМЫЧЕВ Олег Игоревич
ВИНОГРАДОВ Вадим Алексеевич

ПРОГРАММИРОВАНИЕ
МИКРОКОНТРОЛЛЕРОВ

Методические указания
к лабораторным работам по дисциплинам
«Микропроцессорные системы»,
«Аппаратно-программные комплексы»,
«Проектирование микропроцессорных систем»

по направлениям подготовки
«Информатика и вычислительная техника»,
«Электрооборудование автомобилей и тракторов»

Редактор Т.А. Феоктистова

Подписано в печать 16.10.2015 г. Формат 60×84/16.
Усл. печ. л. 5,5. Тираж 100 экз. Заказ . Цена 185 руб.
МАДИ, 125319, Москва, Ленинградский пр-т, 64.

1. АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРОВ

Одним из самых распространённых и удачных по архитектуре, являются микроконтроллеры (МК) общего назначения фирмы Atmel Corp. Выпуск контроллеров AVR начался в 1996 г., в серийном производстве находятся три семейства – «Tiny», «Classic» и «Mega». Под общей маркой AVR Atmel объединены 8-разрядные высокопроизводительные RISC (Reduced Instruction Set Computer).

На настоящий момент по соотношению «цена – производительность – энергопотребление» AVR является одним из лучших на мировом рынке 8-разрядных МК. Микроконтроллеры AVR стали индустриальным стандартом среди МК общего назначения.

Области применения AVR многогранны. Для семейства «Tiny» это интеллектуальные автомобильные устройства самого разного назначения, игрушки, игровые приставки, материнские платы персональных компьютеров, защита доступа в мобильных телефонах, зарядные устройства, детекторы дыма и пламени, бытовая техника, пульты дистанционного управления. Для семейства «Classic» это модемы различных типов, изделия класса Smart Cards и устройства чтения для них, спутниковые навигационные системы для определения местоположения автомобилей на трассе, бортовые системы управления, сетевые карты, материнские платы компьютеров, сотовые телефоны, разнообразные промышленные системы контроля и управления. Для «Mega» AVR это аналоговые (NMT, ETACS, AMPS) и цифровые (GSM, CDMA) смартфоны, принтеры и ключевые контроллеры для них, МК аппаратов факсимильной связи и ксероксов, дисковых накопителей, Flash ROM, CD-ROM и т.д. Многие микропроцессорные системы автоматизированных систем управления в промышленности и на транспорте основаны на контроллерах AVR.

Для дальнейшего точного понимания технических характеристик микроконтроллеров необходимо перечислить основные принятые термины и условные обозначения:

- **Flash ROM** – объем энергонезависимой памяти программ в Кб;

- **EEPROM** – объем энергонезависимой памяти данных в Кб);
- **RAM** – объем статической памяти данных (в байтах);
- **External RAM** – возможность подключения к микроконтроллеру дополнительной микросхемы внешней статической памяти данных (килобайт);
- **ISP** – программирование МК в системе;
- **SPM** – функция само программирования Flash ROM-памяти МК в системе (без внешнего программатора);
- **JTAG** – встроенный JTAG – интерфейс;
- **I/O (pins)** – максимальное количество линий ввода/вывода;
- **Timer(s) 8/16 bit** – количество и разрядность таймеров;
- **USI** – универсальный коммуникационный интерфейс;
- **AC** – аналоговый компаратор;
- **ADC (channels)** – количество каналов АЦП;
- **Internal RC** – наличие внутренней RC-цепочки для автономной работы МК (без внешнего источника опорной частоты);
- **WDT** – сторожевой таймер;
- **BDC** – аппаратный программируемый блок защиты от сбоев при внезапном (в том числе и кратковременном) отключении питания МК;
- **UART** – асинхронный последовательный приемопередатчик;
- **SPI** – синхронный трехпроводной последовательный интерфейс;
- **I2C** – двухпроводной последовательный интерфейс;
- **RTC** – система реального времени (Real Time Control);
- **PWM (channels)** – количество независимых каналов ШИМ;
- **Command Set** – набор инструкций в системе команд.

Все AVR имеют Flash-память программ, которая может быть загружена как с помощью обычного программатора, так и с помощью **SPI**-интерфейса, в том числе непосредственно на целевой плате. Количество циклов перезаписи не менее 1000. Версии кристаллов семейства «Mega» имеют возможность само программирования, т.е. самостоятельно изменять заложенные в них программы и алгоритмы функционирования, и далее работать уже по новой программе или измененному алгоритму.

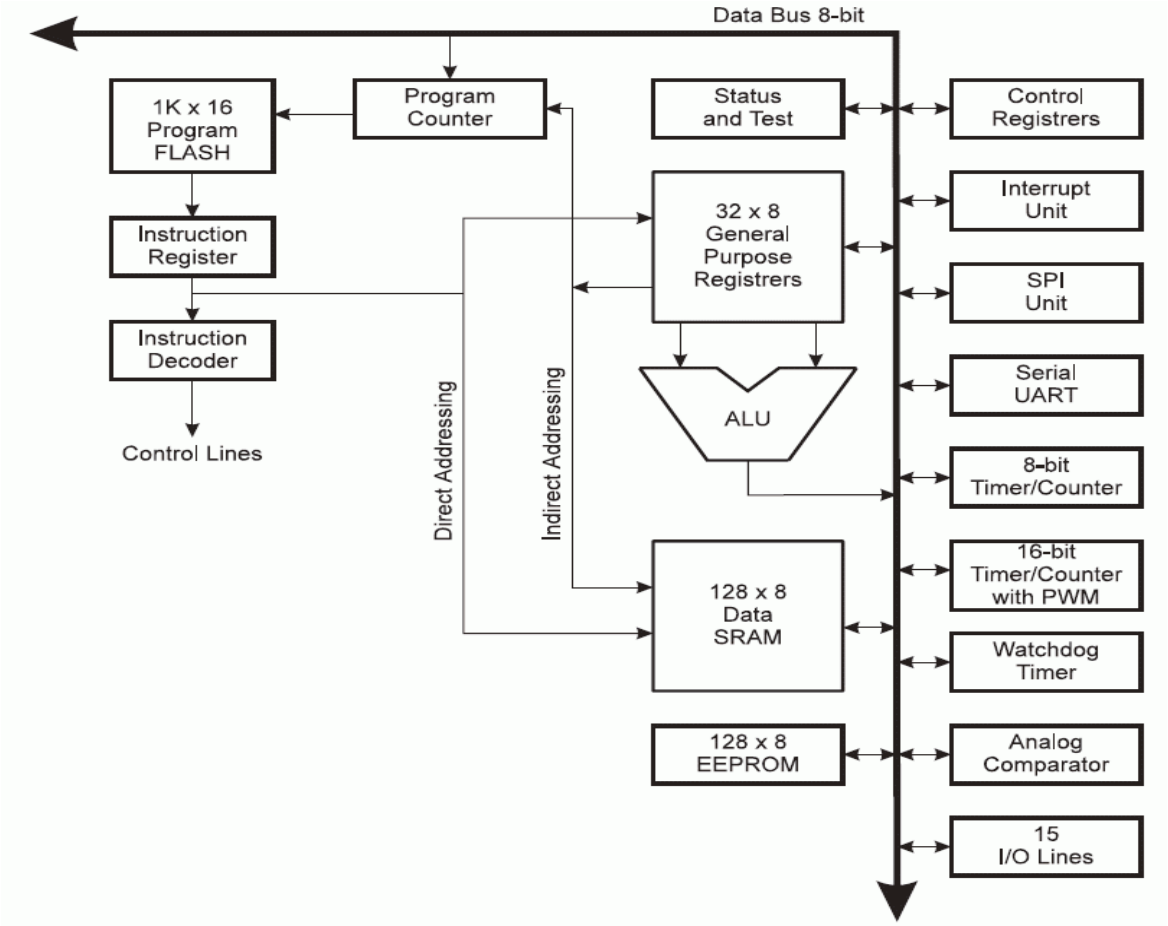

Рис. 1.1. Схема микроконтроллера AVR AT90S2313

Например, можно сохранить несколько версий программы для конкретного приложения во внешней энергонезависимой памяти (DataFlash, SEEPROM и т.п.), а затем, по мере необходимости или по реакции на какие-нибудь внешние или внутренние логические условия, перегружать рабочие программы в тот же самый МК без извлечения его из печатной платы. Для этого весь массив памяти программ подразделяется на две неравные по объему области: блок загрузчика (программа, управляющая перезаписью Flash-памяти программ) и блок для размещения рабочего программного кода, причем свободная память в области загрузчика может быть использована в качестве дополнительного пространства для рабочего кода.

Программа-загрузчик создается самим разработчиком и должна быть запрограммирована внешним программатором.

Все AVR имеют также блок энергонезависимой электрически стираемой памяти данных **EEPROM**. Этот тип памяти, доступный про-

грамме микроконтроллера непосредственно в ходе ее выполнения, удобен для хранения промежуточных данных, различных констант, таблиц перекодировок, калибровочных коэффициентов.

EEPROM может быть также загружена извне – как через SPI-интерфейс, так и с помощью обычного программатора. Два программируемых бита секретности позволяют защитить память программ и энергонезависимую память данных EEPROM от несанкционированного считывания. Чтение и запись ячеек EEPROM выполняются через регистры ввода/вывода EEAR (регистр адреса), EEDR (регистр данных) и EECR (регистр управления).

Внутренняя оперативная память SRAM имеется у всех AVR семейств «Classic», «Mega» и у одного из «Tiny» (ATtiny26/L); для некоторых возможна организация подключения внешней памяти данных объемом до 64 К слов.

Внутренний тактовый генератор AVR может запускаться от различных источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя RC-цепочка). Так как AVR – полностью статические МК, то значение рабочей частоты снизу не ограничено, вплоть до пошагового режима. Максимальная рабочая частота определяется конкретным типом МК.

Сторожевой (WATCHDOG) таймер предназначен для защиты МК от сбоев в процессе работы; в нем имеется собственный RC-генератор, работающий на частоте примерно 1 МГц (в основном его частота зависит от напряжения питания и температуры).

WATCHDOG-таймер снабжен своим собственным предварительным делителем входной частоты с программируемым коэффициентом деления, что позволяет подстраивать временной интервал переполнения таймера и сброса МК. WATCHDOG-таймер может быть отключен программным путем во время работы, как в активном, так и в любом из режимов пониженного энергопотребления. В последнем случае это приводит к значительному снижению потребляемого тока.

Порты ввода/вывода AVR имеют от 3 до 53 независимых линий «Вход/Выход», каждый разряд порта может быть запрограммирован на ввод или вывод информации.

Интересная архитектурная особенность портов ввода/вывода у AVR: для каждого физического вывода существует 3 бита контроля/управления, а не 2, как у распространенных 8-разрядных МК – Intel, Microchip, Motorola.

DDRx в данном случае – бит контроля направления передачи данных и привязки вывода к шине питания (VCC), PORTx – бит привязки вывода к VCC и бит выходных данных, PINx – бит для отображения логического уровня сигнала на физическом выводе микросхемы.

Естественный вопрос: для чего необходимо наличие именно трех битов? Дело в том, что использование только двух битов контроля/управления порождает ряд проблем при операциях типа «чтение – модификация – запись». Например, если имеют место две последовательные операции такого рода, то первый результат может быть безвозвратно потерян, если вывод порта работает на емкостную нагрузку и требуется некоторое время для стабилизации уровня сигнала на внешнем выводе микросхемы. Трехбитовая архитектура портов позволяет полностью контролировать процесс ввода/вывода.

Если необходимо получить реальное значение сигнала на физическом выводе МК, то содержимое бита читается по адресу PINx. Если требуется обновить выходы, то сначала следует считать PORTx-защелку, а затем модифицировать данные. Это избавляет от необходимости иметь копию содержимого порта в памяти для безопасности и повышает скорость работы МК при работе с внешними устройствами. Особую значимость данная архитектура приобретает для реализации систем, работающих в условиях электрических помех.

Аналоговый компаратор входит в состав большинства МК AVR. Типовое напряжение смещения равно 10 мВ, время задержки распространения составляет 500 нс и зависит от напряжения питания. Например, при напряжении 2,7 В оно равно 750 нс. Аналоговый компаратор имеет свой собственный вектор прерывания в общей системе прерываний МК, при этом тип перепада, вызывающий запрос на прерывание при срабатывании компаратора, может быть запрограммирован пользователем как фронт, срез или переключение. Логический выход компаратора может быть программным способом подключен к

входу одного из 16-разрядных таймеров/счетчиков, работающих в режиме захвата, что дает возможность измерять длительность аналоговых сигналов, а также достаточно просто реализовывать АЦП (см. ниже) двухтактного интегрирования.

Аналого-цифровой преобразователь (АЦП) построен по классической схеме последовательных приближений, с устройством АЦП в то время, когда центральный процессор находится в одном из режимов пониженного энергопотребления. При этом помехи, возникающие при работе процессорного ядра, не будут оказывать влияние на точность преобразования выборки/хранения (УВХ). Каждый из аналоговых входов АЦП может быть соединен с входом УВХ через аналоговый мультиплексор.

Разрядность АЦП составляет 10 бит при нормируемой погрешности ± 2 разряда. АЦП может работать в двух режимах – однократного преобразования по любому выбранному каналу и последовательного циклического опроса всех каналов. Время преобразования выбирается программным путем – с помощью установки коэффициента деления частоты специального предварительного делителя, входящего в состав блока АЦП, и равно 70 – 280 мкс для ATmega103 и 65 – 260 мкс для всех остальных МК.

Важной особенностью АЦП является наличие функции подавления шума.

МК AVR могут быть переведены в один из шести режимов пониженного энергопотребления программным путем [1].

2. АППАРАТНЫЕ ОСОБЕННОСТИ МК AVR

Аппаратные возможности микроконтроллеров могут отличаться по различным параметрам. Объемы массивов Flash-, EEPROM- и SRAM- памяти, набор периферийных узлов и построение схемы тактирования существенно различаются как между семействами, так и между МК внутри каждого семейства. Поэтому конкретные детали и полные описания, особенности построения и функционирования микроконтроллеров можно найти в оригинальной технической документации, предоставляемой производителем [1].

2.1. Основные технические характеристики

AVR представляет собой 8-разрядный МК типа RISC, имеющий «быстрый» Гарвардский процессор, память программ, память данных, порты ввода/вывода, схемы интерфейса. Структура МК приведена на рис. 2.1.

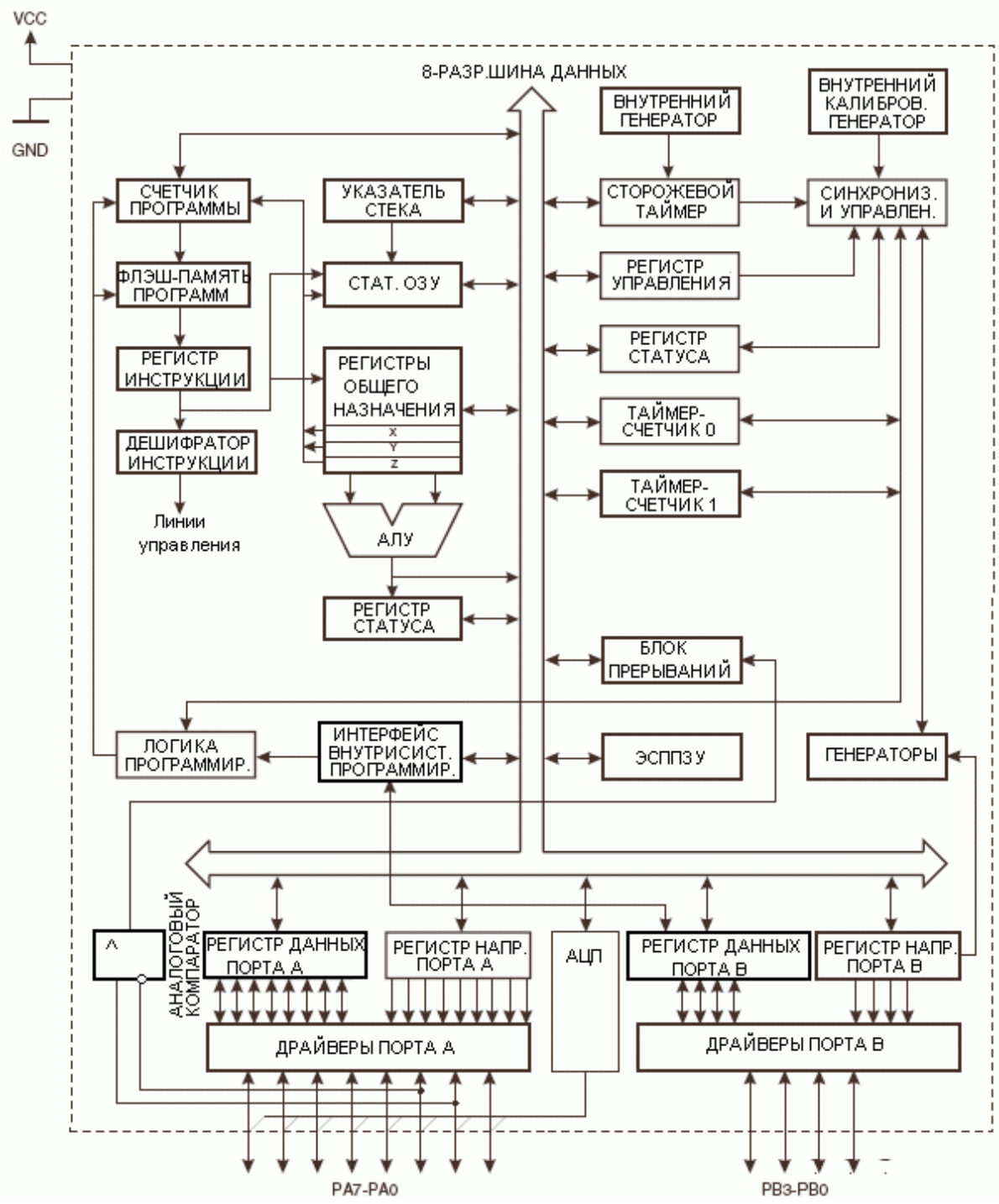

Рис. 2.1. Структура микроконтроллера AVR ATtiny24

Гарвардская архитектура AVR реализует полное логическое и физическое разделение не только адресных пространств, но и информационных шин для обращения к памяти программ и памяти данных, причем способы адресации и доступа к этим массивам памяти также различны. С памятью программ и с памятью данных центральный процессор работает одновременно; разрядность шины памяти программ расширена до 16 бит.

Основную идею всех RISC-архитектур представляет возможность увеличения быстродействия за счет сокращения количества операций обмена с памятью программ. Для этого каждую команду стремятся уместить в одну ячейку памяти программ, что при ограниченной разрядности ячейки неизбежно приводит к сокращению набора команд МК.

В соответствии с этим принципом в одной ячейке памяти программ размещаются практически все команды, исключая лишь те, у которых одним из операндов является 16-разрядный адрес. Но это сделано не за счет сокращения количества команд процессора, а путем расширения ячейки памяти программ до 16 разрядов, что и является причиной, увеличенной системы команд AVR по сравнению с другими RISC-МК.

Следующим шагом на пути повышения быстродействия является использование технологии конвейеризации, благодаря которой заметно сокращается цикл «выборка – исполнение» команды». Под конвейеризацией в данном случае подразумевается возможность выбора из памяти и дешифрации программного кода следующей команды во время исполнения текущей.

