Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра радиотехнических систем

В. Н. Левкович, О. В. Шабров

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Учебно-методическое пособие
для студентов радиотехнических специальностей
всех форм обучения

Минск 2007

УДК 681.325.5-181.48(075.8)
ББК 32.973.26-04 я 73
Л 37

Р е ц е н з е н т:
доцент кафедры сетей и устройств телекоммуникаций БГУИР,
канд. техн. наук И. И. Астровский

Левкович, В. Н.

Л 37 Микропроцессорные устройства: учеб.-метод. пособие для студ. радиотехнич. спец. всех форм обуч. / В. Н. Левкович, О. В. Шабров. – Минск : БГУИР, 2007. – 91 с. : ил.

ISBN 978-985-488-190-4

В пособие включены теоретические сведения о принципах аппаратной организации, функционирования и программирования микропроцессорных устройств, а также способах представления и преобразования информации в этих устройствах.

УДК 681.325.5-181.48(075.8)
ББК 32.973.26-04 я 73

ISBN 978-985-488-190-4

ВВЕДЕНИЕ

Микропроцессоры – это логические устройства, оказывающие заметное влияние на нашу жизнь. Микропроцессоры можно обнаружить в карманных калькуляторах, кассовых аппаратах магазинов, бытовых и научных приборах, медицинском и офисном оборудовании, мобильных телефонах, детских игрушках – и это далеко не полный перечень их применения. Более того, каждый день находятся новые приложения и разрабатываются новые изделия на основе микропроцессоров. Можно констатировать, что в настоящее время практически ни одна разработка радиоэлектронного оборудования не проводится без применения микропроцессоров. Их потенциальное воздействие на нашу жизнь почти не поддается воображению.

С 1951 г., когда был создан первый производимый промышленностью компьютер (Univac I, США), электронные вычислительные машины (ЭВМ) оказывают сильное влияние на наше общество и образ жизни. Возникла новая отрасль промышленности. Такие термины, как «цифровые вычисления», «логическое проектирование» и «программирование», стали научными и инженерными понятиями. Однако широта этих понятий часто приводила к расхождению интересов. Например, интересы одной группы специалистов лежали в использовании ЭВМ и программировании (область программного обеспечения), другой – в создании ЭВМ (область аппаратуры). Хотя такое деление по интересам было, по-видимому, оправданным в отношении больших вычислительных машин, но с момента появления мини-компьютеров в 1965 г. проблемы, с которыми сталкивались прикладные программисты и конструкторы машин, стали переплетаться теснее. Мини-компьютеры уже не были всецело предназначены для обработки данных и решения вычислительных задач; зачастую они входили как составные части в системы, требовавшие быстрого принятия решения, – ***системы реального времени***.

С появлением в 1971 г. микропроцессоров расхождение интересов еще более уменьшилось. Началась эра программируемой логики. В эту эру понятия программирования и принципы проектирования логических схем сблизились настолько, что их взаимопроникновение потребовало и от ученых, и от инженеров полного понимания как принципов программирования, так и аппаратуры. Только при этом условии можно было успешно использовать все заложенные в микропроцессорах возможности.

Основной целью настоящей работы авторы видели сбалансированное изложение информации по аппаратной и логической организации, а также программированию микропроцессорных устройств.

1. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ И ФУНКЦИОНИРОВАНИЯ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

1.1. Основные понятия и определения

Компьютер – термин, используемый в англоязычной среде для наименования электронной вычислительной машины (ЭВМ).

ЭВМ – универсальное устройство для автоматической обработки информации, представленной в форме цифровых кодов.

Процессор – основной функциональный блок компьютера, непосредственно осуществляющий процесс обработки цифровых данных и программное управление этим процессом.

Микропроцессор – процессор, выполненный по технологии больших интегральных схем (БИС-технологии) на одном кристалле в одном корпусе.

Микрокомпьютер – компьютер, построенный на базе микропроцессора.

Микроконтроллер – микрокомпьютер, предназначенный для работы в качестве встроенного блока управления аппарата или прибора под управлением не изменяемой в процессе работы программы.

Однокристальный микроконтроллер – микроконтроллер, выполненный на одном кристалле в одном корпусе.

Микропроцессор является продуктом достижений технологии полупроводников, позволяющей создавать большие интегральные схемы (БИС) с числом транзисторов на одной кремниевой подложке (на одном «кристалле») в десятки и сотни тысяч. Это называют *высокой степенью интеграции.* Собственно говоря, высокая степень интеграции и привела к созданию микропроцессоров.

Как будет показано далее, в конструкцию микропроцессора заложена большая гибкость. Но сам по себе он не может решить ту или иную конкретную задачу. Чтобы решить задачу, его нужно соединить с другими устройствами и запрограммировать. В их число обычно входят память и устройства ввода/вывода. Некоторая совокупность соединенных друг с другом системных устройств, включающая микропроцессор, память и устройства ввода/вывода, нацеленная на выполнение некоторой четко определенной функции, в зависимости от ее сложности и называется ***микропроцессорным устройством,*** или ***микропроцессорной системой***.

Микрокомпьютеры обладают всеми свойствами обычных ЭВМ, однако замечательная их особенность состоит в относительно низкой стоимости и малых размерах. Именно этому они обязаны своей популярностью и успехом. Большие ЭВМ и мини-компьютеры обладают, конечно, большей

вычислительной мощностью, но не для всех приложений эта мощность оказывается необходимой. Более того, стоимость больших компьютеров и мини-компьютеров часто не позволяет включать их в системы, где они могли бы с успехом применяться. Микропроцессоры открывают возможность для применения программируемых устройств в тех логических системах, для которых фактор стоимости оказывается важнее, чем скорость и разнообразие вычислений.

1.2. Типовая структура микрокомпьютера (микропроцессорной системы)

Типовая компьютерная система включает шесть функциональных блоков: устройство ввода, память, арифметико-логическое устройство (АЛУ), устройство управления (УУ), устройство вывода, генератор тактовых импульсов (ГТИ). Пример такой системы приведен на рис. 1.1.

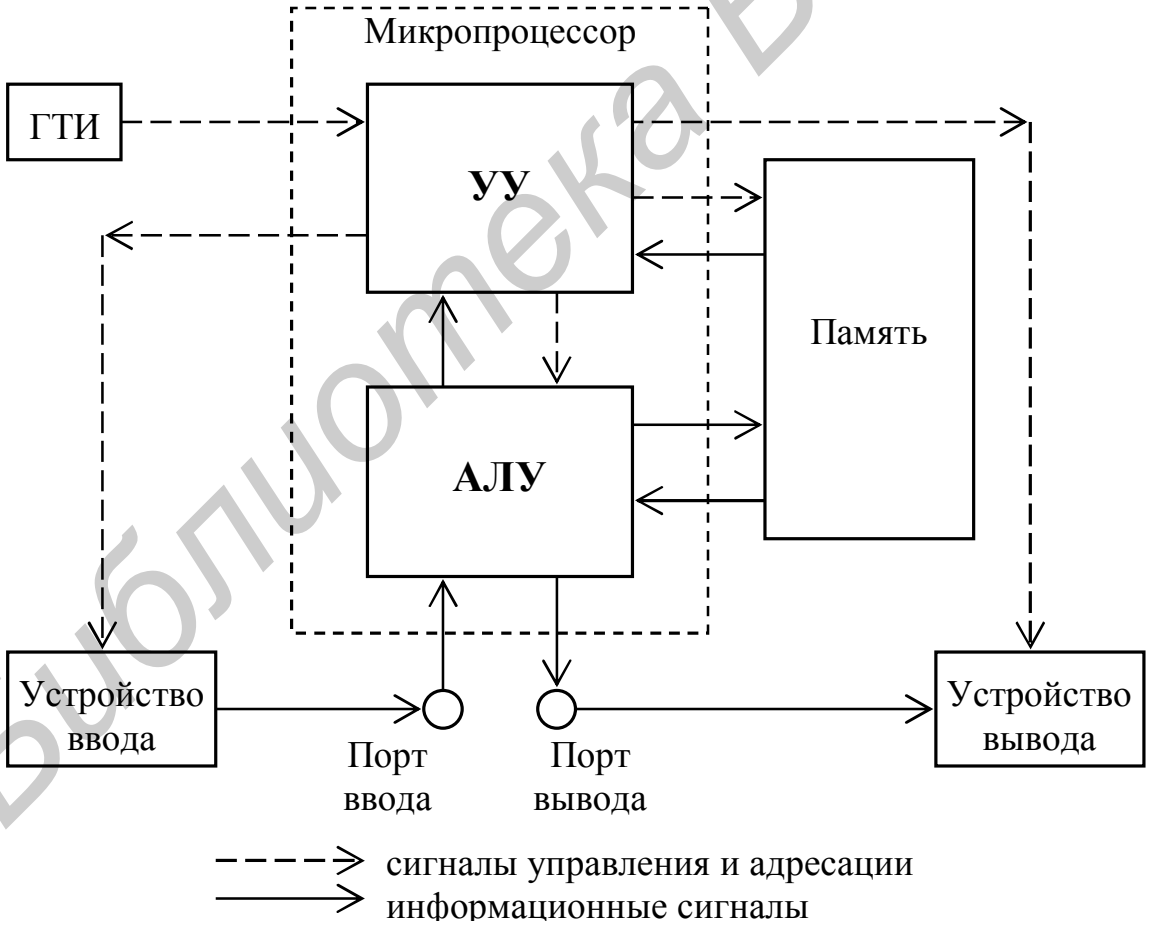


Рис. 1.1. Типовая структура микрокомпьютера

Физические компоненты и схемы, составляющие микрокомпьютер, – это его ***аппаратура*** (hardware). Аппаратура способна выполнять только

ограниченный набор элементарных операций. Аппаратные возможности микроконтроллера описываются его системой машинных команд. Для различных процессоров количество команд различно и изменяется от нескольких десятков до сотен. Все команды, которые способен выполнять микроконтроллер, примитивны. Все прочие более сложные функциональные возможности микрокомпьютера достигаются ***программным путем*** и строятся на основе более простых команд**.**

Программа – это определенным образом организованная совокупность элементарных машинных операций, называемых ***командами*** или ***инструкциями***, с помощью которых осуществляется обработка информации, или данных. Программы, написанные для компьютера, образуют его ***программное обеспечение*** (software).