Для сравнения: у МК семейства MCS-51 выборка кода команды и ее исполнение осуществляются последовательно, что занимает один машинный цикл, который длится 12 периодов кварцевого резонатора. Для сравнения на рис. 2.2 приведены временные диаграммы выполнения типовой команды при различных платформах МК.

В случае использования конвейера приведенную длительность машинного цикла можно сократить, например, у МК PIC фирмы Microchip за счет использования конвейера удалось уменьшить длительность машинного цикла до 4 периодов кварцевого резонатора.

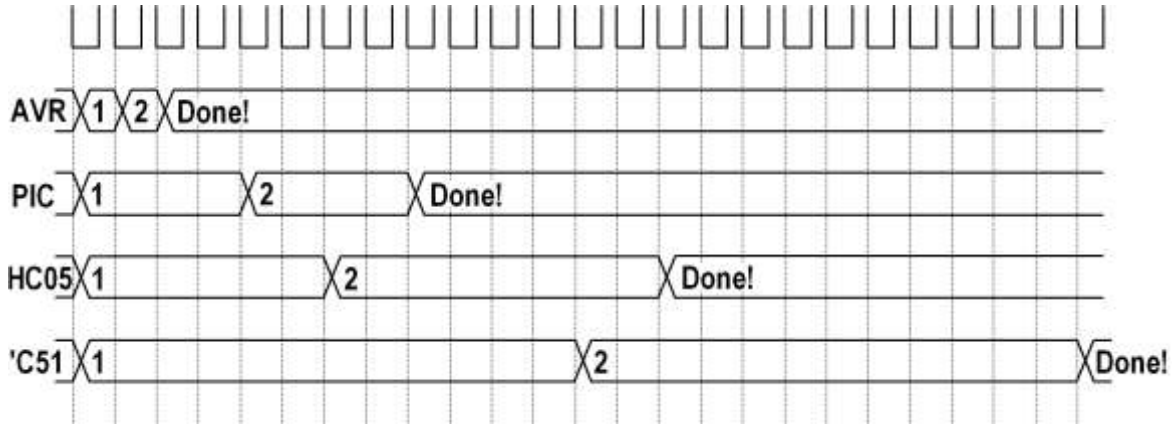

Рис. 2.2. Сравнительные характеристики МК по быстродействию

Длительность же машинного цикла AVR составляет один период кварцевого резонатора, таким образом, заданная производительность AVR обеспечивается при более низкой тактовой частоте. Именно эта особенность архитектуры и позволяет улучшить соотношение энергопотребление/производительность, которые для КМОП-микросхем определяются их рабочей частотой.

Следующая отличительная черта архитектуры AVR – регистровый файл быстрого доступа, структура которого показана на рис. 2.3. Использование трех 16-битных указателей (X, Y и Z Pointers) существенно повышает скорость пересылки данных при работе прикладной программы.

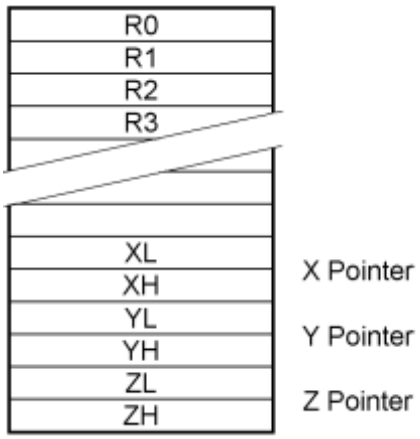

Рис. 2.3. Регистровый файл

Каждый из 32 регистров общего назначения длиной 1 байт непосредственно связан с арифметико-логическим устройством (АЛУ) процессора. Другими словами, в AVR используются 32 регистра-

аккумулятора, что позволяет в сочетании с конвейерной обработкой выполнять одну операцию в АЛУ за один машинный цикл, в течение которого из регистрового файла извлекаются два операнда, выполняется команда и результат записывается обратно в регистровый файл.

При косвенной адресации данных 6 из 32 регистров файла могут использоваться как три 16-разрядных указателя адреса, один из которых (Z Pointer) применяется также для доступа к данным, записанным в памяти программ МК.

Регистровый файл занимает младшие 32 байта в общем адресном пространстве SRAM AVR. Такое архитектурное решение позволяет получать доступ к быстрой «регистровой» оперативной памяти МК непосредственной адресацией в коде команды к любой ячейке или другими способами адресации ячеек SRAM. Это полезное свойство носит в документации Atmel название «быстрое контекстное переключение» и является еще одной отличительной особенностью архитектуры AVR, повышающей эффективность работы и производительность.

Особенно заметно данное преимущество при реализации процедур целочисленной 16-битной арифметики, многократных пересылок данных между различными ячейками памяти в ALU.

3. АССЕМБЛЕР ДЛЯ МИКРОКОНТРОЛЛЕРОВ AVR

Рассмотрим особенности программирования МК семейства AT90Sxxxx на ассемблере: в данном случае имеется в виду язык программирования (под «ассемблером» также подразумевают компилятор с языка ассемблер в машинный код).

3.1. Структура программы

Программу можно условно подразделить на следующие составные части:

1) подключение заголовочных файлов;

2) объявление констант;

3) объявление имён регистров;

4) сегменты программы;

5) комментарии.

Минимальная программа может содержать только сегмент кода, однако гораздо удобнее подключить и заголовочный файл для заданного контроллера. Программа физически хранится в текстовом файле, имеющем расширение *.asm.

Заголовочные файлы

Формат заголовочных файлов, как и файлов программы, текстовый, расширение *.inc. В заголовочных файлах содержатся константы, задающие символьные имена всем устройствам МК. Это позволяет, например, обращаться к порту B по имени PORTB и избежать использования его номера $18. Кроме того, заголовочный файл содержит директиву .device, которая указывает тип используемого МК.

Объявление констант

В дополнение к константам, объявленным в заголовочном файле, можно задавать собственные константы, используемые в конкретной задаче. С помощью констант удобно задавать маски для доступа к портам, граничные значения счётчиков и т.п. Для задания констант следует использовать директиву. equ с форматом записи:

.equ <*имя константы*> **=**<*значение константы*>

При необходимости значение константы может быть задано в шестнадцатеричной форме, для этого перед числом надо поставить знак «$». Кроме того, допускается запись чисел, аналогичная принятой в языке «Си»: **0xЧисло**.

Константа может быть представлена арифметическим выражением

.equ MASK = 1<<PIN6

Объявление имен регистров

Данный МК имеет 32 регистра, для доступа к которым следует использовать имена R0 – R31, на практике удобнее использовать для регистров символьные имена. Чтобы задать такое имя, следует воспользоваться директивой .def. Формат записи директивы:

.def <*имя регистра*> **=**<*регистр*>

В качестве имени регистра может использоваться произвольная строка символов, записанная по тем же правилам, что и имена переменных в языке «Си»: строка может состоять из букв, цифр и знака подчеркивания, первым знаком может быть либо буква, либо знак подчеркивания. В качестве регистра может использоваться любой регистр в диапазоне R0 – R31. Пример использования:

.def cnt = R0

Один и тот же регистр может иметь несколько имён.

Сегменты программы

Как любые программы, написанные для ЭВМ Фон Неймановской архитектуры, программы для данного МК (построенного в соответствии с Гарвардскими принципами) состоят из сегментов кода, данных и стека. В связи с тем, что МК содержит также энергонезависимую память EEPROM, в программу добавляется дополнительный сегмент, позволяющий обеспечить доступ и к ней.

Однако архитектура МК накладывает некоторые ограничения на доступ к этим сегментам. Например, сегмент кода, как правило, доступен только для чтения, причем командами, отличными от команд чтения данных из ОЗУ. Сегмент, описывающий содержимое EEPROM, вообще недоступен для чтения-записи стандартными командами (доступ к нему обеспечивается через порты ввода/вывода). Сегмент стека в программе не описывается, однако, для доступа к нему предусмотрены традиционные команды push/pop.

Физически сегменты располагаются следующим образом: сегмент кода во Flash-памяти, сегмент энергонезависимой памяти – в EEPROM-памяти, сегмент данных и сегмент стека – в RAM.

Начало сегмента кода отмечается директивой .CSEG, сегмент данных – .DSEG, сегмент EEPROM – .ESEG.

Сегмент кода

Сегмент кода состоит из набора инструкций, которые должны быть выполнены МК в процессе работы программы. Инструкции записываются следующим образом:

[*<метка:>*] *<инструкция>* *<операнды>*

Сегмент данных

Сегмент данных традиционно используется для хранения переменных. Для объявления переменных следует воспользоваться директивой **.BYTE**.

<имя переменной>: **.BYTE** 1

Последняя цифра указывает реальное количество переменных, под которые выделяется место. В случае если это значение отличается от 1, переменная является массивом.

Примечание: МК имеет общее адресное пространство для регистров, портов ввода/вывода и ОЗУ, которое располагается, начиная с адреса 0x60, поэтому при написании программ, которые обращаются к ОЗУ, следует учитывать это смещение. При использовании переменных, которые объявляются в программе (как показано выше), смещение вычисляется ассемблером, поэтому учитывать его не требуется.

Сегмент стека

Сегмент стека в программе не представлен, однако для корректной работы программы, в частности, для инструкций push/pop/call/ret, а также для работы прерываний необходимо настроить указатель стека. Обычно он устанавливается на конец области памяти RAM следующими командами:

```
ldi     temp, high(RAMEND)
out     SPH, temp
ldi     temp, low(RAMEND)
out     SPL, temp
```

В данном примере temp – это имя произвольного временного регистра, объявленного директивой **.def**.

Численное значение константы RAMEND может превышать 255, при этом для её хранения требуется более одного байта и используются директивы low и high, которые позволяют получить значения младшего и старшего байтов константы.

В случае использования МК AT90S2313 размер ОЗУ составляет всего лишь 128 байт, и для обращения к памяти нужен адрес в преде-

лах 0 – 127. Такой адрес можно хранить в одном восьмибитовом регистре, в данном случае он имеет имя SPL. Пример инициализации стека для данного МК:

```
ldi     temp, low(RAMEND)
out SPL,temp
```

Сегмент EEPROM

Используется для хранения данных, которые не теряются при выключении питания (примеры приводились выше).

Хранимые переменные могут иметь размер слова или байта. Для объявления следует воспользоваться директивой **db**. Формат записи директивы

<имя>:   **.db**   *<список значений>*

Пример использования директивы для описания таблицы перекодировки цифры в код семисегментного индикатора:

```
;0 1 2 3 4 5 6 7 8 9 E
digits: .db  0xEE, 0x60, 0x2F, 0x6D, 0xE1, 0xCD, 0xCF, 0xE8, 0xEF, 0xED, 0x8F
```

Комментарии

Комментарии могут располагаться в любом месте программы: первый символ – точка с запятой, последующие символы ассемблером не обрабатываются. Комментарий действует до конца строки, способ использования полностью совпадает с однострочными комментариями в языке Си, которые начинаются с символов: //.

Пример программы

Пример программы, демонстрирующий порядок описания в ней элементов, перечисленных выше.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример программы для МК AT90S2313.
; Подключение заголовочного файла:
.include "2313def.inc"
; Указание символьных имён регистров:
.def    temp        =R17
```

```
; Указание значений используемых констант:
.equ    btn_mask   =0x03
;.....................................................................;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода:
.CSEG
; Инструкции сегмента кода:
  ldi     temp, btn_mask
  out     PORTB, temp
;.....................................................................;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента данных:
.DSEG
var:     .BYTE 1 ; объявили переменную var
;.....................................................................;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Использование регистров

В соответствии с Гарвардской архитектурой МК содержит большое количество регистров («регистровый файл»), которые можно использовать для хранения переменных, что позволяет сократить количество обращений к ОЗУ и ускорить выполнение программы или вообще отказаться от использования ОЗУ (как в МК AT90S1200).

Все арифметико-логические инструкции предназначены для работы с регистрами. Для работы с памятью предусмотрены лишь команды пересылки: регистр <-> память, в связи с этим все промежуточные данные, используемые при вычислениях, следует хранить в регистрах.

3.2. Система команд

Система команд МК AVR весьма развита и насчитывает до 133 различных инструкций. В последних версиях семейства «Mega» реализована функция аппаратного умножения, что придает им еще большую универсальность.

Все команды (инструкции) можно разделить на следующие группы:

1) арифметико-логические;

2) битовые;

3) сравнения и условного перехода;

4) безусловного перехода;

5) вызова подпрограмм;

6) пересылки данных;

7) ввода/вывода;

8) специальные;

9) комбинированные.

Примечание: в оригинальной технической документации Atmel [1] перечислено пять групп; дополнительные добавлены авторами для упрощения изложения.

По разнообразию и количеству реализованных инструкций AVR больше похожи на CISC, чем на RISC-процессоры. Например, у PIC-контроллеров система команд насчитывает до 75 различных инструкций, а у MCS51 их число 111. В целом прогрессивная RISC-архитектура AVR в сочетании с наличием регистрового файла и расширенной системы команд позволяет в короткие сроки создавать работоспособные программы с кодом, более эффективным как по компактности реализации, так и по скорости выполнения.

При переходе к старшим моделям AVR существует совместимость по системе команд, однако необходимо помнить, что адреса векторов прерывания одних и тех же периферийных узлов у различных типов AVR различны, что требует внесения соответствующих изменений в программу при ее переносе на другой тип.

На рисунке 3.1 изображена программная модель, представляющая диаграмму программно доступных ресурсов AVR. Центральным блоком здесь является регистровый файл на 32 оперативных регистра (R0 – R31), непосредственно доступных ALU.

Все арифметические и логические операции, а также часть операций работы с битами (см. Приложение), выполняются в ALU только над содержимым оперативных регистров. Следует обратить внимание, что команды, имеющие в качестве второго операнда константу (SUBI, SBCI, ANDI, ORI, SBR, CBR), могут использовать в качестве первого операнда только регистры из второй половины регистрового

файла (R16 – R31). Команды 16-разрядного сложения с константой ADIW и вычитания константы SBIW в качестве первого операнда используют только регистры R24, R26, R28, R30.

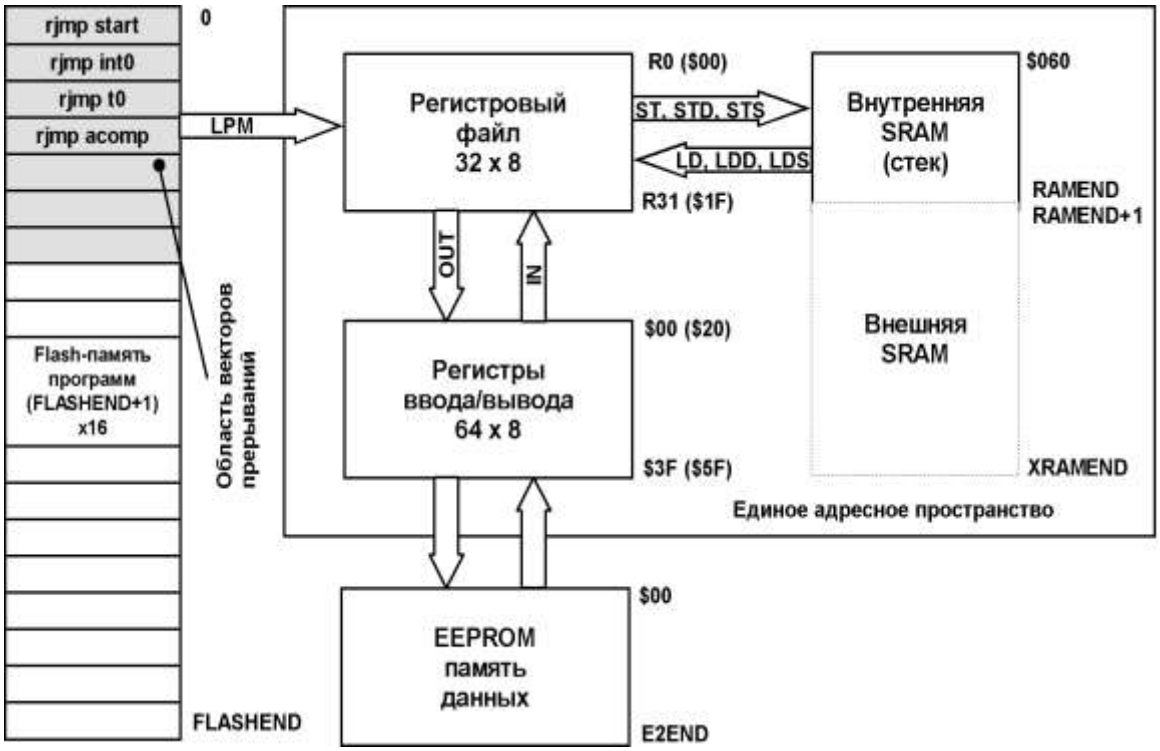

Рис. 3.1. Программная модель МК AVR

Во время выполнения арифметических и логических операций или операций работы с битами ALU формирует те или иные признаки результата операции (см. Приложение), то есть устанавливает или сбрасывает биты в регистре состояния SREG (Status Register), см. рис. 3.2.

Номер бита	7	6	5	4	3	2	1	0
Назначение	I	T	H	S	V	N	Z	C

Рис. 3.2. Регистр состояния SREG (Status Register)

- Бит C (carry) устанавливается, если во время выполнения операции был перенос из старшего разряда результата.
- Бит Z (zero) устанавливается, если результат операции равен 0.
- Бит N устанавливается, если MSB (Most Significant Bit – старший бит) результата равен 1 (правильно показывает знак результата, если не было переполнения разрядной сетки знакового числа).

- Бит V устанавливается, если во время выполнения операции произошло переполнение разрядной сетки знакового результата.
- Бит S = N + V (правильно показывает знак результата и при переполнении разрядной сетки знакового числа).
- Бит H устанавливается, если во время выполнения операции был перенос из 3-го разряда результата.

Признаки результата операции могут быть использованы для выполнения дальнейших арифметико-логических операций или команд условных переходов.

Для хранения оперативных данных можно, помимо регистрового файла, использовать внутренний и внешний блоки SRAM (если они имеются), рис. 3.1.

Работа с внешней SRAM может быть программно разрешена/запрещена установкой/сбросом бита SRE в регистре ввода/вывода USR.

Операции обмена с внутренней оперативной памятью выполняются за два машинных цикла. Доступ к внешней SRAM требует одного дополнительного цикла на каждый байт по сравнению с внутренней памятью. Кроме того, установкой бита SRW в регистре ввода/вывода MCUSR можно программно увеличить время обмена с внешней SRAM еще на один дополнительный машинный цикл ожидания.

Нельзя выполнять арифметико-логические операции и операции сдвига непосредственно над содержимым ячеек памяти, а также записывать константу или очищать содержимое ячейки памяти. Система команд AVR позволяет лишь выполнять операции обмена данными между ячейками SRAM и оперативными регистрами. Преимуществом системы можно считать разнообразные режимы адресации ячеек памяти. Как видно из приложения (см. группу команд передачи данных), кроме прямой адресации имеются следующие режимы: косвенный, косвенный с пост-инкрементом, косвенный с пре-декрементом и косвенный со смещением.

Поскольку внутренняя и внешняя SRAM входят в единое адресное пространство (вместе с оперативными регистрами и регистрами ввода/вывода), то для доступа к ячейкам внутренней и внешней памяти используются одни и те же команды.

В ячейках оперативной памяти организуется системный стек, который автоматически используется для хранения адресов возврата при выполнении подпрограмм, а также может использоваться программистом для временного хранения содержимого оперативных регистров (команды PUSH и POP). Микроконтроллеры, не имеющие SRAM, содержат трехуровневый аппаратный стек.