Программа и данные сначала накапливаются в памяти (П), куда они поступают через устройство ввода. Затем отдельные команды программы одна за другой автоматически поступают в устройство управления (УУ), которое их расшифровывает и выполняет. Для выполнения операции обычно требуется, чтобы данные поступили в арифметико-логическое устройство (АЛУ), содержащее все необходимые для их обработки схемы. Только в АЛУ происходит изменение или преобразование информации, во всех остальных частях микрокомпьютера данные просто передаются без изменений. В процессе вычислений или после их завершения полученные результаты направляются в устройство вывода. АЛУ и УУ вместе обычно называются ***центральным процессорным устройством*** (ЦПУ), или ***центральным процессором*** (ЦП). Центральный процессор в микрокомпьютерной системе – это и есть микропроцессор. Микропроцессор совместно с памятью образуют ***ядро*** микрокомпьютера.

Не только память, но и другие устройства ЭВМ способны хранить информацию. Информация запоминается как содержимое групп двоичных разрядов – ***битов*** – на запоминающих устройствах – ***регистрах.*** По существу любую операцию в ЭВМ можно рассматривать как серию передач информации между регистрами с возможным ее преобразованием (например сложением), выполняемым в процессе передач. Группа двоичных цифр, обрабатываемых одновременно, называется ***машинным словом,*** а число двоичных цифр, составляющих слово, называется ***длиной слова.*** Слово является базовой логической единицей информации в компьютере. Команды или данные обычно состоят из одного или нескольких слов. Типичные микропроцессоры имеют длину слова 4, 8, 12, 16, 32, 64 или 128 двоичных разрядов. В силу особой распространенности слово длиной 8 битов имеет специальное название – ***байт.***

1.3. Память

Запоминание больших объемов информации происходит в памяти или, точнее, в запоминающем устройстве (ЗУ). Этот функциональный блок компьютера подразделяется на подблоки, называемые ***регистрами***, каждый из которых способен хранить одно машинное слово. Каждый такой регистр, или ячейка памяти, имеет свой адрес. ***Адрес*** – это просто целое число, однозначно идентифицирующее ячейку. Слово, хранящееся в ячейке, называют ***содержимым*** этой ячейки.

Итак, как данные, так и программа хранятся в памяти. Это важное обстоятельство приводит к двум основным концепциям проектирования компьютеров. Первая заключается в том, что компьютер имеет два отдельных и четко различающихся вида памяти. Программа находится всегда в одной памяти, а данные – в другой, что позволяет одновременно обращаться и к памяти программы, и памяти данных. Эта возможность значительно повышает быстродействие системы. Машины, спроектированные в соответствии с концепцией разделения памяти на два вида, называют машинами ***гарвардского*** типа.

В соответствии со второй концепцией различие между программной памятью и памятью для данных не проводится. Такие компьютеры называют машинами ***фон-неймановского,*** или ***принстонского*** типа. В них программа может размещаться в любом месте общей памяти, и задача программиста – следить за тем, чтобы данные и программа обрабатывались по-разному. Преимущество второй концепции – в возможности трактовать программу как данные, что позволяет компьютеру изменять свои собственные программы. Существуют микропроцессоры, спроектированные в соответствии как с первой, так и со второй концепцией.

Благодаря низкой стоимости микропроцессоры часто предназначаются для решения одной конкретной задачи. Большие универсальные ЭВМ постоянно перепрограммируются и поэтому могут решать задачи широкого спектра. Микропроцессорам, специализированным для одного конкретного приложения, такая гибкость не нужна. Однажды написанная и отлаженная программа в дальнейшем обычно не изменяется. Поэтому такие микрокомпьютеры часто имеют два вида памяти: память, из которой возможно только считывание (ROM – read only memory), или постоянная память, и память со считыванием и записью (RWM – read/write memory). Изменить информацию, однажды записанную в постоянную память, сложно. Память этого типа благодаря своей низкой стоимости используется для хранения программ и

постоянных данных; изменяющаяся информация хранится в памяти со считыванием и записью.

Память со считыванием и записью принято называть памятью с произвольной выборкой (RAM – random-access memory), несмотря на то что и память только со считыванием также обладает произвольностью выборки. Термин «произвольная выборка», или «произвольный доступ», соответствует тому факту, что обращение к любой ячейке выполняется за одно и то же время.

В качестве русских эквивалентов сокращениям RAM, RWM и ROM часто используются: ЗУПВ – запоминающее устройство с произвольной выборкой, ОЗУ – оперативное запоминающее устройство и ПЗУ – постоянное запоминающее устройство.

Следует обратить внимание на то, что из наличия постоянной памяти не следует, что машина относится к гарвардскому типу. В машине фон-неймановского типа ячейки ROM (только для считывания) могут появляться в памяти где угодно. Главное то, что микропроцессору «безразлично», откуда он получает информацию – из ROM или RAM.

1.4. Арифметико-логическое устройство

Обработка данных осуществляется главным образом в арифметико-логическом устройстве. Эта обработка включает как арифметические, так и логические операции. Встроенные операции чрезвычайно элементарны. Полный перечень возможных встроенных операций для любого процессора задается его системой команд. Более сложные математические действия должны выполняться с помощью программ, пользующихся встроенными операциями.

В качестве модели для пояснения принципа работы АЛУ рассмотрим схему, показанную на рис. 1.2.

На вход АЛУ поступают данные **A** и **B,** над которыми необходимо произвести заданное действие. Над этими данными одновременно производятся все возможные (перечисленные в системе команд) логические и арифметические операции, результаты которых поступают на вход коммутатора (К). На коммутатор также подается код операции (КОП) команды, выполняющейся в данный момент, который указывает, какой из результатов должен быть передан на выход АЛУ.

Обычно в состав АЛУ входит ряд регистров, предназначенных для хранения операндов на время их обработки. Главный регистр в АЛУ называется ***аккумулятором***. В нем, как правило, находится один из операндов перед выполнением операции, и в него же помещается ее результат.

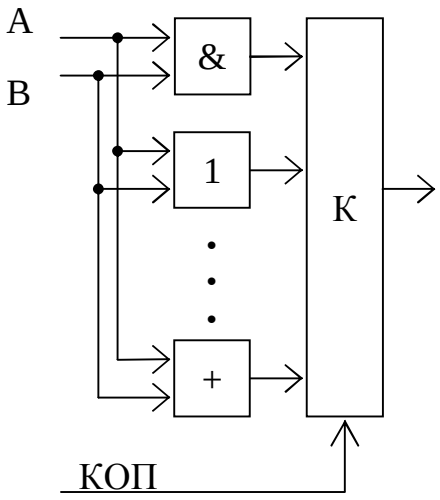


Рис. 1.2. Принципиальная организация АЛУ

В состав АЛУ входит также регистр флагов. ***Флаги*** или ***флажки*** – это просто биты, содержащие информацию, характеризующую состояние микропроцессора, которое важно для выбора дальнейшего пути вычислений. Например, может существовать флажок, указывающий на нулевой результат операции. Программист может воспользоваться проверкой этого флажка для принятия решения: если некоторая операция дала нулевой результат, то будет выполнена одна последовательность команд, в противном случае – другая. Наиболее часто используются три флага: **C** – переполнения, **Z** – нулевого результата, **DC** – десятичного переноса.

Флажковые биты, характеризующие результаты операций или каких-либо проверок, часто размещаются вместе с другой важной информацией о состоянии машины в специальном регистре, называемом ***словом состояния программы*** (PSW – program status word).

1.5. Устройство управления

Устройство управления управляет работой компьютера. Оно автоматически, последовательно по одной выбирает команды из памяти, декодирует каждую из них и генерирует необходимые для ее выполнения сигналы. Для того чтобы получить команду из памяти, устройство управления прежде всего должно знать ее адрес. Обычно команды выбираются из последовательных ячеек памяти, и их адреса указываются ***программным счетчиком*** (ПС), находящимся в устройстве управления.

В момент включения питания автоматически формируется сигнал СБРОС, которым обнуляется программный счетчик, и это обеспечивает выборку команды из нулевой ячейки памяти. После извлечения первой

команды к счетчику добавляется единица, и он указывает уже на следующую ячейку с командой и т.д. При необходимости можно счетчику в нужный момент по отдельному каналу навязать адрес, который будет указывать не на следующую ячейку, а какую-то иную. Благодаря этому можно организовать ветвление в программе.

Далее, чтобы иметь возможность декодировать и выполнить текущую команду, ее нужно где-то запомнить. Этой цели в устройстве управления служит ***регистр команды***.

Для того чтобы быть правильно проинтерпретированной устройством управления, команда должна иметь определенную ***структуру***, т.е. содержать определенные логические части, называемые полями, каждая из которых несет определенную информацию. Взаимное расположение полей в команде с указанием длин полей в битах называют ***форматом команды***. У микропроцессоров разных типов форматы команд различны. Однако есть информация, которая должна присутствовать в команде в любом случае. Наиболее важное значение имеет ***код операции*** (КОП) и в некоторых командах адрес. ***Код операции*** – это совокупность двоичных цифр, которые однозначно определяют операцию, выполняемую в процессе интерпретации команды. ***Адресная часть*** команды (если она присутствует) указывает на ячейки (например в памяти), к которым нужно обратиться, выполняя команду. Например, если выполняется операция сложения, адресная часть команды может указывать на ячейку, где находится второе слагаемое. Но адресный код не обязателен – в некоторых командах он может и отсутствовать.

Структура команд первых процессоров была более громоздкой (рис. 1.3). В команде указывались адреса обоих операндов, адрес, куда записывался результат и адрес следующей команды. Для извлечения такой команды приходилось несколько раз обращаться к памяти, что естественно приводило к значительному снижению быстродействия. За счет введения программного счетчика избавились от части, в которой указывался адрес следующей команды. Так как один из операндов, как правило, после выполнения некоторого действия не нужен, то стали записывать результат на его место, что позволило исключить еще одно поле. С введением аккумулятора в АЛУ приняли, что один из операндов находится в нем, и отпала необходимость указывать адрес этого операнда. Таким образом, в процессе эволюции компьютеров сократили длины команд и за счет этого увеличили их быстродействие.

Важно различать понятия «адрес ячейки памяти» и «содержимое ячейки памяти».

В командах определенного формата существует адресная часть. Это числовой указатель ячейки, связанной с операндом. Однако команда сама находится в памяти и, следовательно, имеет связанный с ней адрес. Как правило, этот адрес не совпадает с адресной частью в самой команде.

КОП	Адрес первого операнда	Адрес второго операнда	Адрес результата	Адрес следующей команды

Рис. 1.3. Структура команд первых процессоров

Следующая функция устройства управления – это синхронизация работы отдельных блоков компьютера. Она осуществляется с помощью ***генератора тактовых импульсов*** (ГТИ), или ***тактового генератора***. Обработка команды занимает несколько периодов тактового генератора. Вообще говоря, выполнение команды можно разделить на три фазы (машинных цикла): *извлечение кода операции; первичная дешифрация кода команды для определения длины ее адресной части и извлечение из памяти адресной части команды; вторичная дешифрация для определения операции и выполнение команды.*

Выполнение каждого из названных циклов требует нескольких периодов тактового генератора (машинных тактов) (рис. 1.4). Совокупное время, требуемое для выборки, декодирования и выполнения команды, образует ***командный цикл,*** или ***цикл выполнения команды.***

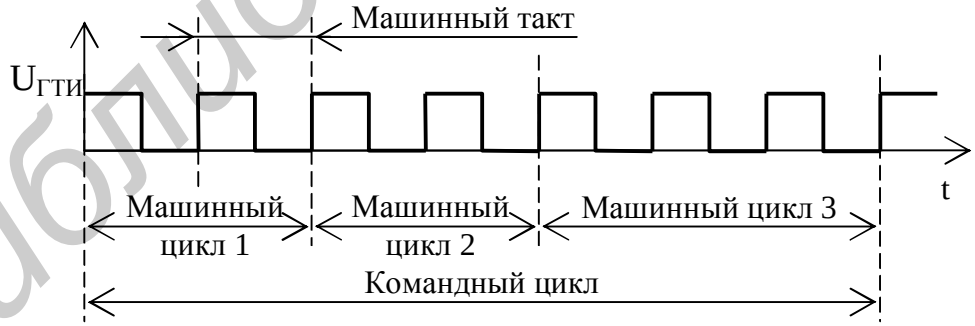


Рис. 1.4. Циклы работы микропроцессора

1.6. Устройства ввода/вывода (периферийные устройства)

Последние два блока машины – это устройство ввода и устройство вывода. Через эти устройства осуществляется контакт компьютера с внешним миром. Они являются буферами для преобразования информации с тех языков

и тех скоростей, на которых работает компьютер, к тем, которые воспринимает человек или другая связанная с компьютером система. Устройство ввода получает из внешнего мира данные и команды, которые поступают в память. Устройство вывода получает вычисленные результаты и передает их человеку-оператору или другой системе.

Устройства ввода и вывода представляют собой ***периферийные устройства*** машины. В качестве примеров можно назвать цифровое табло или клавиатуру. Точки контакта между устройствами ввода/вывода и микропроцессором называются ***портами ввода/вывода***. Порты ввода/вывода – это просто параллельные или последовательные регистры, которые также имеют свои адреса, так что к одному микропроцессору может быть подключено несколько устройств ввода/вывода.

Характерная особенность цифрового компьютера состоит в том, что вся информация хранится и обрабатывается в дискретном виде, т. е. в виде конечных чисел. Часто возникает необходимость сопряжения компьютера с другой системой, не способной обрабатывать дискретную информацию. Недискретная информация называется ***аналоговой*** или ***непрерывной***. В таких случаях приходится осуществлять преобразование из цифровой формы в непрерывную и обратно. Устройства ввода/вывода, осуществляющие соответствующие преобразования, называются ***аналого-цифровыми*** и ***цифро-аналоговыми преобразователями***.

На рис. 1.1 устройства ввода/вывода показаны подключенными к арифметическому устройству. Это не единственно возможный способ подключения. Часто для достижения большей производительности системы желательно позволить устройствам ввода/вывода обращаться к памяти непосредственно, а не через арифметическое устройство. В таком случае говорят о ***прямом доступе к памяти*** (DMA – direct memory access). Организация прямого доступа к памяти обеспечивается в компьютерах с помощью специальных функциональных блоков, называемых контроллерами прямого доступа.

1.7. Шины

Отдельные блоки микрокомпьютера связаны друг с другом с помощью шин. Шина представляет собой совокупность линий, по которым передается информация от любого из нескольких источников к любому из нескольких приемников. Одна из распространенных в микрокомпьютерах структур шин представлена на рис. 1.5. На рисунке показаны шины трех типов.

Шина адреса (ША) – однонаправленная, т.е. информация по ней передается только в одном направлении. Эта шина служит для передачи адреса от микропроцессора к памяти, вводному или выводному устройству.

Шина данных (ШД) – двунаправленная, т. е. информация по ней может передаваться в обоих направлениях, и она служит для передачи данных.

Наконец, ***шина управления (ШУ)*** состоит из линий, по которым передаются тактовые, синхронизирующие сигналы, а также информация о состоянии (статусе) устройств. По шине управления могут передаваться следующие сигналы: «запись», «чтение», «ввод», «вывод» и «готовность». Часть линий в управляющей шине – однонаправленные, часть – двунаправленные. Поэтому на рисунке направленность этой шины никак не обозначена.

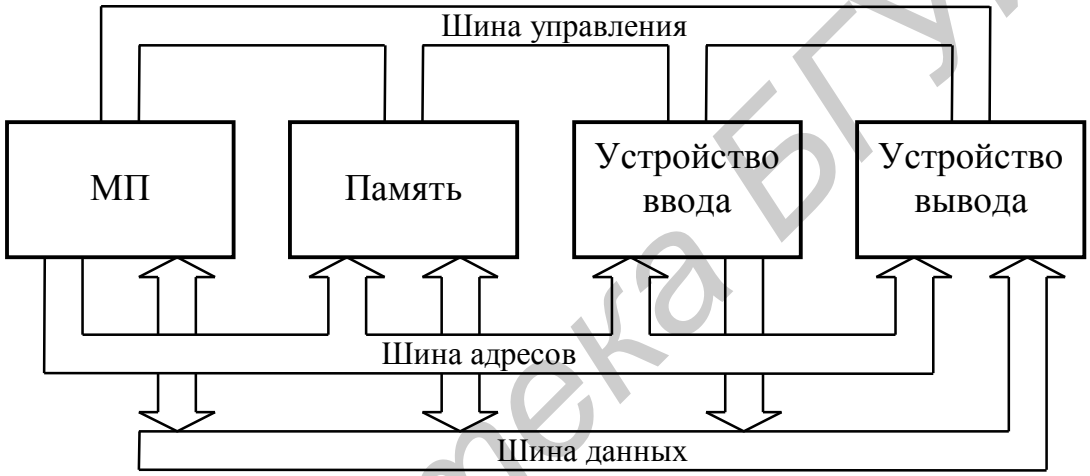


Рис. 1.5. Шины типичного микрокомпьютера

На рис. 1.6. показаны временные диаграммы передачи слова данных из процессора в ячейку памяти, а на рис. 1.7 – в обратном направлении.