Следует иметь в виду, что если стек располагается во внешней SRAM, то вызовы подпрограмм и возвраты из них требуют двух дополнительных циклов, если бит SRW не установлен, и четырех, если установлен.

Размер стека, организуемого в оперативной памяти, ограничен лишь размерами этой памяти. Если МК содержит на кристалле 128 байт внутренней SRAM и не имеет возможности подключения внешней SRAM, то в качестве указателя вершины стека используется регистр ввода/вывода SPL. Если есть возможность подключения внешней памяти или внутренняя память имеет размеры 256 или больше байт, то указатель стека состоит из двух регистров ввода/вывода SPL и SPH.

При занесении числа в стек автоматически выполняются следующие действия.

1. Число записывается в ячейку памяти по адресу, хранящемуся в указателе стека. (SPH:SPL) <- число;

2. Содержимое указателя стека уменьшается на единицу:

SPH:SP = SPH:SPL – 1.

Обратные действия выполняются при извлечении числа из стека.

3. Содержимое указателя увеличивается на единицу:

SPH:SPL = SPH:SPL + 1.

4. Число извлекается из ячейки памяти с адресом, хранящимся в указателе стека: (SPH:SPL) -> число.

Таким образом, стек заполняется от старших адресов к младшим, поэтому, с учетом того, что начальное значение указателя стека после сброса равно нулю, программист AVR обязательно должен в инициализирующей части программы позаботиться об установке указателя стека, если он предполагает использовать хотя бы одну подпрограмму.

Кроме оперативной памяти, программно доступными ресурсами МК являются энергонезависимые, электрически программируемые FLASH- и EEPROM-блоки памяти, которые имеют отдельные адресные пространства.

Так как все команды AVR представляют собой 16-разрядные слова, FLASH-память организована как последовательность 16-разрядных ячеек и имеет емкость от 512 слов до 64 K слов – в зависимости от типа кристалла.

Во **FLASH-память**, кроме программы, могут быть записаны постоянные данные, которые не изменяются во время функционирования микропроцессорной системы. Это различные константы, таблицы знакогенераторов, таблицы линеаризации датчиков и т.п. Данные из FLASH-памяти могут быть считаны программным способом в регистровый файл с помощью команд LPM, ELPM (см. группу команд передачи данных).

Младшие адреса памяти программ имеют специальное назначение. С адреса $0000 программа начинает выполняться после сброса процессора. Начиная со следующего адреса $0001, ячейки памяти программ образуют область векторов прерывания. В этой области для каждого возможного источника прерывания отведен свой адрес, по которому (в случае использования данного прерывания) размещают команду относительного перехода RJMP на подпрограмму обработки прерывания.

3.2.1. Арифметико-логические инструкции

Данная группа инструкций предназначена для выполнения арифметико-логических операций. В качестве операндов могут использоваться один или два регистра, или пара регистр-значение. Большинство инструкций оперируют с данными размером в 1 байт – за исключением инструкций adiw и sbiw, которые позволяют обрабатывать сразу пару регистров, т.е. значение размером 2 байта.

3.2.2. Битовые инструкции

Битовые инструкции позволяют обрабатывать отдельно взятый бит или группу битов. Среди них можно выделить инструкции sbi и cbi,

которые работают с битами портов ввода/вывода, а также использующие в качестве операнда бит T (на самом деле этот бит хранится в регистре SREG, который так же доступен, как регистр ввода/вывода). Особенностью этого бита является возможность его использования в качестве «временной переменной». К битовым инструкциям можно также отнести часть инструкций условного перехода, с анализом битов как портов ввода/вывода, так и расположенных в регистрах.

3.2.3. Инструкции сравнения и условного перехода

Инструкции сравнения и условного перехода имеет смысл рассматривать в паре. Пара таких инструкций позволяет реализовать условия, обычно используемые в языках программирования высокого уровня. Приведём пример использования таких инструкций для реализации условия, проверяющего значение переменной cnt (должно храниться в регистре):

```
    ; if (cnt == 5)
    cpi     cnt, 5          ; сравниваем значения
    brne    skip            ; если не равны, то переход на ветвь else
            ; then ; если равны, то выполняем ветвь then
    in      temp, PORTB
    inc     temp            ; пример действий по ветви then
    out     PORTB, temp
skip:                       ; ветвь else
    ; здесь должны быть действия, выполняемые по ветви else.
```

Инструкции условного перехода могут использоваться и без инструкции сравнения. Это возможно в том случае, если условием является значение некоторого бита, например, бита порта ввода/вывода. Пример использования приводится в листингах (например, листинг программы проверки состояния кнопки).

Инструкции условного перехода существуют в двух вариантах: пропуска следующей инструкции и перехода на метку. Подробнее варианты описаны в таблице инструкций микроконтроллера. К сожалению, инструкции проверки бита порта ввода/вывода и пропуска сле-

дующей инструкции (SBIC, SBIS) позволяют задавать в качестве адреса порта только значения 0x00 – 0x1F, что не позволяет проверять биты порта SREG и, как следствие, получить ту же функциональность, что и набор инструкций условного перехода BRxx.

3.2.4. Инструкции безусловного перехода

Эти инструкции могут использоваться для организации бесконечного цикла, а также при обработке прерываний, так как таблица обработчиков прерываний может содержать только одну инструкцию на каждый вектор. Для замены операции условного перехода может быть использована комбинация инструкций безусловного перехода и пропуска следующей инструкции.

3.2.5. Инструкции вызова подпрограмм

Назначение таких инструкций полностью соответствует названию. При вызове подпрограмм для сохранения адреса возврата используется стек, который необходимо настроить до первого использования вызова. В функцию можно передавать параметры, используя традиционные пути: регистры, переменные в ОЗУ, стек. Кроме инструкции rcall, существует icall, которая берёт адрес вызываемой подпрограммы из регистра, а не в виде непосредственного значения. Любая подпрограмма должна заканчиваться инструкцией ret. Учитывая, что объём ОЗУ ограничен 128 байт, следует избегать частого использования рекурсивных подпрограмм.

3.2.6. Инструкции пересылки данных

Инструкции пересылки служат для передачи данных между регистрами, памятью, а также дают возможность загружать в регистры константы. Операнды, используемые этими инструкциями, можно классифицировать так:

1) регистр <–> регистр;
2) регистр <– константа;
3) регистр <–> память.

Обратите внимание, что не существует инструкций для пересылки значений между ячейками памяти. Чтобы выполнить такую операцию, следует разбить её на две отдельные инструкции: чтение значения из ячейки в регистр и запись значения из регистра в ячейку памяти. Подобная операция может использоваться, например, при сортировке массивов, расположенных в памяти.

Наиболее широко представлены инструкции работы с памятью. Полная классификация таких инструкций приводится в главе «**Система команд**». Учитывая, что данный МК (точнее, всё семейство AVR) разрабатывался для эффективного выполнения программ, написанных на языке Си, широко представлен набор команд, реализующий простейшие операции этого языка. Например, существует команда ld Rd, Z+, которая реализует выражение языка Си вида Rd = *p++;. Так как среди регистров выделены три пары: X, Y и Z, которые могут использоваться операциями пересылки данных, можно реализовать множество операций с массивами, которые будут выполняться весьма эффективно. Например, при сортировке обычно используются два указателя, которые адресуют пару сравниваемых (переставляемых) значений.

В данном МК для реализации алгоритмов сортировки можно использовать любую пару, например, Y и Z.

3.2.7. Инструкции ввода/вывода

Инструкции ввода/вывода используются для доступа к портам ввода/вывода. В принципе, адресные пространства этих портов и памяти различны, т.е. один и тот же адрес может соответствовать как порту, так и ячейке памяти, в зависимости от используемой инструкции. Однако адресное пространство памяти позволяет получить доступ к регистровому файлу, портам ввода/вывода и собственно к ячейкам памяти, расположенным в ОЗУ. Конечно, адрес одного и того же порта ввода/вывода будет разным, если к нему обращаться как к порту или как к памяти.

Для доступа к порту можно использовать инструкции ввода/вывода, которые делятся на 2 класса: битовые и инструкции доступа

ко всему порту сразу. Примеры битовых инструкций приведены выше, в главе «Битовые инструкции». Инструкций для доступа ко всему порту только две: in, out.

Особенность рассматриваемого МК: внутренние регистры (например, регистр флагов) доступны только как порты ввода/вывода. Для сравнения, процессоры семейства x86 обеспечивают доступ к регистру флагов регистровыми инструкциями.

Регистры ввода/вывода представляют собой набор регистров управления процессорного ядра и регистров управления и данных аппаратных узлов. Регистрами ввода/вывода являются регистры SREG, MCUSR и указатель стека SPH: SPL, а также регистры, управляющие системой прерывания МК, режимами подключения EEPROM-памяти, сторожевым таймером, портами ввода/вывода и другими периферийными узлами. Изучение данных регистров удобно выполнять одновременно с изучением конкретного периферийного узла.

Все регистры ввода/вывода могут считываться и записываться через оперативные регистры с помощью команд IN, OUT (см. группу команд передачи данных). Регистры ввода/вывода, имеющие адреса в диапазоне $00 – $1F (напомним, что знак $ указывает на шестнадцатеричную систему счисления), обладают возможностью побитовой адресации. Непосредственная установка и сброс отдельных разрядов этих регистров выполняются командами SBI и CBI (см. группу команд работы с битами). Для признаков результата операции, которые являются битами регистра ввода/вывода SREG, имеется целый набор команд установки и сброса. Команды условных переходов в качестве своих операндов могут иметь как биты – признаки результата операции, так и отдельные разряды побитно адресуемых регистров ввода/вывода.

Регистровый файл, блок регистров ввода/вывода и оперативная память, как показано на рис. 3.1, образуют единое адресное пространство, что дает возможность при программировании обращаться к 32 оперативным регистрам и регистрам ввода/вывода как к ячейкам памяти, используя команды доступа к SRAM (в том числе и с косвенной адресацией).

На рисунке 3.1 показано распределение адресов в едином адресном пространстве. Младшие 32 адреса ($0 – $1F) соответствуют оперативным регистрам. Следующие 64 адреса ($20 – $5F) зарезервированы для регистров ввода/вывода. Внутренняя SRAM у всех AVR начинается с адреса $60.

Таким образом, регистры ввода/вывода имеют двойную нумерацию. Если используются команды IN, OUT, SBI, CBI, SBIC, SBIS, то следует использовать нумерацию регистров ввода/вывода, начинающуюся с нуля (назовем ее основной). Если же к регистрам ввода/вывода доступ осуществляется как к ячейкам памяти, то необходимо использовать нумерацию единого адресного пространства оперативной памяти данных. Очевидно, что адрес в едином адресном пространстве памяти данных получается путем прибавления числа $20 к основному адресу регистра ввода/вывода.

Следует отметить, что регистры ввода/вывода используют не все отведенные для них 64 адреса. Неиспользуемые адреса зарезервированы для будущих применений; дополнительных ячеек памяти по этим адресам не существует.

Следует также иметь в виду, что у разных типов AVR одни и те же регистры ввода/вывода могут иметь различные адреса. Для того чтобы обеспечить переносимость программного обеспечения с одного типа кристалла на другой, следует использовать в программе стандартные символические имена регистров ввода/вывода, принятые в оригинальной технической документации Atmel [1].

Соответствие этих имен реальным адресам следует задавать, подключая в начале своей программы (с помощью директивы ассемблера, INCLUDE) файл определения адресов регистров ввода/вывода.

3.2.8. Специальные инструкции

Существуют только три специальные инструкции: nop, sleep и wdr. первая не выполняет никаких действий, она может использоваться для резервирования места в программе, организации задержек выполнения и т.д. Остальные служат для управления состоянием кон-

троллера. Для изучения этих инструкций рекомендуем обратиться к оригинальной технической документации Atmel [1].

3.2.9. Комбинированные инструкции

Существует только одна комбинированная инструкция: CPSE. Она совмещает проверку условия на равенство (инструкция сравнения) и условный переход (в варианте пропуска инструкции).

Комбинацию инструкций CPSE и RJMP (безусловный переход) удобно использовать для организации циклов:

```
    clr     i         ; счётчик цикла
    ldi     lim, 5    ; конечное значение счётчика
lp:
    ; здесь следует расположить тело цикла
    inc     i
    cpse    i, lim    ; проверяем, достиг ли счётчик значения 5
                      ; если достиг, пропускаем переход в начало цикла
    rjmp    lp        ; переход в начало цикла
```

Приведённый пример соответствует циклу:

for (i = 0; i < lim; i++) stmt.

3.2.10. Обработчики прерываний

Данный МК поддерживает только аппаратные прерывания. Для сравнения: процессоры Intel могут работать как с аппаратными, так и программными прерываниями. Особенности обработки аппаратных прерываний связаны с тем, что последние происходят асинхронно, в произвольный момент времени.

Учитывая, что данный МК не поддерживает автоматическое сохранение регистра флагов, значение, содержащееся в этом регистре, должен сохранять обработчик прерываний. К сожалению, процесс сохранения может быть осложнён тем, что регистр флагов является на деле портом ввода/вывода, и его сохранение требует инструкции ввода и инструкции сохранения (например, в стек).

При возникновении прерывания МК сохраняет в стеке адрес следующей инструкции (который содержится в счётчике команд) и пере-

ходит к выполнению инструкции, содержащейся в заданной ячейке таблицы прерываний (номер ячейки выбирается в соответствии с источником прерывания). Каждый обработчик прерываний является набором инструкций, который заканчивается инструкцией iret. При выполнении последней из стека извлекается адрес возврата и МК продолжает выполнение программы с этого адреса.

Существует несколько источников прерываний, каждому из которых должен быть поставлен в соответствие свой обработчик (если прерывание не используется, то обработчик можно не подключать).

Таблица обработчиков прерываний данного МК должна находиться в памяти программ, начиная с адреса 0. Каждый элемент таблицы должен содержать инструкцию перехода на обработчик прерывания, либо команду iret.

В случае, когда используется минимальное количество обработчиков прерываний, можно использовать директиву .org, которая позволяет расположить инструкции, начиная с некоторого адреса в памяти. Например, если необходим только обработчик прерывания таймера 0, то в самом начале программы можно написать:

```
.org   OVF0addr,
```

что позволит не писать инструкции reti для обработчиков, которые расположены до него. Константа OVF0addr объявлена в файле 2313def.inc и указывает численное значение адреса, с которого начинается выполнение обработчика прерывания OVF0, т.е. обработчика таймера 0.

Существует и крайний случай, когда таблица обработчиков прерываний не используется. В этом случае программа располагается, начиная с адреса 0, где обычно содержится инструкция перехода на обработчик прерывания RESET.

Если вместо инструкции перехода расположить код, то программа будет исполняться корректно, правда, следует запретить все прерывания (точнее, их не надо разрешать: по умолчанию все прерывания после сброса микроконтроллера запрещены). Примеры применения этого крайнего случая можно найти в листингах, большинство которых работает при запрещенных прерываниях.

Пример оформления обработчиков прерываний в программе на ассемблере (взято из фирменного описания микроконтроллера AT90S2313):

Адрес	Метка	Код	Комментарий
$000		rjmp RESET	; Reset handler
$001		rjmp EXT_INT0	; IRQ0 handler
$002		rjmp EXT_INT1	; IRQ1 handler
$003		rjmp TIM_CAPT1	; Timer1 capture handler
$004		rjmp TIM_COMP1	; Timer1 compare handler
$005		rjmp TIM_OVF1	; Timer1 overflow handler
$006		rjmp TIM_OVF0	; Timer0 overflow handler
$007		rjmp UART_RXC	; UART RX complete handler
$008		rjmp UART_DRE	; UDR empty handler
$009		rjmp UART_TXC	; UART TX complete handler
$00a		rjmp ANA_COMP	; Analog comparator handler
;			
$00b	MAIN:	ldi r16, low(RAMEND)	; Main program start
$00c		out SPL, r16	
$00d		…	

В данном примере RESET, EXT_INT0, EXT_INT1 и т.д. – это метки в программе, после которых должны располагаться обработчики прерываний.

4. ГРАФИЧЕСКАЯ СРЕДА РАЗРАБОТКИ «ALGORITHM BUILDER»

Графическая среда «Algorithm Builder» разработки программного обеспечения для МК с архитектурой AVR (графический ассемблер) предназначена для производства полного цикла разработки, начиная от ввода алгоритма, включая отладку, и заканчивая внутрисхемным программированием под ОС Windows. Разработка программ может производиться как на уровне ассемблера, так и на макроуровне, при котором возможна работа со знакопеременными величинами произвольной длины, что приближает возможности программирования к языку высокого уровня.

В отличие от классического ассемблера, программа вводится в виде алгоритма с древовидными ветвлениями и отображается на плоскости в двух измерениях. Сеть условных и безусловных переходов отображается графически в удобной векторной форме. Это к тому же освобождает программу от бесчисленных имен меток, которые в классическом ассемблере являются неизбежным балластом. Вся логическая структура программы становится наглядной.

Становится актуальным такое понятие, как дизайн алгоритма, предполагающее некоторый художественный вкус программиста. Основным предназначением технологий становится максимальное приведение интерфейса разработки к природе человеческого восприятия. Более удобный интерфейс раскрывает новые возможности для разработки.

Под конструированием алгоритма подразумевается размещение на рабочем поле программных блоков, наполнение их необходимым содержанием и установление между ними логических связей в виде условных и безусловных переходов.

По оценке пользователей, по сравнению с классическим ассемблером, время на разработку программного обеспечения сокращается в 3–5 раз, что делает его в этом смысле соизмеримым с языком С, при этом эффективность кода несравненно выше.

«Algorithm Builder» поддерживает автоматическую перекодировку строк ANSI-кодов Windows в коды русифицированного буквенно-цифрового индикатора, объединяет в себе графический редактор, компилятор алгоритма, симулятор МК, внутрисхемный программатор.

В дополнение к симулятору среда поддерживает отладку алгоритма на реальном МК.

4.1. Интерфейс программы «Algorithm Builder»

После запуска программы создайте первую страницу через пункт «Файл\Новый». Это будет первая, базовая страница проекта (может оказаться и единственной). Создаваемая страница с алгоритмом сохраняется на диске с расширением .alp для первой, базовой страницы

проекта и *.alg для остальных. При загрузке файла в редактор на диске создается его копия с расширением .~al, которой при необходимости можно воспользоваться для восстановления исходного файла.

С расширением .ini и именем основного файла проекта (*.alp), располагающегося на самой левой странице редактора, создается файл, который содержит в себе конфигурацию среды для данного проекта. В нем сохраняются имена файлов алгоритма, раскрытых на момент сохранения, а также размеры и положение основного окна и окон симулятора. После успешной компиляции на диске создается файл с содержимым памяти программы с именем проекта и файл с содержимым EEPROM, именем файла проекта с префиксом EE_. Файлы могут быть созданы в бинарном формате с расширением .bin, в текстовом формате General с расширением .rom или в формате «Intel HEX» с расширением .hex. Пункт «Файл», помимо стандартных команд для программ такого рода, содержит команду «Файл \ Печатать \ В метафайл», позволяющую сохранять набранные алгоритмы в метафайлы Windows.

Следующий пункт главного меню «Редактировать» содержит команды, позволяющие вырезáть, копировать, вставлять и удалять объект, находящийся в фокусе, или выделенную строку. Пункт «Откат» позволяет отменять изменения, сделанные в проекте вплоть до последнего сохранения.