1.8. Стек

Как уже говорилось, при помощи команд переходов можно отходить от строго последовательного порядка выполнения команд. Особый тип команды перехода, несколько отличающийся от рассмотренных выше, позволяет многократно использовать некоторую специально выделенную группу команд, называемую ***подпрограммой***.

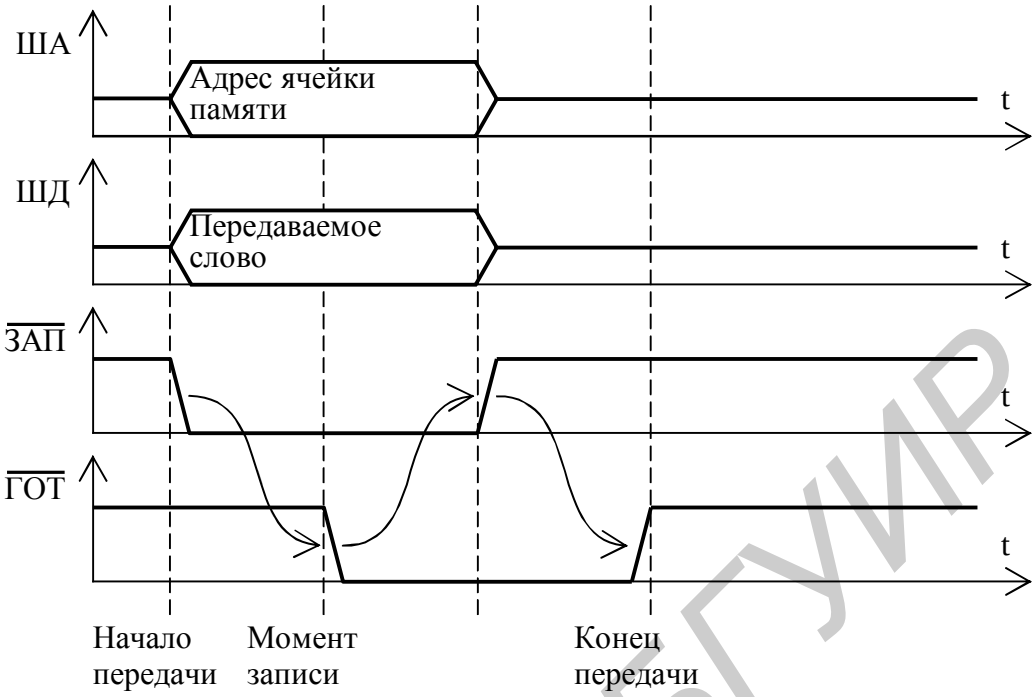


Рис. 1.6. Диаграммы передачи слова из процессора в память

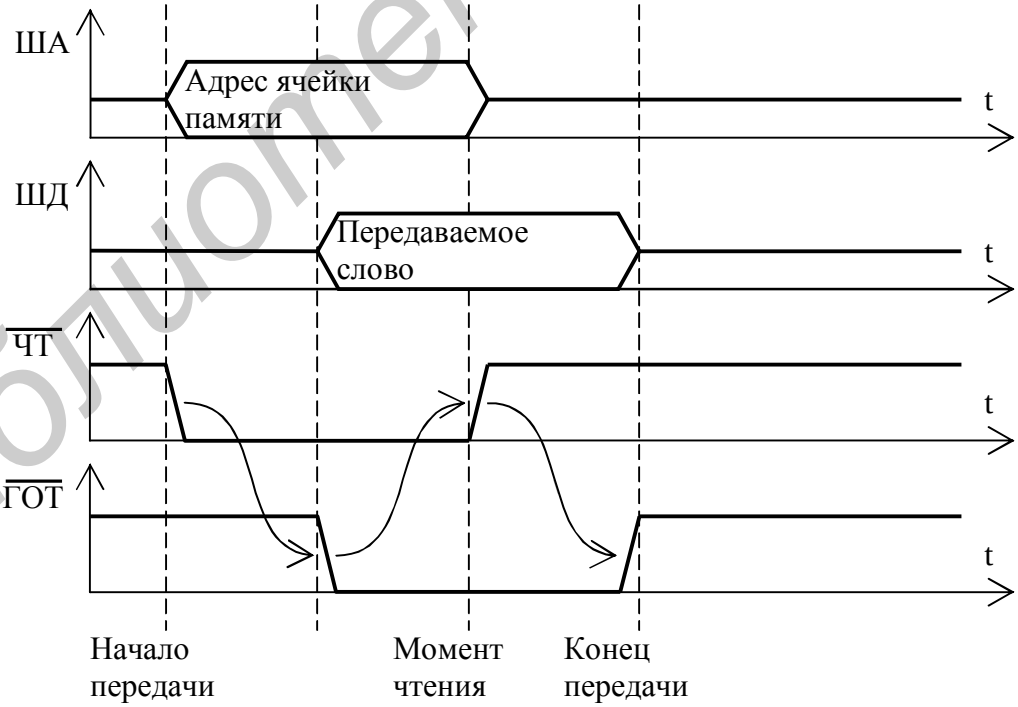


Рис. 1.7. Диаграммы передачи слова из памяти в процессор

В процессе выполняемых вычислений в главной программе может потребоваться выполнить некоторое «подвычисление», оформленное в виде подпрограммы. Компьютер должен передать управление из главной программы на подпрограмму, выполнить соответствующее «подвычисление» и затем осуществить возврат в главную программу для продолжения работы. Чтобы осуществить такой возврат, необходимо сохранить значение программного счетчика, которое было до перехода к подпрограмме. Это и делается в команде ***переход на подпрограмму*** **(*CALL*)**, а именно: запоминается содержимое программного счетчика и затем передается управление. Для того чтобы восстановить программный счетчик, используя запомненное значение, и обеспечить возврат к главной программе, употребляется специальная команда ***возврат из подпрограммы*** **(*RETURN*)**. Место, где запоминается содержимое программного счетчика, обычно представляет собой ***стек.***

В общем случае стек – это совокупность регистров, которые принимают и выдают информацию в соответствии с правилом «последним вошел, первым вышел» (LIFO – last-in first-out). Это означает, что только вершина стека, где находится последний его элемент, непосредственно доступна извне. В зависимости от конструкции компьютера стек может находиться в устройстве управления или быть частью памяти. Принцип работы стека отражен на рис. 1.8.

Если стек может хранить несколько адресов (значений программного счетчика), имеется возможность выполнять вложенные подпрограммы. Таким образом, из одной подпрограммы можно сделать переход на другую подпрограмму или даже на саму себя – это называется ***рекурсией***. Очевидно, что глубина вложений подпрограмм зависит от емкости стека. Если емкость стека будет полностью заполнена, а в него запишется еще одно значение, то значение, записанное первым, будет потеряно.

1.9. Прерывание программы

Во многих приложениях микропроцессоров возникает необходимость прерывать процесс вычислений для обслуживания внешнего устройства. Прерывание может происходить при поступлении в процессор сигнала от внешнего устройства, требующего внимания. В ответ на это процессор должен приостановить выполнение текущей программы, запомнить состояние, в котором она была прервана (содержимое различных регистров, включая программный счетчик), и затем обслужить запрос внешнего устройства. Завершив обслуживание, микропроцессор возвращается к приостановленному

процессу вычислений, воспользовавшись ранее запомненной информацией о состоянии прерванной программы.