Пункт главного меню «Отображение» содержит команды. «Переключить алгоритм / Таблица данных» позволяет переключаться между полем разработки алгоритма и специальной электронной таблицей, в которой можно задавать имена, назначать форматы переменных и распределять ресурсы проекта. Пункт «Шаблоны» отображает форму с различными закладками, на которых находятся шаблоны макроопераций, макроусловий, макрооперандов и др. Для вставки нужного элемента в алгоритм надо его выбрать и нажать кнопку «Insert».

«Поиск» содержит пункт «Найти», который позволяет находить нужный текст в объектах.

В пункте «Объект» сначала перечислены все объекты с которыми возможна работа в среде, а затем дополнительные функции для работы с этими объектами.

Пункт «Запуск» содержит подпункты:

Компилировать – компилирует исходный текст программы. Если до компиляции не выбран тип кристалла и частота, то предлагается это сделать. Компиляция всегда начинается с нулевой страницы (крайняя левая закладка). В случае обнаружения ошибки появляется соответствующее сообщение в статусной строке. При успешной компиляции производится автоматическое сохранение алгоритма и выдаётся сообщение о размере программы и об объеме свободной памяти.

Для запуска исполнения алгоритма в симуляторе необходимо либо выбрать пункт меню «Запуск с симулятором», либо нажать клавишу F9, либо нажать соответствующую кнопку на панели инструментов. При этом вначале произойдет компилирование алгоритма и вызов интерфейса симулятора.

Далее – пошаговое исполнение («Шаг со входом в п/п») производится нажатием либо клавиши F7, либо соответствующей кнопки на панели инструментов. Аналогично пошаговое исполнение без захода в подпрограммы и макрообразования («Шаг без входа в п/п») производится с помощью клавиши F8, а исполнение до выхода из текущей подпрограммы («Выход из п/п») – с помощью клавиши F6. Запуск на исполнение алгоритма до точки останова – по нажатию клавиши F9. Прервать решение можно клавишей F2. Установить или убрать точку останова на редактируемом объекте можно, нажав клавишу F5.

Синяя метка указывает на оператор, перед которым произошла остановка исполнения. Наблюдать и модифицировать текущее состояние различных компонентов микроконтроллера можно, раскрыв необходимые окна через пункт меню «Открыть». Модификация тех или иных значений производится двойным щелчком мыши. При этом, кроме изменения самого значения, возможно изменение его формы представления. По умолчанию все они отображаются в шестнадцатеричном коде, без знака. Кроме того, для окон рабочих регистров и объявленных переменных «SRAM watch» и «EEPROM Watch» величины отображаются в том формате, в котором они были объявлены, возможно их отображение в знакопеременном виде.

Для того чтобы добавить переменную в окне, необходимо щелкнуть правой кнопкой мыши, и в появившемся меню выбрать пункт «Add Watch…» (Добавить переменную). Для наблюдения за длительностью процессов предусмотрено окно «Process Time». Оно содержит четыре автономных счетчика циклов микроконтроллера, для каждого из них предусмотрена возможность остановки процесса по достижении введенного числа.

Для этого необходимо включить флажок «Enable». Если необходимо сбрасывать счетчик после останова, то необходимо включить флажок «Clear After Stop» (Сброс после останова). Если остановка процесса произошла по этому счетчику, то в окне появится красная надпись «STOP».

Пункт главного меню «Опции» содержит подпункт «Опции среды», при выборе которого вызывается одноименное окно, позволяющее на одной из вкладок выбрать язык и задать тип выходного файла, а на другой настроить цвета среды. Файлы могут быть созданы в бинарном формате с расширением .bin, в текстовом формате «General» с расширением .rom или в формате «Intel HEX» с расширением .hex. Подпункт «Опции проекта» позволяет задать тип кристалла и частоту работы.

4.2. Программирование в среде «Algorithm Builder»

4.2.1. Распределение ресурсов и назначение имён

В «Algorithm Builder» распределение ресурсов и назначение имен не обязательно. Программирование возможно с использованием стандартных имен рабочих регистров и регистров ввода/вывода, а обращение к ячейкам памяти SRAM и EEPROM производится непосредственно по физическому адресу, однако такое программирование удобно не всегда.

Распределение ресурсов и назначение имен производятся в специализированной электронной таблице. Переключение редактора между алгоритмом и таблицей производится посредством пункта ме-

ню «Просмотр \ Переключить алгоритм / Таблица данных» клавишей F12 или соответствующей кнопкой на панели инструментов.

В назначаемом имени можно использовать буквы A – Z, a – z, цифры 0 – 9, а также символ подчеркивания «_», причем первым символом не должна быть цифра. Компилятор нечувствителен к регистру буквы, поэтому такие имена, как, например, «GenerateNoise», «GENERATENOISE» и «generatenoise», будут считаться абсолютно одинаковыми. Для распределения ресурсов и назначения имен предусмотрены шесть секций:

1) назначения имен рабочих регистров;

2) назначения имен регистров ввода/вывода;

3) назначения имен битов регистров ввода/вывода;

4) назначения имен переменных оперативной памяти;

5) назначения имен переменных энергонезависимой памяти;

6) назначения имен констант.

Необходимость заполнения той или иной секции определяет сам программист.

Секции таблицы распределения ресурсов

Секция назначения имен рабочих регистров. Ключом начала секции является заголовок «Working registers», рис. 4.1. В этой секции предусмотрены следующие поля:

Name – назначаемое имя;

Index (необязательный параметр) – константа, определяющая индекс регистра. По умолчанию принимается индекс, следующий за предыдущим, а в начале секции – равный нулю;

Format (необязательный параметр) – формат объявляемого регистра. По умолчанию принимается однобайтный формат. При объявлении многобайтного регистра автоматически назначаются имена составляющих их однобайтных регистров. При объявлении двухбайтного регистра к именам добавляются буквы L и H, а для трех- и четырехбайтных – символы 0, 1 и т.д., в соответствии с порядковым номером. Например, при объявлении двухбайтного («Word») регистра с именем «Counter», автоматически объявляются составляющие его

два однобайтных регистра с именами «CounterL» и «CounterH». Многобайтные форматы рабочих регистров используются только в макрооператорах.

При объявлении допускается одному и тому же регистру давать несколько имен. По умолчанию в среде действуют стандартные имена однобайтных регистров “r0…r31” или “R0…R31”, а также “WL”=“r24”, “WH”=“r25”, “XL”=“r26”, “XH”=“r27”, “YL”=“r28”, “YH”=“r29”, “ZL”=”r30”, “ZH”=”r31” и двухбайтных (“Int16” или “Word”) “W”=(“r24:r25”), “X”=(“r26:r27”), “Y”=(“r28:r29”) и “Z”=(“r30:r31”).

Working registers:

Name	Index	Format	Commentary
rr0	0	Int16	двухбайтный рабочий регистр (r0,r1)
rr2		Int16	двухбайтный рабочий регистр (r2,r3)
rrrr0	0	Int32	четырехбайтный рабочий регистр (r0..r3)
rr16	16	Int16	двухбайтный рабочий регистр (r16,r17)
ArCount			однобайтный рабочий регистр (r18)

Рис. 4.1. Пример заполнения секции назначения имён рабочих регистров

Секция назначения имен регистров ввода/вывода. Ключом начала секции является заголовок: «I\O Registers», рис. 4.2. В этой секции предусмотрены следующие поля:

Name – назначаемое имя;

I/O Register – стандартное или ранее объявленное имя регистра ввода/вывода.

Одному и тому же регистру допускается давать несколько имен.

I/O registers:

Name	I/O register	Commentary
InputPort	PortA	объявление PortA как InputPort
OutputPort	p#h22	объявление порта #h22 как OutputPort

Рис. 4.2. Пример заполнения секции назначения имен регистров ввода/вывода

По умолчанию в среде действуют стандартные имена «p0 – p63», а также определенные в оригинальном описании микроконтроллера, такие, как «SPL», «SPH», «TCNT0», «EEDR» и т.д. Кроме того, в мак-

рооператорах возможно использование имен двойных регистров, например: «ADC», «TCNT1».

В большинстве случаев в заполнении этой секции необходимости нет, так как все используемые регистры уже имеют имя по умолчанию.

Секция назначения имен битов регистров ввода/вывода. Ключом начала секции является заголовок I\O Bit, рис. 4.3. В этой секции предусмотрены следующие поля:

Name – назначаемое имя;

I\O bit – уже существующее имя или запись, определяющая бит ввода/вывода.

По умолчанию в среде действуют типовые имена битов, определенные в оригинальном описании микроконтроллера, такие как «DDA7», «ACIC», «ADEN», «ICNC1» и т.д.

I/O bits:

Name	I/O bit	Commentary
Data	PortB.7	Объявление бита PortB.7 как Data
Clock	PortB.6	Объявление бита PortB.6 как Clock

Рис. 4.3. Пример заполнения секции назначения имен битов регистров ввода/вывода

Секция назначения имен переменных оперативной памяти. Ключом начала секции является заголовок SRAM, см. рис. 4.4.

SRAM:

Name	Address	Format	Count	Commentary
EditIndex				однобайтная переменная
ShowMode		Word		двухбайтная переменная
Reg			4	четыре однобайтных переменных
PhaseA	#h100	Int24	8	8 трехбайтных переменных по адресу #h100
PhaseB	@PhaseA		24	24 однобайтных переменнвх по адресу #h100

Рис. 4.4. Пример заполнения секции назначения имен переменных оперативной памяти (SRAM)

В этой секции предусмотрены следующие поля:

Name – назначаемое имя переменной (ячейки памяти);

Address (необязательный параметр) – константа, определяющая конкретное значение адреса. По умолчанию – следующий за предыдущим, либо, в начале #h60;

Format (необязательный параметр) – формат переменной. По умолчанию принимается однобайтный формат. Многобайтные форматы используются в макрооператорах;

Count (необязательный параметр) – число резервируемых ячеек. По умолчанию принимается равным 1.

Имя в алгоритме может быть использовано в операторах с непосредственной адресацией SRAM, например, Phase->r0.

Имя в комбинации с символом @ перед ним представляет собой константу, содержащую физический адрес SRAM, например:

[@PrintPage+4]->r0

Секция назначения имен переменных энергонезависимой памяти. Ключом начала секции является заголовок EEPROM, рис. 4.5.

EEPROM:

Name	Address	Format	Count	Value	Commentary
EE_A					однобайтная ячейка
StartValue		Word	3	259,3331,0	четыре двухбайтных ячейки с загрузкой
					начальных значений
InitScale		DWord		#h27773F	одна четырехбайтная ячейка с загрузкой
					начального значения

Рис. 4.5. Пример заполнения секции назначения имен переменных энергонезависимой памяти (EEPROM)

Здесь предусмотрены следующие поля:

Name – назначаемое имя переменной;

Address (необязательный параметр) – константа, определяющая конкретное значение адреса. По умолчанию – следующий после предыдущего, либо 0 в начале секции;

Format (необязательный параметр) – формат ячейки. По умолчанию принимается однобайтный формат (при необходимости можно задать многобайтный);

Count (необязательный параметр) – число резервируемых ячеек. По умолчанию принимается равным 1;

Value (необязательный параметр) – начальные значения, представленные в виде константы или массива.

В алгоритме объявленное имя может быть использовано в макрооператорах, реализующих операции с EEPROM, например,

EE_Scale->X, 1875->EE_Scale. А имя в сочетании с символом @ перед ним может быть использовано в качестве константы, содержащей физический адрес EEPROM, например @DecLine->Y.

Секция назначения имен констант. Ключом начала секции является заголовок Constants, рис. 4.6.

Constants:

Name	Value	Commentary
Weight_Index	7	объявление Wait_Index константой 7
Weight_Min	10	
Weight_Typ	75	
Weight_Max	Wait_Min+100	

Рис. 4.6. Пример заполнения секции назначения имен констант

В этой секции предусмотрены следующие поля:

Name – назначаемое имя;

Value – алгебраическое выражение, определяющее величину.

Если в операторе алгоритма необходимо выбрать конкретный байт из константы, то к ее имени следует дописать «.n», где n – номер байта, начиная с 0. Например, если константа #h1234 объявлена как Len, то запись Len.1 будет константой #h12.

4.2.2. Константы и массивы констант

Непосредственное представление констант

Константа в алгоритме может быть представлена либо в обычном десятичном, шестнадцатеричном, восьмеричном, двоичном или символьном виде.

В десятичном виде константа записывается как обычно, с использованием символов 0 – 9, например: 35 или 24000.

Запись константы в других системах счисления начинается с символов: #h, #o, #b (соответственно: шестнадцатеричной, восьмеричной, двоичной). Например: #he5, #H23E7, #o107, #O2071, #b01101101, #B10011110.

В представлении константы допускается указание знака в виде символов + или –. Например: –54, +2, –#h5E3F.

При символьном представлении констант пользуются кавычками. В этом случае значением констант будут ANSI-коды включенных в них символов. Например, такие представления констант, как 0 и #h30, будут полностью эквивалентны.

При необходимости коды символов могут быть автоматически модифицированы посредством декодирующего файла *.dcd, имя которого (с расширением или без) указывается в круглых скобках после завершающей кавычки.

Например: "ПРОЦЕСС"(LCD_CYR), где LCD_CYR.dcd – файл, который включён в пакет среды «Algorithm Builder» и обеспечивает адаптацию кодов к кодам буквенно-цифрового жидкокристаллического дисплея с кириллицей.

Представление массива констант

Массив констант может потребоваться при загрузке начальных значений в EEPROM («энергонезависимая перезаписываемая память») или при размещении его в памяти программы. Массив может быть представлен в непосредственном виде или в виде файла.

В непосредственном виде представление массива констант должно начинаться с записи DB:, затем следует необязательное указание на формат данных. По умолчанию принимается однобайтный формат. Затем перечисляются константы, представленные в любом виде, которые должны разделяться пробелами или запятыми. Например:

DB:Word #h1234 #h4E35 –12000

или:

DB: #h2E,#h1F,#h12,#h45,#hF5

Константы могут быть представлены в виде строки в кавычках. При этом массив должен быть однобайтным. В этом случае в качестве констант будут коды каждого из символов.

Пример: DB:”Hello!”.

При необходимости коды введенных символов могут быть автоматически модифицированы посредством декодирующего файла.

Представление массива констант в виде файла должно начинаться с записи Load: с последующей записью имени файла (без ка-

вычек). Например, Load: table.dat. Эта директива обеспечит побайтное копирование файла в память программы.

4.2.3. Операторы среды «Algorithm Builder»

Формат записи операторов существенно отличается от классического ассемблера, он построен по визуально функциональному принципу и содержит образ выполняемого действия. Несколько характерных примеров приведено в табл. 4.1.

Таблица 4.1

Примеры команд

Классический ассемблер	**Algorithm Builder**	Описание
MOV R0,R1	R1->R0	Поместить содержимое регистра R0 в R1
LDI R16, 24	24->R16	Поместить константу 24 в регистр R16
ADD R0, R5	R0+R1	Сложить содержимое регистров R0 и R1
SBI PortB,3	1 ->PortB.3	Установить бит в 3-м разряде порта B
INC R1	R1++	Увеличить значение R1 на 1

Операторы алгоритма делятся на элементарные операторы и макрооператоры (рис. 4.7).

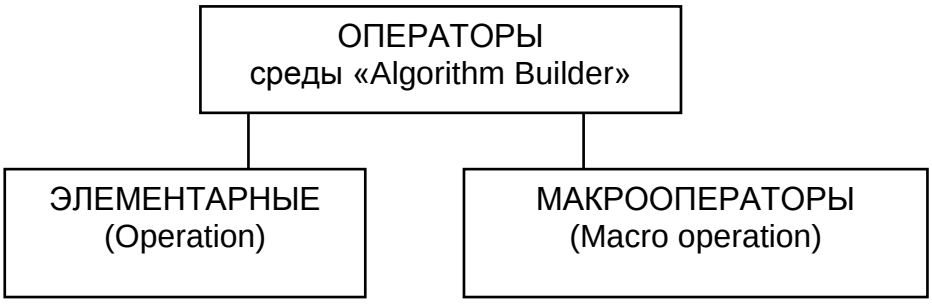

Рис. 4.7. Состав операторов среды «Algorithm Builder»

Элементарные операторы реализуют одну элементарную инструкцию микроконтроллера. Программирование с использованием таких операторов обеспечивает разработку программы на уровне ассемблера. В алгоритме эти операторы отображаются обычным (нежирным) шрифтом.

Макрооператоры (см. табл. 4.2) предназначены для реализации в более удобной форме ряда операций, которые не включены в систему команд микроконтроллера (например, 124->R0), а также многобайтных операций (например, #H25F->SP). При компиляции макрокоманды преобразуются в набор необходимых элементарных инструкций микроконтроллера. В алгоритме эти операторы отображаются жирным шрифтом. Отдельно следует выделить **условные операторы**, рис. 4.8, которые включают в себя элементарные операторы условного перехода, пропуска следующего оператора и макроусловия.

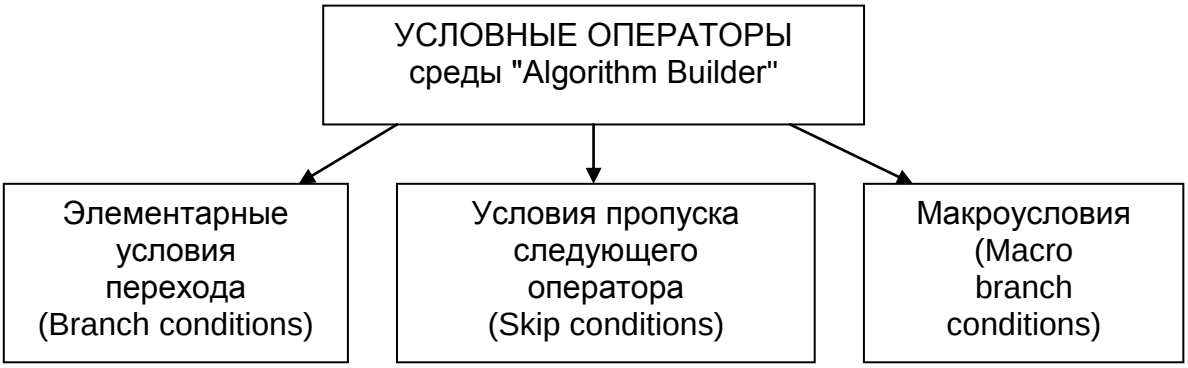

Рис. 4.8. Условные операторы

Таблица 4.2

Условные операторы

Шаблон	Комментарий
*** -> ***	Копирование
*** + ***	Арифметическое сложение
*** - ***	Арифметическое вычитание
*** & ***	Побитная операция “И”
*** ! ***	Побитная операция “ИЛИ”
*** ^ ***	Побитная операция “Исключающее ИЛИ”
*****	Сброс (запись нуля)
*** ++**	Инкремент
*** --**	Декремент
- * -	Побитная инверсия
*** >>**	Логический сдвиг вправо
> * >>	Логический сдвиг вправо с переносом
± * >>	Арифметический сдвиг вправо
<< *	Логический сдвиг влево
<< * <	Логический сдвиг влево с переносом

Таблица 4.3

Шаблоны макроусловий

Шаблон	Комментарий
*** = ***	Переход, если равно
*** -= ***	Переход, если не равно
*** < ***	Переход, если меньше
*** >= ***	Переход, если больше или равно
*** ±< ***	Переход, если меньше с учетом знака
*** ±>= ***	Переход, если больше или равно с учетом знака
R--	Декремент однобайтного регистра и переход, если не равно нулю

Вместо звездочки (*****) в шаблоны может быть подставлен любой из операндов.

Таблица 4.4

Операнды в макрооператорах

Операнд	Комментарий
1	**2**
R	Рабочий регистр, стандартный или объявленный в секции **Working registers:** (**...**)
#	Константа (**...**)
SRAM Var	Переменная, объявленная в секции **SRAM:** (**...**)
EEPROM Var	Переменная, объявленная в секции **EEPROM:** (**...**)
[#]	Ячейка SRAM, адресуемая непосредственно
[X]	Ячейка SRAM, адресуемая косвенно по X
[--X]	Ячейка SRAM, адресуемая косвенно по X с предекрементом
[X++]	Ячейка SRAM, адресуемая косвенно по X с постинкрементом
[Y]	Ячейка SRAM, адресуемая косвенно по Y
[--Y]	Ячейка SRAM, адресуемая косвенно по Y с предекрементом
[Y++]	Ячейка SRAM, адресуемая косвенно по Y с постинкрементом
[Y+#]	Ячейка SRAM, адресуемая косвенно по Y со смещением адреса на # байт
Z]	Ячейка SRAM, адресуемая косвенно по Z
[--Z]	Ячейка SRAM, адресуемая косвенно по Z с предекрементом
[Z++]	Ячейка SRAM, адресуемая косвенно по Z с постинкрементом

Продолжение табл. 4.4

Операнд	Комментарий
1	**2**
[Z+#]	Ячейка SRAM, адресуемая косвенно по Z со смещением адреса на # байт
P	Регистр ввода/вывода (**...**)
EE[#]	Ячейка EEPROM адресуемая непосредственно
EE[X]	Ячейка EEPROM, адресуемая косвенно по X
EE[X++]	Ячейка EEPROM, адресуемая косвенно по X с постинкрементом
EE[Y]	Ячейка EEPROM, адресуемая косвенно по Y
EE[Y++]	Ячейка EEPROM, адресуемая косвенно по Y с постинкрементом
EE[Z]	Ячейка EEPROM, адресуемая косвенно по Z
EE[Z++]	Ячейка EEPROM, адресуемая косвенно по Z с постинкрементом

Операнды, отмеченные как (**...**), могут быть объявленными многобайтными, допускается подстановка операндов разного формата. При этом если операнд, принимающий результат операции, короче, то размерность операции будет ограничена наименьшим, в противоположном случае недостающие байты будут заполнены нулями. Следует иметь в виду, что для корректности операции с величинами, у которых учитывается знак, у обоих операндов должен быть одинаковый формат, в противном случае, отрицательное число может быть искажено. Операнды в макроусловиях должны иметь одинаковый формат.

Операнды с косвенной адресацией являются однобайтными. Для многобайтных операций с косвенной адресацией необходимо приведение их формата. Для этого к записи необходимо добавить двоеточие и объявление формата, например:

#hAB3E->[Y]:Word

Такой макрооператор будет преобразован в следующую последовательность операторов:

```
#h3E->r16
r16->[Y]
#hAB->r16
r16->[Y+1]
```

По умолчанию в макроусловиях используются элементарные инструкции условных переходов МК, которые могут обеспечить короткий переход в пределах ±63 от текущего содержимого программного счетчика. Макроусловие может быть помечено как условие с длинным переходом. Для этого необходимо выбрать пункт меню «Объект\Long branch». Такое макроусловие будет отображаться с небольшим треугольником возле правого края контура, рис. 4.9. В графической среде предусмотрено наличие шаблонов операторов, предназначенных для выбранного типа микроконтроллера.

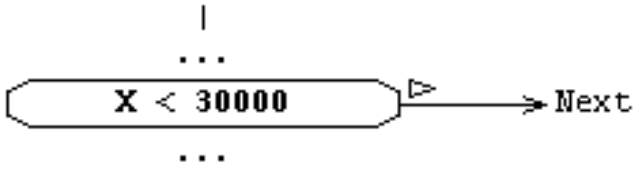

Рис. 4.9. Макроусловие

4.2.4. Элементы конструкции алгоритма

Под конструированием алгоритма подразумевается размещение на рабочем поле программных блоков, наполнение их необходимым содержанием и установка между ними логических связей в виде векторов условных и безусловных переходов.

Для этого в «Algorithm Builder» используются семь базовых объектов: Текст (Text), Вершина (Vertex), Поле (Field), Метка (Label), Условие (Condition), Вектор безусловного перехода (JMP Vector), Настройщик (Setter).

Вызов объекта осуществляется из меню «Объект» или с помощью кнопок на панели инструментов.

Объект “Text” (строка локального текстового редактора) представляет собой текстовую строку, начинающуюся от левого края поля алгоритма. Совокупность из нескольких таких строк образует локальный текстовый редактор, окаймлённый пунктирными линиями, правила работы в нём аналогичны правилам для прочих текстовых редакторов. Строки предназначены для записи в них директив компилятора, а также для комментариев. Комментарии должны начинаться с двух косых: //.

Объект “Label” представляет собой вертикальный штрих, расположенный на оси блока операторов, и необязательное имя, располагающееся слева или справа от штриха. Метка предназначена для обозначения мест в алгоритме, куда возможно осуществление условных и безусловных переходов.

При необходимости метке может быть назначен конкретный адрес программы, для этого перед именем (если оно есть) необходимо записать константу или алгебраическое выражение, которое определяет этот адрес.

Объект “Vertex” (вершина блока) по своему отображению и назначению полностью идентичен метке, но, в отличие от нее, задает расположение блока на рабочей плоскости и всегда является его началом.

Объект “Field” представляет собой строку, отцентрированную в блоке. Предназначен для записи большинства операторов.

Объект “Condition” (условный переход) предназначен для реализации условных переходов. Графически представляет собой овальный контур, внутри которого вписывается условие перехода и возможный вектор перехода в виде ломаной линии, исходящей от одного из краёв контура. Линия на конце имеет стрелку, возле которой возможно необязательное имя вектора. Для правильной адресации вектора его конец должен заканчиваться на метке, или вершине, или на отрезке другого вектора, либо иметь имя адресуемой метки. Если вписанное условие выполняется, то действие интерпретируется как ветвление. Частным случаем использования является условный пропуск следующего оператора (рис. 4.10).

При этом вектор либо отсутствует, либо имеет зарезервированное слово "Skip". В нижеследующих примерах выполнение операции "R3->R2" будет пропущено, если выполнится условие "R1=R2".

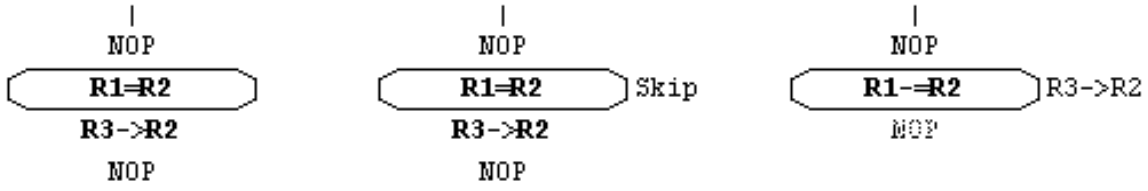

Рис. 4.10. Условный пропуск оператора

Показанные на рис. 4.10 три примера полностью идентичны.

Объект “JMP Vector” (безусловный переход) предназначен для реализации коротких безусловных переходов (в базовом ассемблере – оператор RJMP). Представляет собой ломаную линию, исходящую из середины блока со стрелкой на конце и аналогичную вектору объекта **“Condition”**.

Следует обратить внимание на ряд особенностей при реализации безусловных переходов и вызовов подпрограмм:

- короткий безусловный переход (аналог “RJMP”) реализуется исключительно посредством объекта “JMP Vector”;
- дальний безусловный переход (аналог “JMP k”) реализуется записью с шаблоном “JMP k” в объекте “Field”, где “k” может быть константой с адресом перехода или именем метки (вершины);
- короткий вызов подпрограммы (аналог “RCALL”) реализуется простой записью имени метки (вершины) или константы с адресом подпрограммы в объекте “Field”;
- дальний вызов подпрограммы (аналог “CALL k”) реализуется записью с шаблоном “CALL k” в объекте “Field”, где “k” может быть константой с адресом подпрограммы или именем метки (вершины);
- косвенный переход (аналог “IJMP”) и вызов подпрограммы (аналог “ICALL”) реализуются соответственно записями “JMP” и “CALL” в объекте “Field” без параметров.

Объект “Setter” представляет собой серый прямоугольник, внутрь которого вписано имя настраиваемого периферийного компонента микроконтроллера, такого как таймер, АЦП, регистр маски прерываний и пр. Предназначен для формирования последовательности операций МК, которые обеспечивают загрузку необходимых констант в управляющие регистры ввода/вывода в соответствии с выбранными свойствами.

Объект “Setter” является макрооператором. После компиляции он преобразуется в последовательность команд МК, которые обеспечивают загрузку необходимых констант в соответствующие управляю-

щие регистры. При этом следует иметь в виду, что в этих операциях будет использован регистр-посредник R16.

Для таких компонент, как ADC и UART, настройщик может воздействовать на несколько управляющих регистров. В этом случае при необходимости воздействие на каждый конкретный регистр можно заблокировать.

Недостатком среды «Algorithm Builder» можно считать невозможность конвертирования графического представления мнемокода программы в классический ассемблер. Переход к использованию такой среды труден психологически, однако освоение этого инструмента и последующая работа с ним заметно проще классического ассемблера.

4.2.5. Метки обслуживания прерываний

Прерывание прекращает нормальный ход программы для выполнения приоритетной задачи, определяемой внутренним или внешним событием микроконтроллера. При возникновении прерывания микроконтроллер сохраняет в стеке содержимое счетчика команд PC и загружает в него адрес соответствующего вектора прерывания. По этому адресу должна находиться команда относительного перехода к подпрограмме обработки прерывания.

Кроме того, последней командой подпрограммы обработки прерывания должна быть команда RETI, которая обеспечивает возврат в основную программу и восстановление предварительно сохраненного счетчика команд.

Поскольку источниками прерываний являются различные периферийные устройства микроконтроллеров, количество прерываний (2…16) зависит от конкретной модели.

Таблица векторов прерываний

Микроконтроллеры AVR семейства Classic имеют многоуровневую систему приоритетных прерываний. Младшие адреса памяти программ, начиная с адреса $001, отведены под таблицу векторов прерывания. Каждому прерыванию соответствует свой адрес в этой таблице, и именно этот адрес загружается в счетчик команд при возник-

новении прерывания. Положение вектора в таблице определяет также и приоритет соответствующего прерывания: чем меньше адрес, тем выше приоритет прерывания. Размер таблицы зависит от модели микроконтроллера и составляет от 2 (адреса $001, $002) до 16 (адреса $001…$010) векторов. Распределение таблицы векторов прерываний для всех микроконтроллеров семейства приведено в табл. 4.5.

Таблица 4.5

Таблица прерываний

Источник	Описание	AT90S1200		AT90S2313		AT90S/LS2323 AT90S/LS2343		AT90S/LS2333 AT90S/LS4433		AT90S/LS4434 AT90S/LS8535		AT90S4414 AT90S8515		AT90C8534	
		№	Адр.	№	Адр.	№	Адр.	№	Адр.	№	Адр.	№	Адр.	№	Адр.
INT0	Внешнее прерывание 0	1	$001	1	$001	1	$001	1	$001	1	$001	1	$001	1	$001
INT1	Внешнее прерывание 1			2	$002			2	$002	2	$002	2	$002	2	$002
TIMER2 COMP	Совпадение таймера/счетчика T2									3	$003				
TIMER2 OVF	Переполнение таймера/счетчика T2									4	$004				
TIMER1 CAPT	Захват таймера/счетчика T1			3	$003			3	$003	5	$005	3	$003		
TIMER1 COMP	Совпадение таймера/счетчика T1			4	$004			4	$004						
TIMER1 COMPA	Совпадение «A» таймера/счетчика T1									6	$006	4	$004		
TIMER1 COMPB	Совпадение «B» таймера/счетчика T1									7	$007	5	$005		
TIMER1 OVF	Переполнение таймера/счетчика T1			5	$005			5	$005	8	$008	6	$006	3	$003
TIMER0 OVF	Переполнение таймера/счетчика T0	2	$002	6	$006	2	$002	6	$006	9	$009	7	$007	4	$004
SPI, STC	Передача по SPI завершена							7	$007	10	$00A	8	$008		

Для удобства программирования «Algorithm Builder» поддерживает специальный вид меток – метки обслуживания прерываний. Для обслуживания прерывания обычным путем необходимо размещение по адресу вектора прерывания кода безусловного перехода на соответствующую подпрограмму. При использовании специального вида меток компилятор проделывает все это автоматически. Для этого вам

необходимо дать метке (вершине) стандартное имя прерывания, и пометить ее как макро-образование, нажав клавишу F2, при этом имя будет отображаться жирным шрифтом. Тот же результат можно проще получить, выбрав пункт меню “Elements\Interrupt vectors\…”. Встретив хотя бы одну такую метку в алгоритме, компилятор заполнит свободное пространство векторов прерывания кодом возврата из подпрограммы обслуживания прерывания (“RETI”), а по соответствующему прерыванию адресу поместит код безусловного перехода на данную метку. Внимание: если вы используете метки обслуживания, то начальные адреса программы будут автоматически заняты безусловными переходами на подпрограммы обслуживания прерываний. Поэтому, для того чтобы программа могла нормально стартовать, началом алгоритма обязательно должна быть макро-метка “Reset”, как показано на рис. 4.11. Это обеспечит загрузку в нулевой адрес безусловного перехода на начало алгоритма.

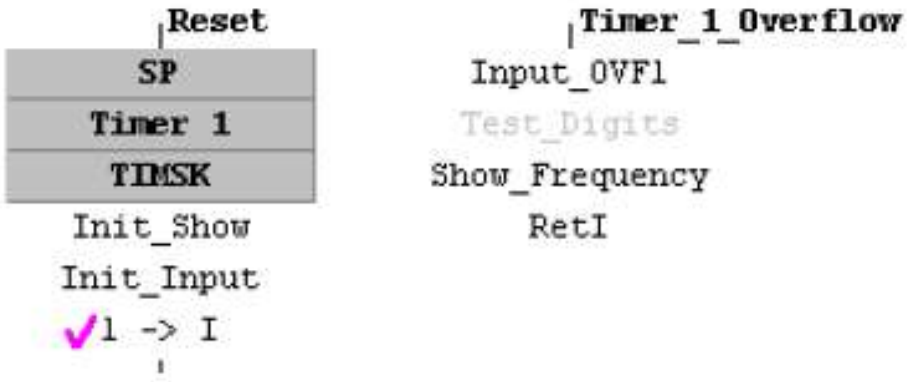

Рис. 4.11. Пример подпрограммы прерывания

Если алгоритм построен таким образом, что может прервать исполнение какого-либо процесса, то в подпрограмме обработки прерываний необходимо позаботиться о сохранении состояния регистров, бит, переменных и т.д., которые могут быть искажены этой подпрограммой.

Таким образом, для того, чтобы создать прерывание, необходимо обеспечить следующее.

1. Создать вершину “Reset”, с которой будет начинаться исполнение программы.

2. Определить указатель стека настройщиком “SP” (обычно это максимальный адрес SRAM).

3. Разрешить данное прерывание. (Для таймеров – это соответствующие биты регистра TIMSK.)

4. Разрешить глобальное прерывание оператором “1 -> I”.

5. Ввести подпрограмму обработки прерывания, которая должна начинаться с вершины с именем прерывания, а заканчиваться обязательно оператором “RetI”.

Для обслуживания прерываний загрузочной секции используйте имена прерываний с префиксом “BOOT_”.

4.2.6. Примеры программ для «Algorithm Builder»

Пример 1. Программа умножения

Программа умножения (рис. 4.12, 4.13, 4.14) двух однобайтных чисел без знака (умножение осуществляется сложением в цикле).

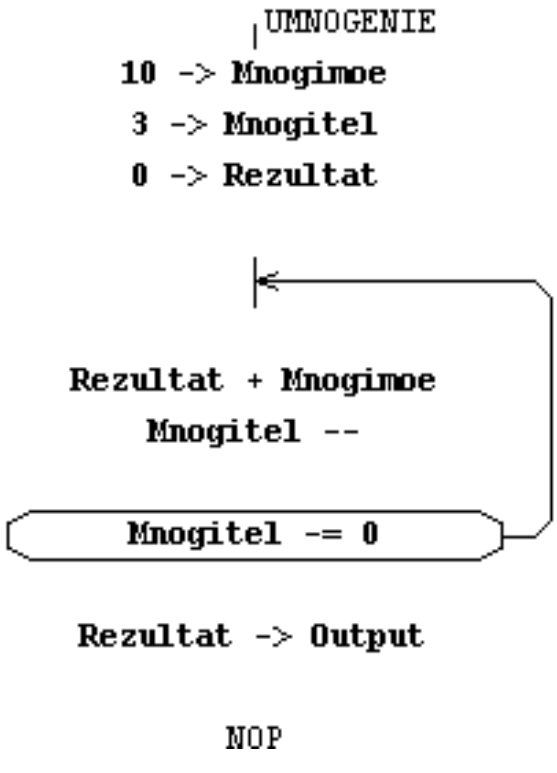

Рис. 4.12. Графическое представление текста программы

Программа умножения двух однобайтных чисел без знака. В SRAM (ОЗУ) по адресу 60 (Mnogitel) помещен множитель в EEPROM

(энергонезависимая память) по адресу 14 (Mnogimoe) помещено множимое, а по адресу 15 (Rezultat) – результат. Результат также выводится в порт B (Output).