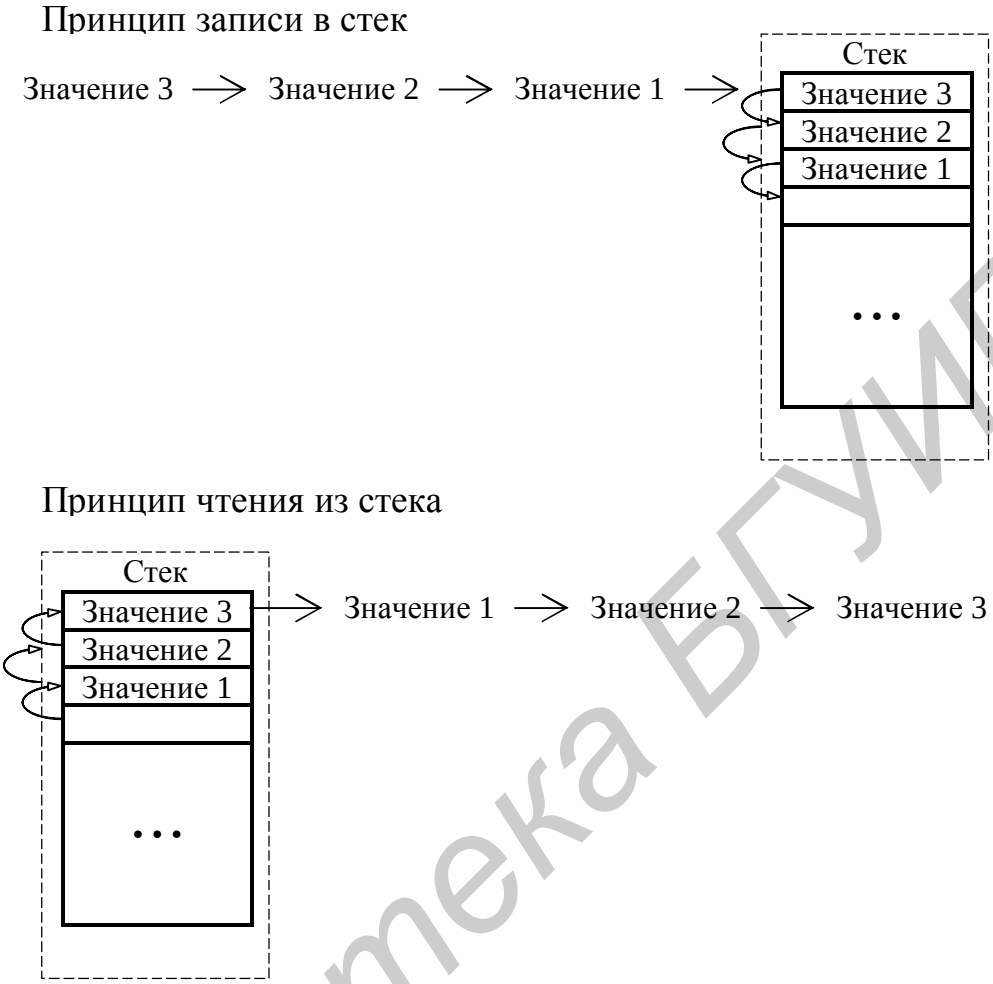


Рис. 1.8. Работа стека в режиме чтения/записи

Понятие прерывания можно обобщить для случая нескольких внешних устройств. В этом случае любое из устройств может прислать свой запрос на внимание. Обработка запросов может либо следовать правилу обслуживания «первым вошел – первым вышел», либо некоторой приоритетной схеме в соответствии с приоритетом запросов.

При всех условиях способность обрабатывать прерывания может оказаться очень мощным средством во многих применениях микропроцессоров. Поскольку при прерываниях нужно сохранять информацию о текущем состоянии процессора, можно опять воспользоваться стеком, т.е. сохранить в нем внутреннее состояние микропроцессора в момент поступления запроса, включая программный счетчик. В некотором смысле прерывание подобно

переходу на подпрограмму, но этот переход проинициирован внешним сигналом, а не командой в программе.

1.10. Функционирование микрокомпьютера

В предыдущем разделе были рассмотрены функции отдельных блоков микрокомпьютера. Здесь мы остановимся на взаимодействии блоков и на динамике информационных потоков.

Устройство управления в процессе функционирования проходит через три фазы: выборка, декодирование и выполнение. После того как программа и данные поступили в память, адрес первой выполняемой команды помещается в программный счетчик, и в устройстве управления устанавливается фаза выборки. При этом содержимое программного счетчика поступает на адресную шину, и тем самым обеспечивается возможность выборки соответствующей команды из памяти.

Команда, хранящаяся в ячейке с адресом, заданным на программном счетчике, посылается по шинам данных в регистр команды в устройстве управления. Поскольку команды в памяти располагаются в последовательных ячейках, программный счетчик увеличивается на 1, и на нем появляется адрес следующего слова в программе. Затем устройство управления декодирует код операции только что полученной команды. Если код операции показывает, что команда состоит более чем из одного слова, фаза выборки повторяется нужное число раз, чтобы выбрать команду целиком. При этом каждый раз увеличивается содержимое программного счетчика.

После выборки и декодирования всей команды устройство управления переходит в фазу выполнения. Оно генерирует управляющие сигналы, и соответствующие схемы выполняют заданную в команде операцию. Если в команде задан адрес операнда, устройство управления переходит к пересылке адресуемой информации между указанной ячейкой и соответствующим блоком машины, например арифметическим устройством или устройством вывода.

Для осуществления пересылки адресная часть команды передается на адресную шину, подготавливая последующее появление адресуемой информации на шине данных. В конечном счете, устройство управления обеспечивает фактическое выполнение заданной операции и после ее завершения снова возвращается к фазе выборки, чтобы получить из памяти следующую команду, адрес которой содержит программный счетчик. Этот процесс повторяется до тех пор, пока компьютер не получит указание остановиться.

В общем описанном выше случае команды выполняются последовательно в порядке их расположения в памяти. Однако в некоторых случаях этот порядок желательно изменить, например, в зависимости от состояния флажковых битов. В этом случае адрес следующей команды может не быть адресом ячейки, следующей по порядку за ячейкой, откуда была взята текущая выполняемая команда. Команды, в которых происходят такие изменения порядка выборки команд, называются ***командами переходов, передач управления*** или ***ветвлений***.

В этих командах адресная часть содержит адрес следующей команды, которую должно выбрать устройство управления, если последовательность выборки изменяется. Поэтому после того как устройство управления декодирует команду перехода и установит, что условия изменения порядка выполнены, оно поместит адресную часть выполняемой команды, содержащую адрес следующей команды, в программный счетчик. Таким образом, после того как устройство управления перейдет в фазу выборки, оно автоматически получит нужную ему следующую команду.

2. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВАХ

2.1. Системы счисления, применяющиеся в микропроцессорных устройствах

Система счисления – *совокупность символов и правил для обозначения чисел.* Все системы счисления разделяются на позиционные и непозиционные. Наиболее древними системами счисления являются непозиционные (в настоящее время они почти полностью заменены позиционными).

Для этих систем числа представляются в виде последовательности цифр $X_k = X_1, X_2 \mathbf{K} X_k \mathbf{K} X_n$, в которой значение каждой цифры X_k зависит от места ее расположения в последовательности. Пример непозиционной системы счисления – римская система счисления.

2.1.1. Позиционные системы счисления

Любое число в позиционной системе счисления с постоянным основанием можно представить в виде следующего полинома:

$$\mathrm{X} = \pm \sum_{k=1}^{n} \mathrm{X}_k q^{p-k},$$

где q – основание системы счисления; X_k – цифра k-го разряда числа; n – количество разрядов в числе; p – порядок числа (целое число, показывающее место запятой в числе).

Пример:

Число 128, для которого $q = 10$ и $p = 3$, получаем $128 = 1 \cdot 10^2 + 2 \cdot 10^1 + 8 \cdot 10^0 = 100 + 20 + 8.$

Двоичная система счисления (ДвСС)

В этой системе счисления для представления любого разряда двоичного числа достаточно иметь один физический элемент только с двумя резко различимыми состояниями, одно из которых изображает 1, а другое 0.

Достоинства ДвСС:

а) простота выполнения арифметических и логических операций и, как следствие, простота устройств, реализующих эти операции;

б) возможность использования аппарата алгебры логики для анализа и синтеза операционных устройств.

Недостатки ДвСС:

а) громоздкость по сравнению с десятичной для использования человеком;

б) необходимость преобразовывать десятичные числа в двоичные и наоборот.

Восьмеричная и шестнадцатеричная системы счисления

Такие системы относятся к двоично-кодированным системам, когда основание системы счисления представляют целые степени двойки: 2^3 для 8-ричной и 2^4 для 16-ричной системы счисления.

Изображение целых чисел в 8-ричной и 16-ричной системах счисления вместе с их двоичными и десятичными эквивалентами представлены в табл. 2.1.

В этой таблице для $q \leq 10$ учитывалось использование десятичных цифр для изображения цифровых символов, а для $q > 10$ – использование кроме десятичных цифр первых букв латинского алфавита.

Большим достоинством 8- и 16-ричной системы счисления является:

- возможность более компактно представить запись двоичного числа, а именно, запись одного и того же двоичного числа в 8- и 16-ричной системах будет соответственно в 3 и 4 раза короче двоичной;
- сравнительно просто осуществляется преобразование чисел из двоичной в 8- и 16-ричную системы и наоборот.

Действительно, так как для 8-ричного числа каждый разряд представляется группой из трех двоичных разрядов (*триад*), а для 16-ричного – группой из четырех двоичных разрядов (*тетрад*), то для такого преобразования достаточно объединить двоичные цифры в группы по 3 и 4 бита соответственно, продвигаясь от разделительной запятой вправо и влево. При этом в случае необходимости добавляют нули в начале и в конце числа и каждую такую группу – триаду или тетраду – заменяют эквивалентной 8- или 16-ричной цифрой.