SRAM:					
Name	Address	Format	Count	Commentary	
Mnogitel	#h60	Int8		Множитель	
Working registers:					
Name	Index	Format	Commentary		
EEPROM:					
Name	Address	Format	Count	Value	Commentary
Mnogimoe	#h14	Int8			Множимое
Rezultat	#h15	Int8			Результат
I/O registers:					
Name	I/O regis	Commentary			
Output	PortB				
I/O bits:					
Name	I/O bit	Commentary			
Constants:					
Name	Value	Commentary			

Рис. 4.13. Таблица назначения имён и распределения ресурсов

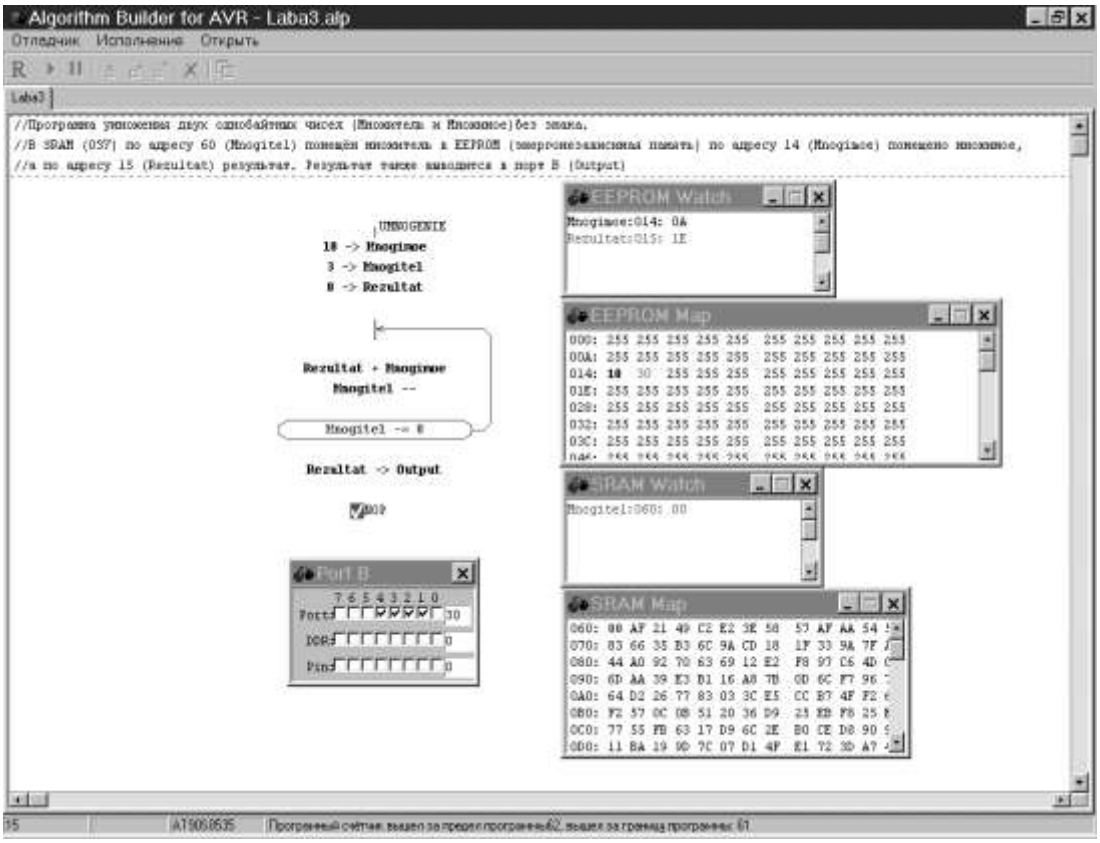

Рис. 4.14. Вид симулятора после выполнения программы

Пример 2

Программа устройства (рис. 4.15, 4.16), анализирующего аналоговый сигнал A и два цифровых сигнала B1 и B2.

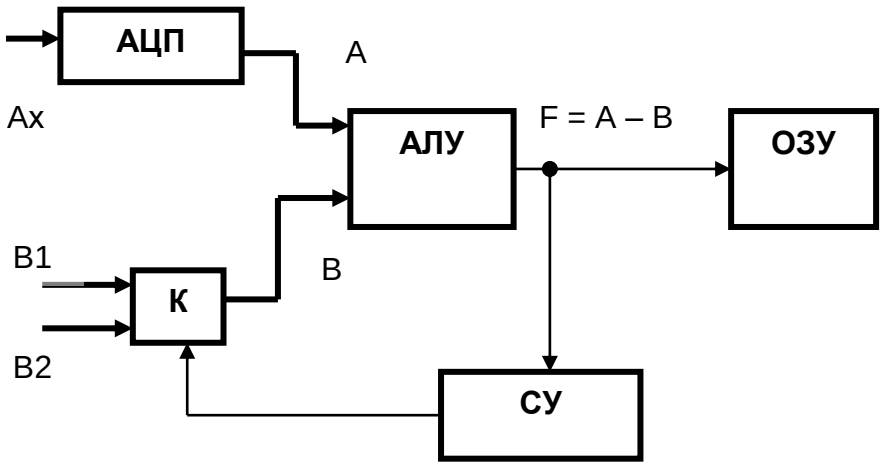

Рис. 4.15. Блок-схема устройства

Таблица 4.6

Код и мнемокод ассемблера

Позиция: код	Мнемокод	Позиция: код	Мнемокод
00000000: E000	ldi r16,#0	00000024: 1026	cpse r2,r6
00000002: 2E00	mov r0,r16	00000026: C002	rjmp 0000002C
00000004: E000	ldi r16,#0	00000028: B258	in r5,$18
00000006: 2E30	mov r3,r16	0000002A: C001	rjmp 0000002E
00000008: E6A0	ldi r26,#96	0000002C: B25B	in r5,$1B
0000000A: E0B0	ldi r27,#0	0000002E: 2C13	mov r1,r3
0000000C: E001	ldi r16,#1	00000030: 1815	sub r1,r5
0000000E: 2E20	mov r2,r16	00000032: 1401	cp r0,r1
00000010: E000	ldi r16,#0	00000034: F018	brlo 0000003C
00000012: 2E60	mov r6,r16	00000036: 921D	st x+,r1
00000014: 2C43	mov r4,r3	00000038: E001	ldi r16,#1
00000016: E000	ldi r16,#0	0000003A: 2E60	mov r6,r16
00000018: B907	out $7,r16	0000003C: 2C01	mov r0,r1
0000001A: EE09	ldi r16,#233	0000003E: E000	ldi r16,#0
0000001C: B906	out $6,r16	00000040: 2E60	mov r6,r16
0000001E: B034	in r3,$4	00000042: CFE8	rjmp 00000014
00000020: 1434	cp r3,r4		
00000022: F3C1	breq 00000014		

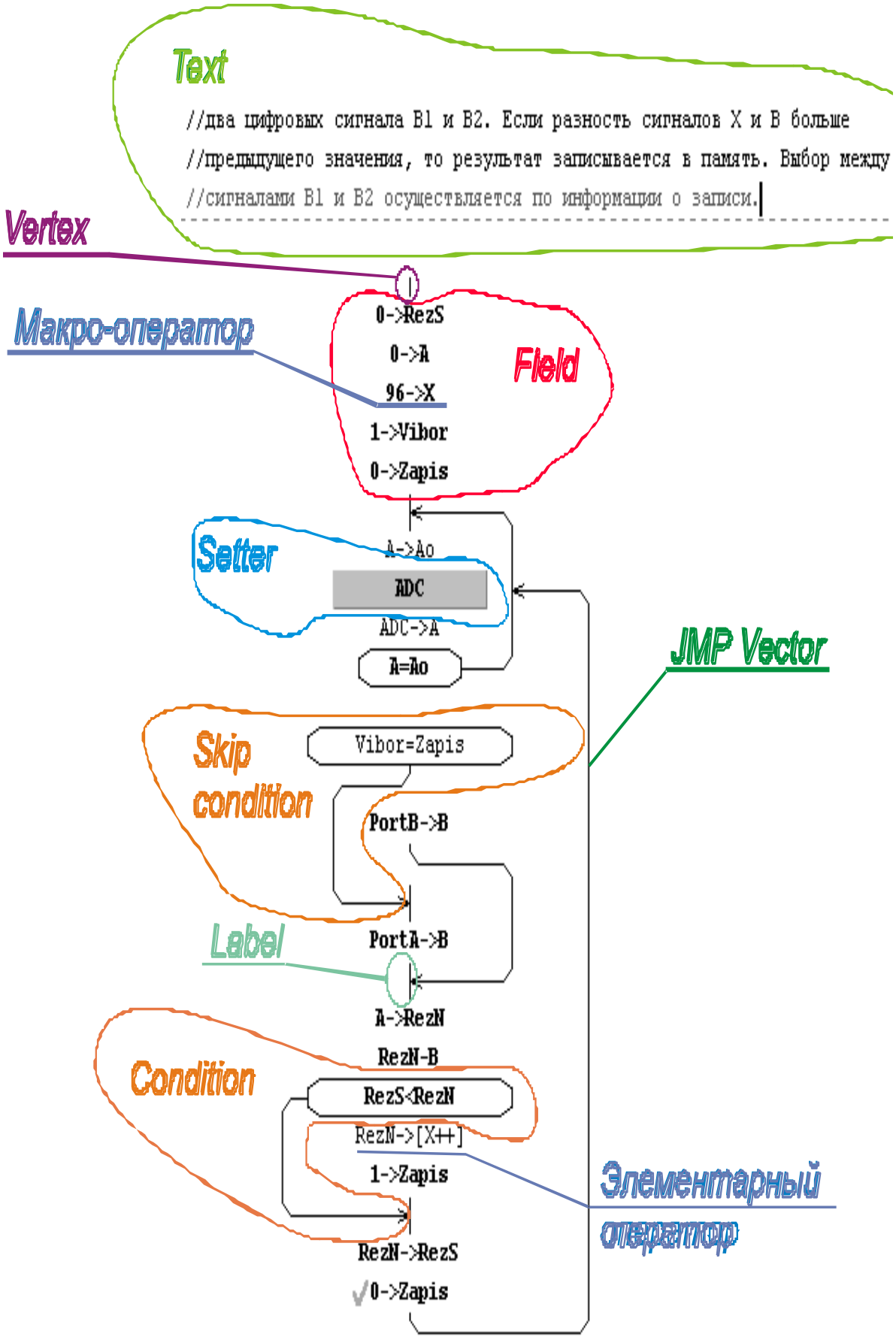

Рис. 4.16. Программа, выполненная в среде «Algorithm Builder»

5. ЛИСТИНГИ ПРИМЕРОВ ПРОГРАММ

Пример 1

```
; Пример 1. Использование инструкций ввода/вывода.
; Программа инициализирует порт B и выводит на вывод PB0 лог. 1.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключение заголовочного файла с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def   temp =R16 ; Теперь к регистру R16 можно обращаться
                 ; по имени temp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
       ldi    temp, 1 << PB0   ; Преобразование номера бита в маску.
       out    DDRB, temp       ; Настраиваем вывод PB0 как выход.
                               ; Вывод всех битов в порт.
       sbi    PORTB, PB0       ; Устанавливаем на выходе PB0 лог. 1.
                               ; Вывод бита в порт.
stop:
       rjmp   stop    ; Бесконечный цикл для остановки выполнения.
```

Пример 2

```
; Пример 2. Использование инструкций условного перехода
; в варианте пропуска следующей инструкции.
; Программа инициализирует порты B (на вывод) и D (на ввод).
; При появлении на выводе PD0 лог. 0 (нажата кнопка 1),
; программа выводит на PB0 лог. 1 (включаем светодиод).
; При появлении на выводе PD1 лог. 0 (нажата кнопка 2),
; программа выводит на PB0 лог. 0 (выключаем светодиод).
; Фактически, это эмуляция RS триггера.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключение заголовочного файла с описаниями устройств МК.
.include "2313def.inc"
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        ; Инициализируем порты.
sbi     DDRB, DDB0          ; Настраиваем вывод PB0 как выход.
        cbi     DDRD, DDD0  ; Настраиваем выводы PD0-1 как вход.
        cbi     DDRD, DDD1  ; В принципе, это не обязательно делать,
                            ; так как при сбросе контроллера
                            ; все порты настраиваются как входы.
        ; ВНИМАНИЕ! При проверке состояния входа следует использовать
        ; порты PINx, а не PORTx.
iloop:  ; Бесконечный цикл для проверки состояния кнопки.
        sbis    PIND, PIND0 ; Если на выводе PD0 лог. 1 (кнопка не
                            ; нажата),
                            ; то пропускаем следующую команду.
        sbi     PORTB, PB0  ; Выводим 1 на PB0.
        sbis    PIND, PIND1 ; Если на выводе PD1 лог. 1 (кнопка не
                            ; нажата),
                            ; то пропускаем следующую команду.
        cbi     PORTB, PB0  ; Выводим 0 на PB0.
        rjmp    iloop       ; Бесконечный цикл.
```

Пример 3

```
; Пример 3. Использование инструкций сравнения и условного перехода.
; Программа инициализирует порты B (на вывод) и D (на ввод).
; При появлении на выводе PD0 лог. 0 (нажата кнопка)
; программа изменяет состояние выводов PB0-PB3
; (инкрементирует четырёх битовое значение на выходе).
; В случае, если кнопка удерживается, состояние выхода не изменяется.
; Фактически, это эмуляция счётчика.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключение заголовочного файла с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
```

```
.def   btn          =R16 ; Теперь к регистру R16 можно обращаться по
                         ; имени btn.
; Регистр хранит 1, если кнопка была нажата и уже обработана.
.def   temp         =R17 ; Теперь к регистру R17 можно обращаться
                         ; по имени temp. Временный регистр.
.equ   btn_mask     =0x01; Маска для битов кнопки (для бита PD0).
.equ   light_mask   =0x0F; Маска для битов светодиодов (для битов PB0-3).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        ; Инициализируем порты.
        ldi     temp, light_mask
        out     DDRB, temp      ; Настраиваем выводы PB0-3 как выход.
        cbi     DDRD, DDD0      ; Настраиваем вывод PD0 как вход.
                                ; В принципе, это не обязательно делать,
                                ; так как при сбросе контроллера
                                ; все порты настраиваются как входы.
        clr     btn             ; Обнуляем регистр.
        ; ВНИМАНИЕ! При проверке состояния входа следует использовать
        ; порты PINx, а не PORTx.
iloop: ; Бесконечный цикл для проверки состояния кнопки.
        in      temp, PIND      ; Считываем значения входов порта D.
        andi    temp, btn_mask  ; Обнуляем все биты, кроме бита 0
                                ; (выделяем PD0).
        cp      temp, btn       ; Сравниваем текущее и предыдущее значение
                                ; бита.
        breq    iloop           ; Если значения совпали, то переход на метку iloop.
        mov     btn, temp       ; Не совпали, запись состояния кнопки.
        cpi     btn, btn_mask   ; Проверка нажатия кнопки (установлен
                                ; ли бит)?
        breq    iloop           ; Совпало (кнопка отпущена), переход на метку
                                ; iloop.
        ; Кнопку нажали, обрабатываем её.
```

```
        in      temp, PORTB     ; Считываем содержимое порта B.
        inc     temp            ; Увеличиваем значение.
        cpi     temp, 0x10 ; Достигли значения 0x10?
        brne    skip_zero   ; Нет, пропускаем обнуление регистра.
        clr     temp            ; Обнуляем значение, если достигли 0x10.
skip_zero:
        out     PORTB, temp     ; Выводим полученное значение обратно
                                ; в порт.
        rjmp    iloop           ; Бесконечный цикл.
```

Пример 4

```
; Пример 4. Использование инструкций условного перехода
; для организации циклов.
; Программа инициализирует порт B и выводит на вывод PB0
; сигнал, скважность которого изменяется со временем.
; При подключении светодиода к выводу PB0 его яркость
; будет изменяться.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключение заголовочного файла с описаниями устройств МК.
.include "2313def.inc"
.def    cnt             =R16 ; Счётчик цикла.
.def    lim             =R17 ; Конечное значение счётчика цикла.
.def    one             =R18 ; Регистр для хранения числа 1.
.def    time            =R19 ; Счётчик проходов для уменьшения
                                ; скорости мерцания (изменения lim).
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        sbi     DDRB, DDB0      ; Настраиваем вывод PB0 как выход.
        sbi     PORTB, PB0      ; Устанавливаем на выходе PB0 лог. 1.
        ldi     lim, 255
        ldi     one, 1
iloop:
        sbi     PORTB, PB0      ; Выводим на PB0 лог. 1
```

```
                                ; (включаем светодиод).
        ; Цикл 1, считает от lim до 0.
        mov   cnt, lim      ; Инициализируем счётчик цикла.
del1:
        sub   cnt, one      ; Декрементируем счётчик.
        brne  del1          ; Если не достигли значения 0,
                            ; то переход в начало цикла.
        cbi   PORTB, PB0    ; Выводим на PB0 лог. 0
                            ; (выключаем светодиод).
        ; Цикл 2, считает от lim до 256 (фактически, до 0).
        mov   cnt, lim      ; Инициализируем счётчик цикла.
del2:
        add   cnt, one      ; Инкрементируем счётчик.
        brne  del2          ; Если не достигли значения 0,
                            ; то переход в начало цикла.
        add   time, one     ; Инкрементируем счётчик проходов.
        brne  skip          ; Если не достигли 0, то переход
                            ; на метку skip.
        dec   lim           ; декрементируем границу цикла.
skip:
        rjmp  iloop         ; Бесконечный цикл.
```

Пример 5

```
; Использование арифметико-логических операций
; Программа инициализирует порты B (на вывод) и D (на ввод).
; Получает на вход с порта D два двухразрядных числа, складывает их
; и выводит полученное значение в порт B.
; Фактически, это эмуляция сумматора.
.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def   mask         =R16 ; Регистр для хранения маски.
```

```
; Регистры R17 и R18 используются для хранения временных
; переменных.
.def   temp1            =R17 ; Теперь к регистру R17 можно обращаться
                             ; по имени temp1.
.def   temp2            =R18 ; Теперь к регистру R18 можно обращаться
                             ; по имени temp2.
; Маска для кнопки (для битов PD0-1).
.equ  btn_mask   =0x03
; Маска для светодиодов.
.equ  light_mask =0x0F
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        ; Инициализируем порты.
        ldi     temp1, light_mask       ; Загрузка маски в регистр.
        out     DDRB, temp1     ; Настройка PB0-PB3 на вывод данных.
        ldi     mask, btn_mask  ; Загрузка маски в регистр.
; ВНИМАНИЕ! При проверке состояния входа следует использовать
        ; порты PINx, а не PORTx.
iloop: ; Бесконечный цикл для проверки состояния кнопки.
        in      temp1, PIND     ; Считываем значения входов порта D.
        mov     temp2, temp1    ; Запоминаем его во временном регистре.
        and     temp1, mask     ; Выделяем значение двух младших битов,
                                ; получаем значения состояний кнопок 1-2.
        eor     temp1, mask     ; Инвертируем значение
                                ; (так как нажатая кнопка == лог. 0).
        lsr     temp2
        lsr     temp2           ; Сдвигаем значение на два бита вправо,
        and     temp2, mask     ; получаем значение кнопок 3-4.
        eor     temp2, mask     ; Инвертируем значение
                                ; (так как нажатая кнопка == лог. 0).
        add     temp1, temp2    ; Производим сложение чисел.
        out     PORTB, temp1    ; Выводим полученное значение в порт.
        rjmp    iloop           ; Бесконечный цикл.
```

Пример 6

```
; Использование многобайтовых операций.
; Программа инициализирует порт B (на вывод).
; На выходе генерируется сигнал с большим периодом (~1 с).
; При генерации используется цикл на ~500.000 шагов для задержки.
; Количество шагов цикла рассчитывается исходя из того,
; что тело цикла состоит из семи команд, на выполнение каждой
; из которых требуется один такт. При тактовой частоте 6 МГц
; выполнение тела цикла происходит за ~1мкс, следовательно,
; цикл должен состоять из ~500.000 шагов (0x07A120).
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def   cnt1        =R16 ; Счетчик цикла (байт 1).
.def   cnt2        =R17 ; Счетчик цикла (байт 2).
.def   cnt3        =R18 ; Счетчик цикла (байт 3).
.def   val1        =R19 ; Конечное значение счётчика (байт 1).
.def   val2        =R20 ; Конечное значение счётчика (байт 2).
.def   val3        =R21 ; Конечное значение счётчика (байт 3).
.def   one         =R22 ; Регистр для хранения значения 1.
.def   zero        =R23 ; Регистр для хранения значения 0.
.def   temp        =R24 ; Регистр для хранения временного значения.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        ; Инициализируем порты.
        sbi    DDRB, DDB0      ; Настраиваем вывод PB0 как выход.
        clr    cnt1            ; Чистим регистры.
        clr    cnt2
        clr    cnt3
        ldi    val1, 0x20      ; Загружаем значение границы цикла.
        ldi    val2, 0xA1
```

```
        ldi     val3, 0x07
        clr     zero                ; Загружаем 0 и 1.
        ldi     one, 1
iloop:
        ; Инкремент многобайтового значения счётчика цикла.
        add     cnt1, one     ; Увеличим значение счётчика.
        adc     cnt2, zero    ; Добавляем перенос к следующему байту.
        adc     cnt3, zero    ; И ещё раз добавляем.
        ; Проверка на совпадение с конечным значением счётчика цикла.
        cp      cnt1, val1    ; Просто сравниваем байт 1.
        cpc     cnt2, val2    ; Сравниваем значения с учётом переноса.
        cpc     cnt3, val3    ; Сравниваем значения с учётом переноса.
        brne    iloop         ; Если значения не совпали,
                              ; то переходим в начало цикла.
        clr     cnt1          ; Обнуление регистров.
        clr     cnt2
        clr     cnt3
        in      temp, PORTB         ; Считывание значения из порта B.
        eor     temp, one     ; Инвертируем бит 0.
        out     PORTB, temp ; Вывод полученного значения обратно
                                ; в порт.
        rjmp    iloop         ; Бесконечный цикл.
```