Таблица 2.1

Система счисления	Десятичная	8-ричная	16-ричная	Двоично-десятичная
ЧИСЛОВЫЕ ЭКВИВАЛЕНТЫ	0	0	0	0
	1	1	1	1
	2	2	2	10
	3	3	3	11
	4	4	4	100
	5	5	5	101
	6	6	6	110
	7	7	7	111
	8	10	8	1000
	9	11	9	1001
	10	12	A	00010000
	11	13	B	00010001
	12	14	C	00010010
	13	15	D	00010011
	14	16	E	00010100
	15	17	F	00010101
	16	20	10	00010110
	17	21	11	00010111
	18	22	12	00011000
	19	23	13	00011001
	20	24	14	00100000
	21	25	15	00100001
	22	26	16	00100010
	23	27	17	00100011
	24	30	18	00100100
	25	31	19	00100101
	26	32	1A	00100110
	27	33	1B	00100111
	28	34	1C	00101000

Двоично-кодированная десятичная система счисления

Представляя каждую десятичную цифру совокупностью из четырех разрядов (*тетрад*), можно получить комбинированную систему счисления,

которая обладает достоинством двоичной системы и удобством десятичной. В ЭВМ наибольшее применение нашли системы кодирования *8421*, *2421* и *8421 + 3*.

Код *8421* называется кодом с ***естественными весами***, здесь цифры *8*, *4*, *2*, *1* – веса двоичных разрядов тетрад. Любая десятичная цифра в этом коде изображается ее эквивалентом в двоичной системе счисления. Этот код нашел наибольшее применение при кодировании десятичных чисел в устройствах ввода-вывода.

Особенность кодов *2421* и *8421 + 3* состоит в том, что кодирование любой десятичной цифры и дополнительной к ней цифры до 9 осуществляется взаимно дополняющимися тетрадами. Эта особенность дает простой способ получения дополнения до 9 путем инвертирования двоичных цифр тетрады. Изображение десятичной системы счисления в этих видах кодов приводится в табл. 2.2.

Таблица 2.2

Десятичная система счисления	Вид кодировки		
	8421	2421	8421+3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	1001	0110
4	0100	1010	0111
5	0101	0101	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

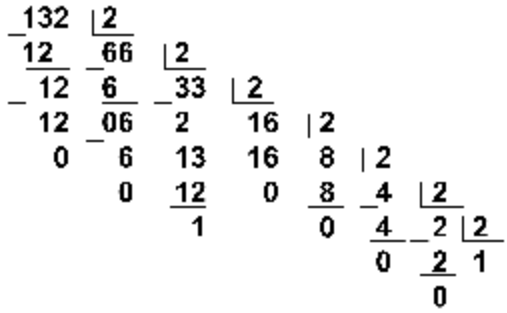
2.2.2. Перевод чисел из одной системы счисления в другую

Перевод целых чисел

Правило перевода целых чисел. Для перевода целого числа N_p, представленного в системе счисления с основанием p, в систему счисления с основанием q необходимо данное число делить на основание q (по правилам в системе счисления с основанием p) до получения целого остатка, меньшего q. Полученное частное снова необходимо делить на основание q до получения целого остатка, меньшего q и т.д. до тех пор, пока последнее частное будет меньше q. Число N_q в системе счисления с основанием q представляется в виде

упорядочений последовательности остатков от деления в порядке, обратном их получению. Причем старшую цифру числа N_q дает последнее частное.

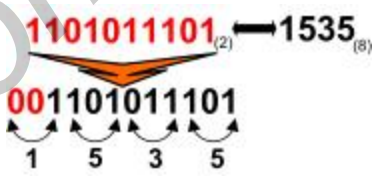
Пример: перевести число 132 из десятичной системы счисления в двоичную.



Десятичному числу $N_{10} = 132$ соответствует двоичное $N_2 = 10000100$.

Перевод чисел из двоичной системы в 8- и 16-ричную системы исчисления и обратно

Для перевода двоичного числа в 8-ричную систему счисления необходимо исходное число, начиная с правого младшего разряда, разбить на триады. Недостающие разряды заменить нулями, а затем каждую триаду записать в виде эквивалентного 8-ричного числа. При обратном переводе каждое 8-ричное число переводится обратно в триады. Для 16-ричной системы производятся аналогичные действия, только вместо триад деление ведется по тетрадам.



Перевод дробей

Правило перевода правильной дроби. Перевод правильной дроби N_p, представленной в системе счисления с основанием p, в систему счисления с основанием q заключается в последовательном умножении этой дроби на основание q (по правилам в системе счисления с основанием p), причем перемножению подвергаются только дробные части. Дробь N_q в системе счисления с основанием q представляется в виде упорядоченной последовательности целых частей произведений в порядке их получения, где старший разряд является первой цифрой произведения. Если требуемая точность перевода есть q^{-k}, то число указанных последовательных произведений равно k.

0,7689
× 2
1,5378
× 2
1,0756
× 2
0,1512
× 2
0,3024
× 2
0,6048
× 2
1,2096

Пример: перевести десятичную дробь *0,7689* в двоичную с точностью 2^{-6}.

Десятичной дроби N_{10} = 0,7689 соответствует двоичная N_2 = = 0,0110001 с точностью 2^{-6}.

Правило перевода неправильной дроби. Для чисел, имеющих как целую, так и дробную часть, перевод из одной системы счисления в другую осуществляется отдельно для целой и дробной части по правилам, указанным выше.

Существуют и другие алгоритмы перевода целых и дробных чисел из одной системы счисления в другую, основанные на представлении исходных чисел в виде многочленов.

Пример:

2.2. Формы представления чисел в микропроцессорных устройствах

Микропроцессор оперирует целыми и действительными числами. В свою очередь числа могут быть представлены с фиксированной и плавающей запятой.

Для всех представляемых действительных чисел их знаки кодируются цифрами: «плюс» кодируется «0», «минус» соответственно «1». При этом код знака всегда записывается перед старшим разрядом числа (рис.2.1).

Рис. 2.1. Представление десятичного числа со знаком в микропроцессорных устройствах

2.2.1. Представление чисел в форме с фиксированной запятой

В случаях фиксированной запятой (ФЗ) для всех чисел, над которыми выполняются операции в машине, положение запятой строго фиксировано, т.е. порядок числа p в машине постоянен. Выбор величины порядка p при использовании вида с ФЗ может быть принципе произволен. При этом предполагают следующее:

1. Положение запятой закрепляется в определенном месте относительно разрядов числа и сохраняется неизменным для всех чисел изображаемых в принятой разрядной сетке машины.

2. Хотя запятую и фиксируют, но это никак не выделяется в коде числа, а только подразумевается.

Например, число –10,01 может быть записано разными способами (рис. 2.2).

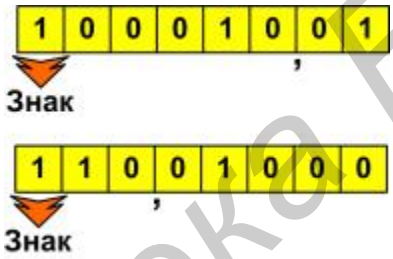


Рис. 2.2. Представления числа с фиксированной запятой

Как видно, место запятой может быть различно, причем на рисунке оно лишь символически показано, реально же в коде никаких указаний на нее нет. Мало того, её положение может меняться, поэтому программисту необходимо всегда следить и помнить, где находится запятая.

На практике в микропроцессорах используют два способа расположения запятой:

1) перед старшим разрядом ($p = 0$);

2) после младшего разряда ($p = n$, где n – число разрядов, необходимых для представления числа)

Виды расположения запятой в общем случае приведены на рис. 2.3.

В микропроцессорных системах для чисел с ФЗ их длина принимается равной длине машинного слова, или машинных слов, или части машинного слова.

Введём две характеристики чисел: *диапазон* изменения и *точность* представления.

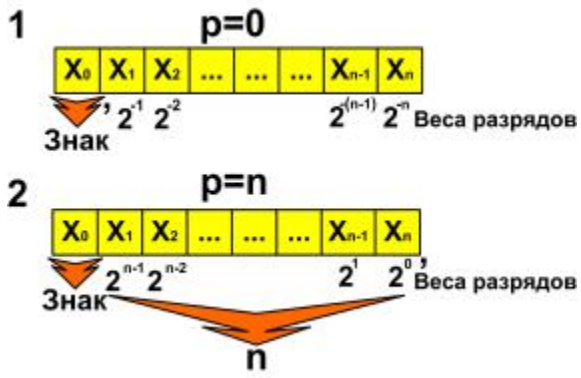


Рис. 2.3. Виды расположения запятой

Диапазон изменения характеризуется теми пределами, в которых могут находиться числа, с которыми оперирует микропроцессорное устройство.