Пример 7

```
; Использование инструкций доступа к переменным и
; массивам, расположенным в ОЗУ.
; Примечание: данный пример является лишь иллюстрацией,
; его следует запускать в симуляторе.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def    temp1           =R0
.def    temp2           =R1
```

```
.def    i               =R16
.def    val             =R17
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
        ; c = a + b
        lds     temp1, a        ; Загружаем переменную из ОЗУ в регистр.
        lds     temp2, b
        add     temp1, temp2    ; Складываем значения.
        sts     c, temp1        ; Записываем сумму в ОЗУ.
        ; Записываем в регистры R0-R9 число 123.
        ; Примечание: по адресу 0 находится регистровый файл.
        clr     ZL              ; byte *q;
        clr     ZH              ; q = 0;
        ldi     i, 10           ; i = 10;
        ldi     val, 123        ; val = 123;
lp:                             ; do {
        st      Z+, val         ;       *q++ = val;
; Инструкции dec и brne работают в паре, их не следует разделять,
        ; dec уменьшает значение и модифицирует флаги,
        ; brne проверяет значения флагов.
        dec     i               ;       i--;
        brne    lp              ; while (i != 0);
        ; Заполняем массив arr, находящийся в ОЗУ:
        ; arr[0] = 129, arr[1] = 128... arr[9] = 120.
        ; Примечание: по адресу 0x60 находится начало ОЗУ,
        ; массив arr находится в ОЗУ по адресу 0x63.
        ldi     ZL, low(arr)            ; byte *p;
        ldi     ZH, high(arr)   ; p = &arr;

        ldi     i, 10           ; i = 10;
        ldi     val, 129        ; val = 129;
lp2:                            ; do {
        st      Z+, val         ;       *p++ = val;
```

```
        dec   val           ;       val--;
        dec   i             ;       i--;
        brne  lp2           ; } while (i > 0);
stop:
        rjmp  stop
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента данных.
.DSEG
; Объявление переменных:
a:      .BYTE       1           ; byte a;
b:      .BYTE       1           ; byte b;
c:      .BYTE       1           ; byte c;
arr:    .BYTE       10          ; byte arr[10];
```

Пример 8

```
; Использование стека и вызов подпрограмм.
; Это пример 6, представленный с использованием подпрограмм.
; Также изменяется значение сразу четырёх битов порта.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def   cnt1         =R16 ; Счетчик цикла (байт 1).
.def   cnt2         =R17 ; Счетчик цикла (байт 2).
.def   cnt3         =R18 ; Счетчик цикла (байт 3).
.def   val1         =R19 ; Конечное значение счётчика (байт 1).
.def   val2         =R20 ; Конечное значение счётчика (байт 2).
.def   val3         =R21 ; Конечное значение счётчика (байт 3).
.def   one          =R22 ; Регистр для хранения значения 1.
.def   zero         =R23 ; Регистр для хранения значения 0.
.def   temp         =R24 ; Регистр для хранения временного значения.
; Маска для светодиодов.
.equ   light_mask   =0x0F
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; Начало сегмента кода.
.CSEG
        ldi     temp, low(RAMEND)
        out     SPL, temp  ; Настраиваем стек на конец ОЗУ.
        ldi     temp, light_mask
        out     DDRB, temp      ; Настраиваем PB0-PB3 как выходы.
        rcall   initproc        ; Вызов процедуры инициализации.
iloop:
        rcall   delayproc   ; Вызов процедуры задержки.
        rcall   portproc    ; Вызов процедуры работы с портом.
        rjmp    iloop       ; Бесконечный цикл.
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура инкрементирования значения порта.
; Используются биты порта PB0-PB3.
portproc:
        in      temp, PORTB     ; Читаем значение битов порта B.
        inc     temp            ; Инкрементируем значение.
        cpi     temp, 0x10 ; Сравниваем с 0x10.
        brne    skip            ; Если значения не равны, то переходим
                                ; к метке skip.
        clr     temp            ; Очистка регистра.
skip:
        out     PORTB, temp     ; Выводим значение обратно в порт B.
        ret                     ; Выходим из процедуры.
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура для задержки выполнения. Для организации задержки
; используется длинный цикл с многобайтовым счётчиком.
delayproc:
loop:
        ; Инкремент многобайтового значения счётчика цикла.
        add     cnt1, one   ; Увеличим значение счётчика.
        adc     cnt2, zero  ; Добавляем перенос к следующему байту.
        adc     cnt3, zero  ; И ещё раз добавляем.
        ; Проверка на совпадение с конечным значением счётчика цикла.
```

```
        cp      cnt1, val1   ; Просто сравниваем байт 1.
        cpc     cnt2, val2   ; Сравниваем с учётом переноса.
        cpc     cnt3, val3   ; Сравниваем старший байт.
        brne    loop         ; Если значения не совпали,
                             ; то переходим в начало цикла.
        clr     cnt1         ; Чистим регистры.
        clr     cnt2
        clr     cnt3
        ret                  ; Выходим из процедуры.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура инициализации.
initproc:
        clr     cnt1         ; Чистим регистры.
        clr     cnt2
        clr     cnt3
        ldi     val1, 0x40   ; Загружаем значение границы цикла.
        ldi     val2, 0x42
        ldi     val3, 0x10
        clr     zero         ; Загружаем 0 и 1.
        ldi     one, 1
        ret                  ; Выходим из процедуры.
```

Пример 9

```
; Обработчик прерывания таймера.
; Данная программа переключает светодиод на выводе PB0
; с частотой ~1Гц.
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
.def   temp1            =R16
.def   temp2            =R17
.def   cnt          =R18
; Задаем константы.
.equ   CK1024           =0x05       ; Коэффициент предварительного
                                    ; делителя таймера
```

```
.equ  TIMER0_MASK  =0x02        ; Маска разрешения
                                ; прерывания таймера.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Начало сегмента кода.
.CSEG
; Таблица векторов прерываний.
        rjmp reset          ; Этот вектор сброса микроконтроллера.
.org    OVF0addr            ; Пропускаем ненужные обработчики.
        rjmp timer          ; Это прерывание от таймера 0.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Обработчик вектора reset,
; вызывается при сбросе микроконтроллера.
reset:
        sbi     DDRB, DDB0      ; Настраиваем вывод PB0 как выход.
        sbi     PORTB, PB0      ; Устанавливаем на выходе PB0 лог. 1.
        ; Настраиваем стек.
        ldi     temp1, low(RAMEND)
        out     SPL, temp1
        clr     cnt             ; Обнуляем счётчик заходов в таймер.
        ldi     temp1, CK1024   ; Загружаем значение в предварительный
        out     TCCR0, temp1    ; делитель частоты таймера.
        ldi     temp1, TIMER0_MASK
        out     TIMSK, temp1    ; Разрешаем работу таймера.
        sei                     ; Разрешаем все прерывания.
iloop:
        rjmp iloop              ; Цикл.
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Обработчик прерывания таймера,
; вызывается всякий раз при переполнении таймера.
timer:
        inc     cnt             ; Увеличиваем значение счётчика.
        cpi     cnt, 11         ; Если счётчик не достиг значения 11,
        brne    tquit           ; то осуществляется переход к метке tquit.
        clr     cnt             ; Если достиг, обнуляем счётчик.
```

```
        in      temp1, PORTB    ; Получаем значение порта B.
        ldi     temp2, 1<<PB0   ; Загружаем маску для инвертируемого бита.
eor     temp1, temp2        ; Инвертирование бита PB0.
out     PORTB, temp1    ;Вывод инвертированного значения в порт.
tquit:
        reti                ; Выход из обработчика прерывания.
```

6. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа № 1
«Инструментальная среда разработки – AVR Studio»

Цель работы

1. Изучить инструментальную среду программирования микроконтроллеров AVR Studio.

2. Ознакомится с интерфейсом и принципом работы в среде программирования микроконтроллеров.

Задание

1. Создайте проект в среде разработки.
2. Произведите компиляцию программы.
3. Устраните ошибки.
4. Произведите тестирование программы.
5. Создайте отчёт.

Методика выполнения работы

1. Ознакомьтесь со средой разработки AvrStudio. Для экспериментов используйте листинг примера 1.

2. Создайте проект (см. **Приложение**), введите текст листинга в автоматически созданном файле *.asm, сохраните файл, ассемблируйте проект. Убедитесь в том, что ассемблирование прошло успешно.

3. Воспользуйтесь отладчиком для пошагового выполнения программы. Для контроля выполнения программы используйте окно просмотра значений портов ввода/вывода.

Контрольные вопросы

Ответьте на контрольные вопросы и проиллюстрируйте их на примере архитектуры микроконтроллеров AVR.

1. Для чего необходим заголовочный файл?

2. С какими операндами могут работать арифметико-логические инструкции?

3. Допускают ли микроконтроллеры AVR обработку операндов разрядностью более восьми бит?

4. Какие типы инструкций условного перехода существуют в системе команд микроконтроллера?

5. Адресное пространство микроконтроллеров является линейным или сегментированным?

6. Какого типа прерывания допустимы в микроконтроллерах AVR?

Лабораторная работа № 2
«Архитектура и аппаратные возможности микроконтроллеров AVR»

Цель работы

1. Изучить основы построения аппаратных комплексов на основе микроконтроллеров.

2. Изучить устройство, характеристики, возможности микроконтроллеров AVR (Atmel).

3. Изучить инструментальную среду симуляции работы микроконтроллеров.

4. Ознакомиться с принципом разработки проектов на основе микроконтроллеров.

Задание

1. Создайте проект в среде разработки (см. Приложение).

2. Произведите компиляцию программы.

3. Разработайте структуру микропроцессорной системы.

4. Создайте отчёт.

Методика выполнения работы

1. Ознакомьтесь с интерфейсом среды разработки AvrStudio.

2. Для работы используйте листинг примера 1. П. 5.

3. Выберете необходимый микроконтроллер, согласно техническим характеристикам, для реализации предложенной программы.

4. Создайте проект, введите текст листинга в автоматически созданном файле *.asm, сохраните файл, ассемблируйте проект. Убедитесь в том, что ассемблирование прошло успешно.

5. Воспользуйтесь отладчиком для пошагового выполнения программы. Для контроля выполнения программы используйте интерфейсы просмотра РОН, памяти и значений портов ввода/вывода.

Контрольные вопросы

Ответьте на контрольные вопросы и проиллюстрируйте их на примере архитектуры микроконтроллеров AVR.

1. Расшифруйте аббревиатуру ОМЭВМ и объясните смысл названия.

2. К какому типу архитектуры ЭВМ относятся микроконтроллеры AVR?

3. За счёт каких технических особенностей контроллеры AVR имеют высокую производительность?

4. Какие виды памяти используются в микроконтроллерах AVR?

5. Что такое «регистровый файл»?

6. Из каких устройств состоит система ввода-вывода микроконтроллера?

Лабораторная работа № 3
«Синтаксис языка ассемблер»

Цель работы

1. Изучить правила написания программ на языке ассемблера.

2. Изучить функции отладки инструментальной среды программирования микроконтроллеров.

Задание

1. Создайте проект в среде разработки. Введите следующий листинг, найдите в нём ошибки, выполните отладку исправленной программы:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; Пример с ошибками;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Программа инициализирует порты B (на вывод) и D (на ввод).
```

```
; При появлении на выводе PD0 лог. 0 (нажата кнопка)
; включает светодиод и останавливается.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подключаем заголовочный файл с описаниями устройств МК.
.include "2313def.inc"
; Задаем символьные имена регистров.
    .def     btn          R16   ; Теперь к регистру R16 можно
                                ; обращаться
                                ; по имени btn. Регистр хранит 1, если
                                ; кнопка была нажата и уже обработана.
.equ  btn_mask   =0x01; Маска для битов кнопки (для бита PD0).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.CSEG                           ; Начало сегмента кода.
        sbi   DDIRB, PB0        ; Инициализация портов.
        clr   btn               ; Обнуление регистра.
iloop:                          ; Опрос кнопки.
        in    btn, PIND         ; Считывание входов порта D.
        and   btn, btn_mask     ; Выделяем бит 0 (значение PD0).
        breq  iloop             ; Если кнопка отпущена, то
                                ; переход на метку iloop.
                                ; Активация.
        sbi   PORTB, PB0        ; Включение светодиода.
stop:         jmp   stop        ; Бесконечный цикл.
```

2. Произведите тестирование программы.

3. Создайте отчёт, содержащий в том числе: листинг и блок схемы алгоритма исправленной программы.

Методика выполнения работы

1. Ознакомьтесь со средой разработки AVRStudio. Для экспериментов используйте листинг примера.

2. Создайте проект (см. Приложение), введите текст программы с ошибками в файле *.asm, сохраните файл, ассемблируйте проект.

3. Произведите компиляцию кода программы. Исправьте имеющиеся синтаксические ошибки.

4. Убедитесь в том, что ассемблирование прошло успешно.

5. Воспользуйтесь отладчиком для пошагового выполнения программы. Для контроля выполнения программы используйте окна просмотра значений устройств микроконтроллера.

Контрольные вопросы

1. Для чего необходим заголовочный файл?

2. С какими операндами могут работать арифметико-логические инструкции?

3. Допускают ли микроконтроллеры AVR обработку операндов разрядностью более восьми бит?

4. Какие типы инструкций условного перехода существуют в системе команд микроконтроллера?

Лабораторная работа № 4
«Циклы, условия»

Цель работы

1. Изучить возможности графической среды разработки Algorithm Builder для микроконтроллеров AVR (Atmel).

2. Получить навыки разработки программ в графической среде.

3. Изучить возможности системы команд ветвлений и условных и безусловных переходов.

Задание

1. Разработать программу проверки состояния выключателя, в случае замыкания, производится вывод в порт микроконтроллера, циклически повторяющийся двоичный код (номер варианта увеличенный на 10).

2. Реализовать возможность изменения частоты в процессе вывода числа в порт.

Методика выполнения работы

1. Ознакомиться с элементами конструкции алгоритма в среде программирования.

2. Изучить систему условных операторов и методы их применения.

3. Изучить примеры организации графического отображения условных операторов.

4. Изучить методы редактирования программы (алгоритма) в инструментальной среде AB.

5. Записать и выполнить примеры программ, приведённые в п. 4.2.5.

6. Разработать программу на основе выданного задания.

Контрольные вопросы

1. Какие типы инструкций условного перехода существуют в системе команд микроконтроллера?

2. Перечислите возможные условия переходов.

3. Как называется графический элемент для реализации операторов условных переходов?

4. Приведите пример оператора условного пропуска следующего оператора. Поясните на примере.

Лабораторная работа № 5
«Обработка массива данных»

Цель работы

1. Изучить возможности адресации к памяти микроконтроллеров AVR (Atmel).

2. Изучить данные примеры программ «Algorithm Builder».

3. Изучить методы и алгоритмы обращения к массивам данных, размещаемых в памяти SRAM и EEPROM микроконтроллеров AVR.

Задание

1. Разработать программу согласно выданному заданию.

2. Адрес первой ячейки памяти соответствует номеру варианта.

3. Произвести инициализацию области памяти SRAM, объёмом равному номеру варианта, увеличенному на 10.

4. Переместить массив данных из EEPROM в SRAM при нечётном номере варианта и в обратном направлении при чётном.

5. Произвести поиск минимального значения элемента массива, хранящегося в памяти, при переносе данных (п. 3).

Методика выполнения работы

1. Изучить возможности системы команд передачи данных в микроконтроллерах AVR.

2. Записать и выполнить примеры программ обработки массивов данных.

3. Изучить способ подключения данных из внешнего файла.

4. Подготовить массив данных во внешнем файле согласно заданию.

5. Разработать программу передачи массива данных между областями памяти микроконтроллера SRAM и EEPROM на основе приведённых примеров 1 и 2.

Пример 1. Инициализации массива данных в SRAM

Заголовок секции SRAM:

В этой секции предусмотрены следующие поля:

Name – объявляемое имя переменной (ячейки памяти);

Address(необязательный параметр) – константа, определяющая конкретное значение адреса. По умолчанию – следующий за предыдущим либо $60 в начале компиляции;

Format (необязательный параметр) – формат переменной. По умолчанию принимается однобайтный формат. Многобайтные форматы используются в макро-операторах;

Count (необязательный параметр) – число резервируемых ячеек. По умолчанию принимается равное 1.

SRAM:

Name	Address	Format	Count	Commentary
EditIndex				однобайтная переменная
ShowMode		Word		двухбайтная переменная
Reg		Int24	4	четыре трехбайтных переменных
LCD_Page			16	16 однобайтных ячеек
Phase	$100			однобайтная переменная по адресу $100
XArray	@Phase+4		24	24 однобайтных переменных по адресу $104

Рис. 6.1. Заголовок секции SRAM

В операторах с непосредственной адресацией SRAM “[#]->R” и “R->[#]”, имя переменной может быть использовано вместо “[#]”. Имя переменной с префиксом @ является константой, содержащей ее фи-

зический адрес SRAM. Приведенные ниже примеры будут откомпилированы с одинаковым результатом:

[$100] -> r0, Phase -> r0, [@Phase] -> r0

Объявленные многобайтные переменные могут быть использованы в макрооператорах, о которых будет изложено ниже. Если переменная объявлена как массив (Count>1, например, “LDC_Page”), то ее имя будет указывать на первый байт массива. Для непосредственно адресации произвольного элемента массива используйте смещение адреса. Например, если необходимо копировать r0 в пятый элемент массива, то следует записать:

r0->[@LCD_Page+5]

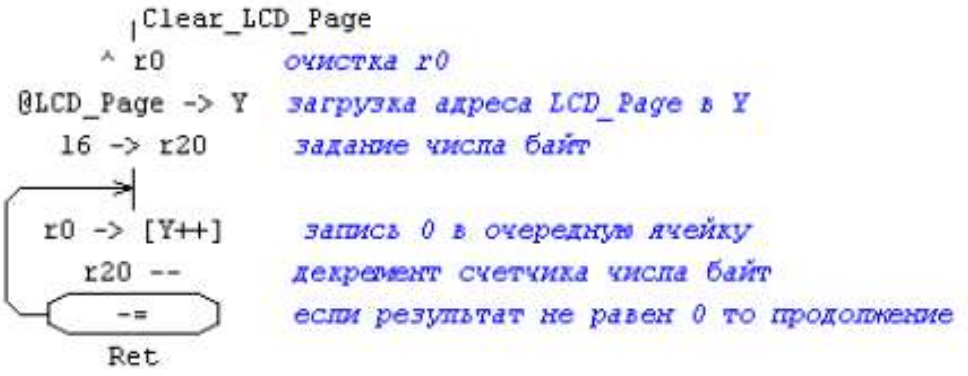

Рис. 6.2. Инициализация массива данных в SRAM

Пример 2. Копирование массива EEPROM в массив SRAM

Секция объявления переменных EEPROM

Заголовок секции: EEPROM:

В этой секции предусмотрены следующие поля:

Name – назначаемое имя переменной;

Address (необязательный параметр) – константа, определяющая конкретное значение адреса. По умолчанию – следующий после предыдущего либо 0 в начале секции;

Format (необязательный параметр) – формат ячейки. По умолчанию принимается однобайтный формат. При необходимости можно задать многобайтный формат;

Count (необязательный параметр) – число резервируемых ячеек. По умолчанию принимается равным 1;

Value (необязательный параметр) – начальные значения, представленные в виде константы или массива констант через запятую (если Count>1).

Альтернативно, начальные значения могут быть загружены из файла директивой “Load: FileName”, где FileName – имя подгружаемого файла. См. справку «Непосредственное подключение файла данных». Если данное поле не заполнено, то область переменной будет заполнена $FF.

EEPROM:

Name	Address	Format	Count	Value	Comment
EE_Cell					однобайтная ячейка
EE_LCD_Page			16	1,2,3,4,5,6,7,8	16 однобайтных ячеек
InitValue		Word			двухбайтная ячейка
StartValue		Word	3	259,3331,0	четыре двухбайтных ячейки с загрузкой
					начальных значений
InitScale		DWord		#h27773F	одна четырехбайтная ячейка с загрузкой
					начального значения
FileTable		Word		Load: Table.db	двухбайтный массив с загрузкой из файла

Рис. 6.3. Заголовок секции EEPROM

В алгоритме объявленное имя может быть использовано в макрооператорах, реализующих операции с EEPROM, например: InitValue -> X, 1875 -> InitValue. Имя переменной с префиксом “@“ является константой, содержащей ее физический адрес в EEPROM.

```
          Read_From_EEPROM
@EE_LCD_Page.0 -> XL  Загрузка начального адреса EEPROM в X
@EE_LCD_Page.1 -> XH
 @LCD_Page.0 -> YL    Загрузка начального адреса SRAM в Y
 @LCD_Page.1 -> YH
     10 -> r20        Задание числа байт (10)
    ┌──►┐
    XL -> EEARL       Загрузка текущего адреса EEPROM
    XH -> EEARH          из X в двойной регистр EEAR
     1 -> EERE        Запись 1 в бит EERE
   EEDR -> r16        Копирование EEDR в r16
  r16 -> [Y++]        Копирование r16 в SRAM по адресу в Y
     X + 1            Увеличение текущего адреса EEPROM (в X) на 1
     r20 --           Уменьшение счетчика числа байт на 1
    └─ -= ─┘          Если результат не равен 0, то возврат к началу цикла
       Ret
```

Рис. 6.4. Копирование массива EEPROM в массив SRAM

Пример копирования массива EEPROM “EE_LCD_Page” в массив SRAM “LCD_Page” приведен на рис. 6.4.

Пример 3. Копирование массива SRAM в массив EEPROM

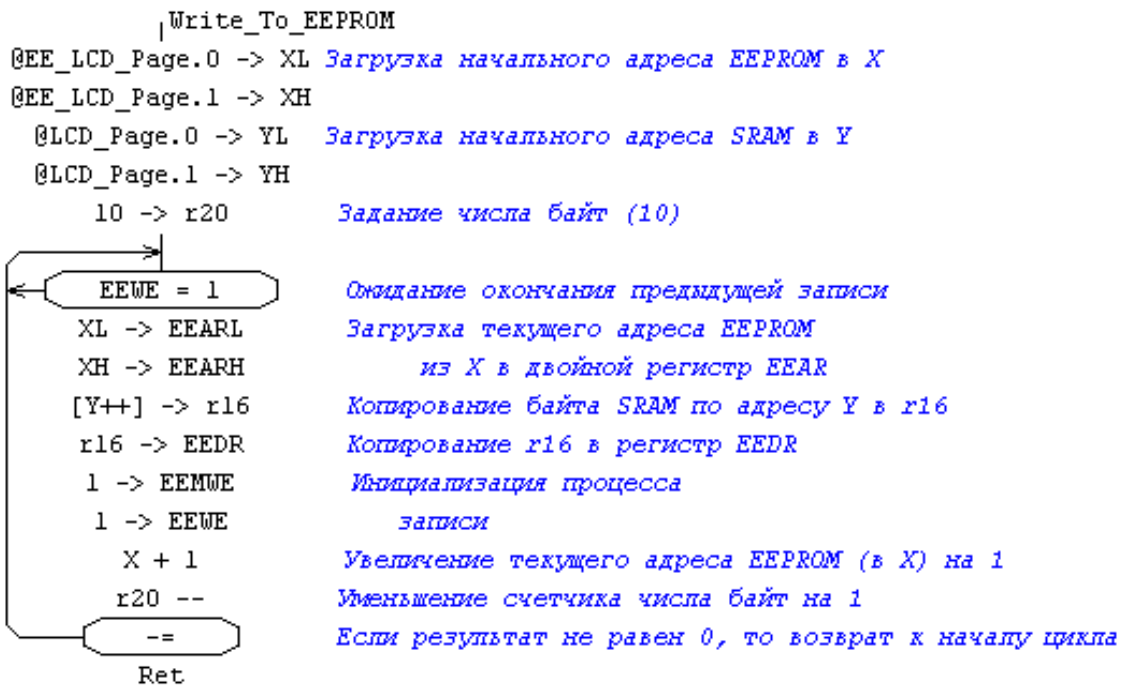

Рис. 6.5. Копирование массива SRAM в массив EEPROM

Контрольные вопросы

1. Какие регистры предназначены для пересылки данных?
2. Какие виды памяти используются в МК AVR?
3. Какие виды инструкции адресации к памяти имеются в системе команд AVR?
4. Через какие регистры осуществляется доступ EEPROM?
5. Как производится непосредственное подключение файла данных?

Лабораторная работа № 6
«Арифметико-логические операции»

Цель работы

Изучить арифметико-логические операции.

Задание

Произвести операцию сложения двух массивов чисел. Данные вводить через порты.

Методика выполнения работы

1. Ознакомиться с перечнем операции копирования и арифметико-логических преобразований.

2. Изучить возможности системы арифметико-логических команд в микроконтроллерах AVR.

3. Записать и выполнить пример № 1 программы из п.п. 4.2.5.

4. Разработать программу передачи массива данных между областями памяти микроконтроллера SRAM и EEPROM.

Контрольные вопросы

1. Приведите пример операторов умножения двух рабочих регистров с учетом знака.

2. В каких регистрах может размещаться результат умножения.

3. Приведите пример побитной логической операции «исключающее ИЛИ» двух рабочих регистров.

4. Приведите пример представления констант в виде алгебраических выражений.

Лабораторная работа № 7
«Структурирование программ. Подпрограммы»

Цель работы

1. Поиск минимального значения элемента массива.

2. Изучить методы и алгоритмы обращения к массивам данных, размещаемых в долговременной и оперативной памяти.

Задание

Результатом работы программы должен быть номер минимального элемента массива.

Методика выполнения работы

1. Произвести чтение массива из файла данных.

2. На основе лабораторной работы № 5, оформить поиск как подпрограмму.

3. Для передачи указателя на массив использовать регистр Z.

4. Применить запись с использованием макрооператоров, показанных на рисунке 6.6.

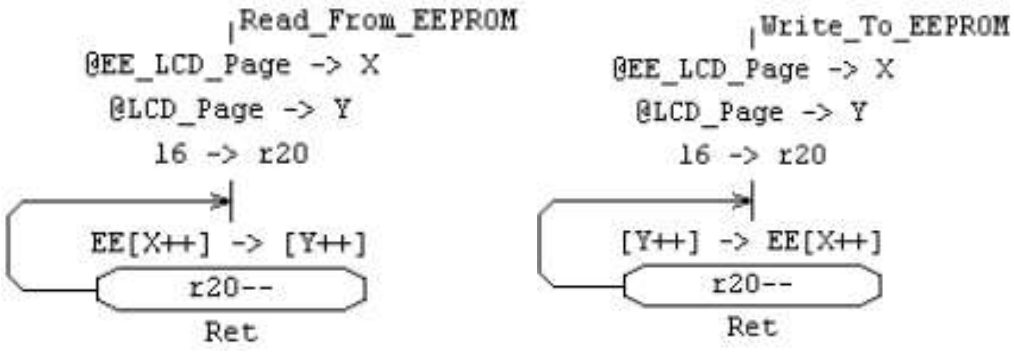

Рис. 6.6. Оптимизированные программы записи массивов данных

Контрольные вопросы

1. Как осуществляется вызов подпрограммы?

2. Приведите пример подпрограмм с параметрами.

3. Как осуществляется подключение к проекту алгоритмов из других файлов?

4. Как подключить к программе файл данных?

Лабораторная работа № 8
«Программное прерывание»

Цель работы

Изучение принципа организации программного прерывания.

Задание

1. Выполните задание из лабораторной работы № 4, используя прерывание по таймеру для осуществления момента вывода данных в порт.

2. При работе использовать источник [2]

Методика выполнения работы

Для создания прерывания, необходимо:

1. Создать вершину “Reset”, с которой будет начинаться исполнение программы.

2. Определить указатель стека настройщиком “SP” (обычно это максимальный адрес SRAM).

3. Разрешить данное прерывание. (Для таймеров – это соответствующие биты регистра TIMSK.)

4. Разрешить глобальное прерывание оператором “1 -> I”.

5. Ввести подпрограмму обработки прерывания, которая должна начинаться с вершины с именем прерывания, а заканчиваться обязательно оператором “RetI”.

Для обслуживания прерываний загрузочной секции используйте имена прерываний с префиксом “BOOT_”.

Контрольные вопросы

1. Какие виды прерываний применяются в микропроцессорной технике?

2. Какие данные содержит вектор прерывания?

3. Что такое «метка обслуживания прерываний»?

4. Что такое «таблица прерываний»?

5. Какой флаг разрешает прерывания?

6. Какой период времени соответствует отклику на прерывание?

7. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

Аппаратные характеристики микроконтроллеров

1. К какому типу относится архитектура микроконтроллеров AVR?

2. Какова разрядность микроконтроллеров AVR?

3. С помощью, каких средств может быть загружена память программ микроконтроллера?

4. Какие виды памяти применяются в микроконтроллерах AVR?

5. Что такое регистровый файл?

6. Как производится защита памяти от несанкционированного доступа?

7. Приведите технические характеристики тактового генератора.

8. Назовите функции сторожевого таймера (WATCHDOG).

9. Приведите основные функции таймер-счётчиков.

10. Какова нагрузочная способность портов ввода/вывода?

11. Какова особенность построения портов ввода/вывода с тремя битами контроля и управления?

12. Перечислите средства обработки аналоговых сигналов.

13. Приведите технические характеристики аналогового компаратора.

14. Какова организация аналого-цифрового преобразователя в микроконтроллерах AVR.

15. Перечислите режимы управления энергопотреблением.

Программные характеристики микроконтроллеров

16. Для чего необходим заголовочный файл?

17. С какими операндами могут работать арифметико-логические инструкции?

18. Допускают ли микроконтроллеры AVR обработку операндов разрядностью более восьми бит?

19. Какие типы инструкций условного перехода существуют в системе команд микроконтроллера?

20. Адресное пространство микроконтроллеров является линейным или сегментированным?

21. Какого типа прерывания допустимы в микроконтроллерах AVR?

22. Где допустимо располагать таблицу векторов прерываний?

23. Сохраняется ли в стеке регистр флагов при вызове прерывания?

Дополнительные вопросы для самоконтроля

1. Какой блок в ЭВМ должен рассматриваться как ЦП?

2. Перечислите три типа связей в микроЭВМ.

3. Адресная шина является однонаправленной, шина ____, напротив, двунаправленная.

4. Обычно постоянные программы располагаются в БИС, называемой ____.

5. Временные данные и программы располагаются в БИС, называемой (ОЗУ, ПЗУ).

6. Размещение данных в микроЭВМ выполняется _____ (временно, постоянно).

7. Размещение программ в ПЗУ выполняется (временно, постоянно).

8. Ввод или вывод информации в (из) микроЭВМ выполняется с использованием (порта, датчика времени).

9. Список команд в микроЭВМ составляет _____.

10. Программа помещается внутри микроЭВМ в памяти ____.

11. Большинство команд микроЭВМ состоит из двух частей – операции и _____.

12. Команды, составляющие программу микроЭВМ, обычно выполняются (последовательно, случайно).

13. Для выполнения каждой команды МП действует в последовательности: ____, ____.

14. Какие, по меньшей мере, пять основных устройств входят в типовую микроЭВМ?

15. Список команд составляют _____ (программные, аппаратные) средства микроЭВМ.

16. Интегральные схемы, применяемые в ЭВМ для организации памяти со сменяющимися данными, называются _____.

17. Команды в программной памяти _____ (постоянны, сменяемы).

18. Центральный процессор получает доступ к ячейке памяти посредством шины _____.

19. Кодированная информация передается из аккумулятора МП в ячейку памяти данных посредством шины _____.

20. МикроЭВМ содержит, по меньшей мере, устройства ввода, вывода, центральный процессор и _____ программ и данных.

21. Центром всех операций микроЭВМ является _____ (МП, ОЗУ, ПЗУ).

22. Шина ____ (адреса, данных) является однонаправленной.

23. Посредством 16 линий адресной шины можно получить доступ к _____ (16384, 65536) ячейкам памяти и портам ВВ.

24. Назовите, по меньшей мере, три типа выходов МП.

25. Назовите, по меньшей мере, четыре типа входов МП.

26. Назовите, по меньшей мере, четыре входа ПЗУ.

27. Назовите, по меньшей мере, четыре типа входов ОЗУ.

28. Какие действия предпринимаются МП, когда линия прерываний активизируется клавишным устройством?

29. Какова роль дешифратора адреса?

30. Запись в ячейки памяти или считывание из них представляет собой _____ (доступ в, поиски) память.

31. Оперативное и постоянное запоминающие устройства, используемые в микроЭВМ, являются примерами памяти с ____ (произвольным, последовательным) доступом.

32. Где располагается резидентная память данных (РПД) микроконтроллера?

33. Центральный процессор обычно содержит:

а) устройство размещения данных, называемое _____;

б) устройство счета, называемое _____;

в) устройство _____;

г) устройство _____ и синхронизации.

34. Какая важная часть ЦП предназначена для управления всеми событиями внутри системы?

35. Регистр ЦП, удерживающий адрес последующей команды, извлекаемой из программной памяти, называется _____.

36. Обычно счетчик команд инкрементируется в прямом направлении для адресации команд в программной памяти в порядке возрастания адреса, за исключением случаев, когда его содержимое изменяется командами _____.

37. В начале процедуры выполнения команды КОП первой команды помещается в регистр (аккумулятора, команд) МП _____.

38. Часть МП, интерпретирующая КОП, помещенный в регистр команд, и определяющая последующую процедуру управления и синхронизации для выполнения команды, является ____.

39. Центральный микропроцессор микроЭВМ обычно содержит модификацию оперативной памяти, составляемой _____ (ОЗУ, регистрами).

40. Центральный процессор микроЭВМ содержит обычно устройство интерпретации команд, называемое _____.

Решения

1. Микропроцессор (обозначается как ЦП, CPU) контролирует все прочие устройства микроЭВМ (или управляет ими).

2. Адресная шина, шина данных и линий (шина) управления. В действительности линий может быть больше.

3. Данных.

4. ПЗУ.

5. ОЗУ.

6. Временно.

7. Постоянно.

8. Порта.

9. Программу.

10. Программ.

11. Операнда.

12. Последовательно.

13. Извлечение, декодирование, выполнение.

14. Устройства ввода, вывода, памяти, управления и арифметических действий.

15. Программные.

16. ОЗУ.

17. Постоянны.

18. Адреса.

19. Данных.

20. Памяти.

21. МП, однако работой микроЭВМ управляют команды, содержащиеся в ПЗУ и/или в ОЗУ.

22. Адреса.

23. 65536.

24. Выводы адресной шины, шины данных и шины управления.

25. Входы питания, ГТИ, прерываний и шины данных.

26. Входы питания, адресов (адресная шина), выбора кристалла и активизации чтения.

27. Входы питания, адресов (адресная шина), выбора кристалла, активизации чтения/записи (шина управления) и шина данных.

28. а) МП завершает выполнение текущей команды; б) МП прерывает свою последовательную работу; в) МП обращается к специальной группе команд, управляющих вводом данных с клавиатуры.

29. Он выбирает и активизирует единственное требуемое устройство (интерфейс с клавиатурой, ПЗУ, ОЗУ пли интерфейс индикатора).

30. Доступ в.
31. Произвольным.
32. На кристалле микроконтроллера.
33. а) регистром; б) АЛУ; в) декодирования команд; г) управления.
34. Блок управления и синхронизации.
35. Счётчик команд.
36. Ветвления, возврата, вызова прерываний.
37. Команд.
38. Дешифратором команд.
39. Регистрами.
40. Дешифратором команд.

ПРИЛОЖЕНИЕ

ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Запустите среду инструментальную среду разработки программ для микроконтроллеров AVR Studio x.x.

2. Создайте проект.

3. В диалоге 1 создания проекта укажите:

a) тип проекта: ассемблер (Project type: Atmel AVR Assembler);

b) имя проекта (Project Name);

c) флаг создания исходного файла (Create initial File);

d) флаг создания папки для проекта (Create Folder);

e) имя файла с исходным текстом (Initial File) (по умолчанию совпадает с именем проекта);

f) папку, в которой будет сохранён проект (Location).

4. В диалоге 2 создания проекта укажите:

a) отладочную платформу (Debug Platform): AVR Simulator;

b) устройство (Device): AT90S2313.

5. Создайте программу в редакторе и сохраните (File -> Save или Ctrl-S).

6. Скопируйте в папку проекта файл 2313def.inc.

7. Скомпилируйте программу (Project -> Build или F7).

8. Исправьте ошибки (если они есть, повторите п. 7).

9. Запустите программу на пошаговое выполнение (Debug -> Start Debugging).

10. Проверьте значения используемых регистров и портов ввода-вывода в окне View -> Workspace -> IO.

11. Если значение порта ввода-вывода требуется изменить, «кликните» в бит, значение которого надо изменить.

12. Для остановки процесса отладки выберите в меню соответствующий пункт (Debug -> Stop Debugging).

Примечание к п. 6. Вы можете хранить файл 2313def.inc в одном экземпляре в папке, общей для всех проектов (лабораторных работ), а при оформлении программы записывать директиву подключения так: .include "..\2313def.inc"

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Голубцов, М.С. Микроконтроллеры AVR: от простого к сложному / М.С. Голубцов, А.В. Кириченкова. – М.: СОЛОН-Пресс, 2004.

2. Громов, Г.А. Графический ассемблер / Г.А. Громов // URL: http://www.chipnews.ru, algrom@tula.net

3. Евстифеев, А.В. Микроконтроллеры AVR семейства Classic фирмы ATMEL / А.В. Евстифеев. – 2-е изд., стер. – М.: Издательский дом «Додека-XXI», 2004. – (Серия «Мировая электроника»).

4. URL: http://www.atmel.ru

5. URL: http://www.fulcrum.ru

6. URL: http://www.google.ru, конференция fido7.ru.embedded

СОДЕРЖАНИЕ