Для первого способа расположения запятой минимально и максимально представимые числа таковы:

$$|\mathrm{X}|_{\max} = 0,\ \underbrace{111\ldots1}_{\text{n разрядов}} = 1 - 2^{-\mathrm{n}},$$

$$|\mathrm{X}|_{\min} = 0,\ \underbrace{000\ldots1}_{\text{n разрядов}} = 2^{-\mathrm{n}}.$$

Тогда диапазон представимых чисел равен $|\mathrm{X}|_{\min} \le |\mathrm{X}| \le |\mathrm{X}|_{\max}$ или $2^{-n} \le |X| \le 1 - 2^{-n}$, т.е. числа, которые выходят за диапазон представления, в микропроцессорных устройствах не могут быть представлены точно. Если $|\mathrm{X}| < |\mathrm{X}_{\min}| < 2^{-\mathrm{n}}$, то такое число воспринимается как нуль и называется машинным нулем. Если $|\mathrm{X}| > |\mathrm{X}_{\max}| > 1 - 2^{-\mathrm{n}}$, то такое число воспринимается как бесконечно большое и называется машинной бесконечностью. При оптимальном округлении абсолютная ошибка равна $|\Delta X| \le 0{,}5 \cdot 2^{-n}$. Минимальная и максимальная относительная ошибки будут при этом равны

$$|dX|_{\min} = \frac{|\Delta X|}{|X|_{\max}} = \frac{0{,}5 \cdot 2^{-n}}{1 - 2^{-n}} \approx 2^{-(n+1)},$$

$$|dX|_{\max} = \frac{|\Delta X|}{|X|_{\min}} = \frac{0{,}5 \cdot 2^{-n}}{2^{-n}} = 0{,}5$$

соответственно. Ошибка представления числа зависит от величины самого числа и способа округления и для малых чисел ошибка может достигать большой величины.

Важнейшим достоинством представления чисел с ФЗ является то, что они позволяют построить несложные АЛУ с высоким быстродействием.

Недостатки формы представления чисел с ФЗ состоят в том, что у них ограниченный диапазон представимых чисел и ограниченная точность.

2.2.2. Представление чисел в форме с плавающей запятой

Представление чисел в форме с плавающей запятой (ПЗ) позволяет следующее:

1) избежать масштабирования исходных чисел;

2) снять необходимость слежения за местоположением запятой программистом;

3) расширить диапазон и точность представляемых чисел.

Число в форме с ПЗ представляется в следующем виде:

$$X = \pm M_x \cdot q^p .$$

Здесь X – само число, $\pm$ – знак числа, M_x – мантисса числа, q – основание системы исчисления, p – порядок числа.

В микропроцессорах нашли применение *нормальная* форма представления чисел с ПЗ. В этом случае на мантиссу числа накладываются два условия:

1. Мантисса должна быть меньше 1.
2. Первая цифра после запятой должна быть отлична от «0».

$M_x = X_0 \cdot 1 \cdot X_2 \cdot X_3 \cdot X_4 \cdot \ldots \cdot X_n$ – нормальная форма числа. Причем X_0 – это знак числа.

Рассмотрим сетку с ПЗ в двоичной системе исчисления (рис. 2.4).

Пусть m разрядов отведено под изображение *мантиссы*, а k разрядов под изображение *порядка*. Тогда для нормализованного вида числа

$$|X|_{max} = 0{,}111\ldots1 \cdot 2^{+11\ldots1} = (1 - 2^{-m}) \cdot 2^{+(2^k - 1)} \quad \text{и}$$

$$|X|_{min} = 0{,}1 \cdot 2^{-11\ldots1} = 2^{-1} \cdot 2^{-2^k + 1} = 2^{-2^k} .$$

То есть диапазон чисел $2^{-2^k} < |X| < (1 - 2^{-m}) \cdot 2^{+(2^k - 1)}$.

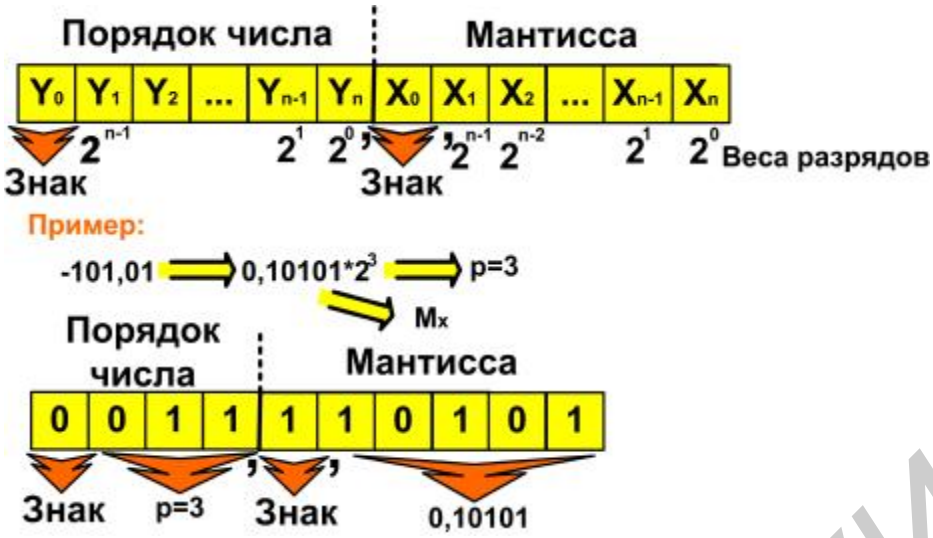


Рис. 2.4. Сетка с ПЗ в двоичной системе исчисления

Абсолютная ошибка представления числа с *плавающей запятой* равна $|\Delta X| \le 0{,}5 \cdot 2^{-m}$. Минимальная и максимальная относительная ошибки будут равны

$$|dX|_{\min} = \frac{0{,}5 \cdot 2^{-m}}{1 - 2^{-m}} \approx 2^{-(m+1)}$$ при m большом и

$$|dX|_{\max} = \frac{0{,}5 \cdot 2^{-m}}{2^{-1}} = 2^{-m}$$ при m малом

соответственно.

Теоретически *плавающая запятая* имеет преимущества перед *фиксированной*.

Достоинство формы представления с ПЗ в том, что относительная ошибка не зависит от *порядка* числа. При этом *точность* представления больших и малых чисел изменяется незначительно. *Плавающая запятая* снимает с программиста обязанность отслеживать положение запятой в вычислениях и значительно упрощает сам процесс программирования вычислительных задач.

Недостаток формы представления чисел с ПЗ заключается в том, что требуется более сложное оборудование для реализации, и оно имеет меньшее быстродействие, так как требует большего числа микроопераций.

Кодирование отрицательных чисел

Основное неудобство построения устройств, реализующих арифметические операции, состоит в сложном характере алгоритма вычитания. Для его преодоления в микропроцессорных устройствах операция вычитания всегда

выполняется по иным правилам, чем это делается обычно. В ее основе лежит операция сложения. Алгоритмы выполнения такого рода операций требуют специальных кодов представления отрицательных чисел.

Для кодирования отрицательных чисел применяют три способа:

прямой код;

дополнительный код;

обратный код.

При кодировании знака числа «+» представляется как «0», а «–» представляется как «1».

Прямой код числа (ПКЧ)

ПКЧ двоичного числа получается в виде его абсолютного значения и кода знака (рис. 2.5.)

Рис. 2.5. Пример получения ПКЧ

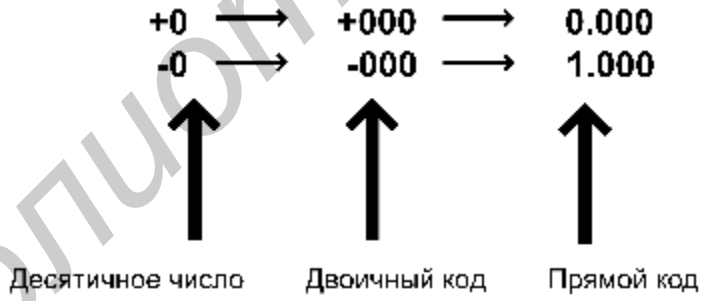


Рис. 2.6. Двоякое изображение нуля в прямом коде

В цифровых разрядах пишется модуль положительного или отрицательного числа.

Недостаток прямого кода:

1) «0» имеет два изображения (рис. 2.6)*;*

2) неудобен для сложения и вычитания чисел, т.к. требует сложной аппаратуры.

Тем не менее прямой код нашел свое применение в хранении чисел в запоминающих устройствах и устройствах ввода-вывода.

Дополнительный код (ДК)

Алгоритм получения ДК:

$$[X]_{ДК} = K - |X|, \text{ где } K = 2^{n-1}.$$

Здесь n – количество разрядов для хранения, включая знаковый (т.е. $[X]_{ДК}$ – это дополнение $|X|$ до определенной константы K).

Достоинство дополнительного кода состоит в том, что *«0» имеет одно изображение.*

Правило получения ДК. В знаковый разряд кода отрицательного числа записать «1», в цифровых разрядах «1» заменить на «0», а «0» заменить на «1». После этого к младшему разряду прибавить «1».

Обратное преобразование от дополнительного к прямому коду осуществляется по тому же правилу.

Рис. 2.7. Пример получения ДК

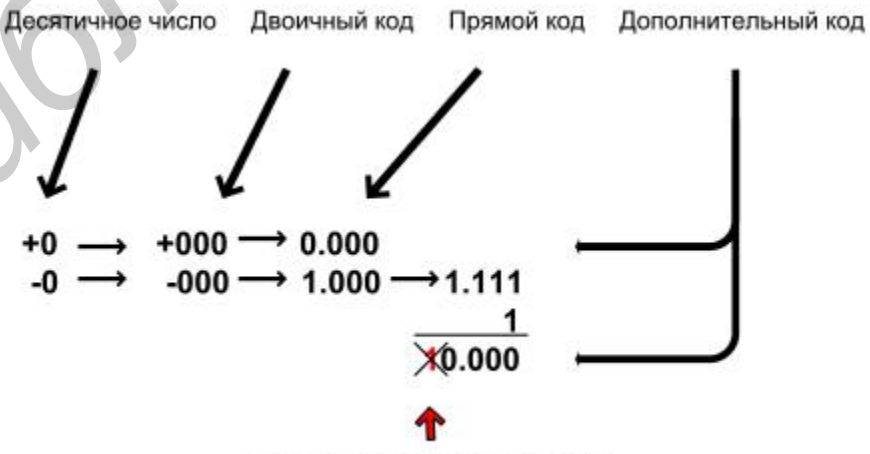


Рис. 2.8. Устранение недостатка отображения нуля

Обратный код (ОК)

Обратный код отрицательного числа **X** получается из выражения

$$[X]_{ОК} = (K - 1) - |X|, \text{ где } K = 2^{n-1}.$$

Здесь n – количество разрядов для хранения, включая знаковый.

Правило получения обратного кода из прямого кода. В знаковый разряд обратного кода записывается «1», а в цифровых разрядах «1» заменяется «0», а «0» заменяется «1».

Обратное преобразование из прямого кода в обратный производится по тому же правилу.

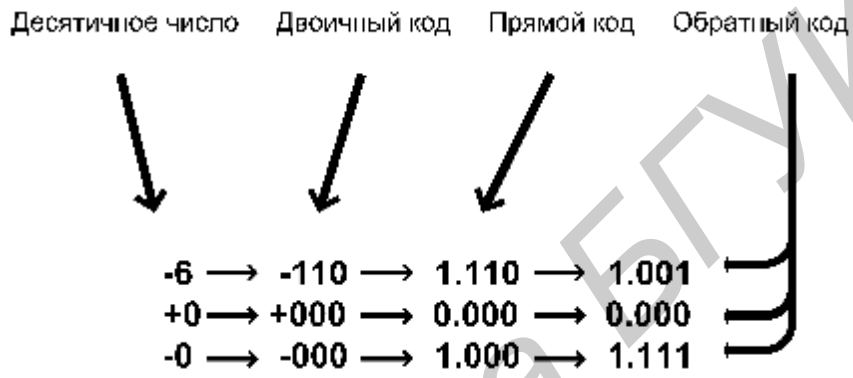


Рис. 2.9. Пример получения ОК

Важным достоинством дополнительного и обратного кодов является то, что при выполнении арифметических операций сложения и вычитания цифру знакового разряда и цифровую часть числа можно рассматривать как единое целое и обращаться со знаковым разрядом так же, как и с разрядами цифровой части числа.

2.2.3. Представление алфавитно-цифровой информации в микропроцессорных устройствах

В микропроцессорах все символы кодируются восьмью разрядными кодами (байтами). Каждому такому символу соответствует свой уникальный код. Посредством байтов можно закодировать $2^8 = 256$ символов.

Специальная форма хранения данных *– строка символов.*

Строка символов *– формат хранения алфавитно-цифровой информации; может быть разной длины, но не может превышать 256.*

Для кодирования алфавитно-цифровых символов применяются стандартные коды для обмена информации:

а) КОИ – 8; б) ANSI.

3. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Основной особенностью различных методов выполнения арифметических операций является то, что любая операция (сложение, вычитание, умножение, деление и др.) сводится к некоторой последовательности микроопераций, таких, как:

- сложение;
- сдвиг;
- передача;
- преобразование кодов.

Сложение выполняется по правилам сложения чисел в позиционных системах счисления, т.е. эта операция выполняется поразрядно, а возникающий в младших разрядах перенос направляется в старшие разряды.

3.1. Поразрядные операции над числами

Поразрядные операции над числами – это операции, которые выполняются над одноименными разрядами чисел независимо от соседних разрядов.

1. Поразрядное дополнение (операция получения инверсионного кода)

Набор символов $X_0, X_1, X_2 ... X_n$ превращается в следующий набор $\overline{X}_0, \overline{X}_1, \overline{X}_2 ... \overline{X}_n$. $\overline{X}_i = 1 - X_i$.

2. Поразрядное сложение двух чисел

Сложение одноименных разрядов чисел по модулю происходит в соответствии с правилами:

$$0 \oplus 0 = 0 \qquad 0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1 \qquad 1 \oplus 1 = 0.$$

(эта операция применяется при сравнении двух чисел на равенство).

3. Поразрядное логическое сложение

Сложение одноименных разрядов двух чисел происходит в соответствии с правилами:

$$0 \vee 0 = 1 \qquad 0 \vee 1 = 1 \qquad 1 \vee 0 = 1 \qquad 1 \vee 1 = 0.$$

Здесь «$\vee$» – логическое сложение.

4. Поразрядное логическое умножение

Умножение одноименных разрядов двух чисел происходит в соответствии с правилами:

$$0 \wedge 0 = 0 \qquad 0 \wedge 1 = 0 \qquad 1 \wedge 0 = 0 \qquad 1 \wedge 1 = 1.$$

Здесь « ^ » – логическое умножение.

Операции ***3*** и ***4*** используются для модификации команд и чисел.

Операции сдвигов

Для микропроцессорных вычислителей сдвиг влево на один разряд эквивалентен умножению на 2, а сдвиг вправо – делению на 2. Сдвиг заключается в одновременном переносе всех разрядов числа на определенное число разрядов влево или вправо.

Используют три вида сдвигов:

-логический (5);

-циклический(6);

-арифметический(7).

5. Логический сдвиг

Заключается в смещении всей числовой последовательности слова, включая разряд знака. Причем в освободившиеся разряды записываются нули. Выходящие за пределы разрядной сетки числа теряются.

X1, X2 … X(*n*–1), *Xn* => 0, 0, *X1 …X*(*n*–3), *X*(*n*–2)

6. Циклический сдвиг

Заключается в смещении всей числовой последовательности, при которой цифры, выходящие за пределы разрядной сетки, вводятся в освободившиеся при сдвиге позиции с другой стороны разрядной сетки.

X1, X2 … X(*n*–1), *Xn* => *X*(*n*–1), *Xn*, *X1 …X*(*n*–3), *X*(*n*–2).

7. Арифметический сдвиг

Заключается в сдвиге всей числовой последовательности без изменения позиции знака. Различают два арифметических сдвига: ***простой*** и ***модифицированный***.

Простой арифметический сдвиг эквивалентен умножению числа на основание системы счисления (т.е. на 2), возведенную в степень, равную величине сдвига. *Модифицированный* арифметический сдвиг используют для чисел с плавающей запятой. При таком сдвиге величина исходного числа не меняется, так как наряду с изменением мантиссы изменяется и порядок числа.

$1101 == 1101 \cdot 2^0 = +0{,}1101 \cdot 2^4 \overset{k=1}{\rightarrow} +0{,}01101 \cdot 2^{4+1}$.

8. Нормализация

Заключается в последовательном выполнении операции модифицированного арифметического сдвига до выполнения условия нормализации (рис. 3.1).

$$X=\pm 0.\underbrace{0001}_{k}X_2X_3\ldots Xn*2^{Px},$$

$$X=\pm 0.1X_2X_3\ldots Xn*2^{Px-k}.$$

Рис. 3.1. Пример нормализации мантиссы

Двоичная арифметика

Сложение **a+b**

		a	
		0	1
b	0	0	1
	1	1	0

→1 *в старший разряд*

Вычитание **a–b**

		a	
		0	1
b	0	0	1
	1	1	0

→ *заем 1 из старшего разряда*

Необходимо учитывать увеличение веса заема из старшего разряда.

Умножение a·b

		a	
		0	1
b	0	0	0
	1	0	1

Деление

В целом алгоритм аналогичен выполнению деления в десятичной системе:

```
-101000 | 11
  11    |----
 -100   | 1101
   11
  -0100
    11
     1
```

Все рассмотренные алгоритмы используются при расчетах вручную и неудобны для машинной реализации. Разработаны специальные алгоритмы с

использованием кодирования, в качестве базовых используются операции сложения, сдвига и поразрядного дополнения.

3.2. Операции над числами с фиксированной запятой

3.2.1 Сложение и вычитание целых чисел с использованием дополнительного и обратного кодов

Операция вычитания реализуется через операцию сложения путем изменения на противоположный знак вычитаемого числа. То есть **a–b = a+(–b)**.

Пусть два числа представлены в *n*-разрядном коде, где *n* – число разрядов, включая знаковый. Тогда коды обоих чисел суммируются по правилам двоичного сложения разряд за разрядом, включая и старший знаковый разряд.

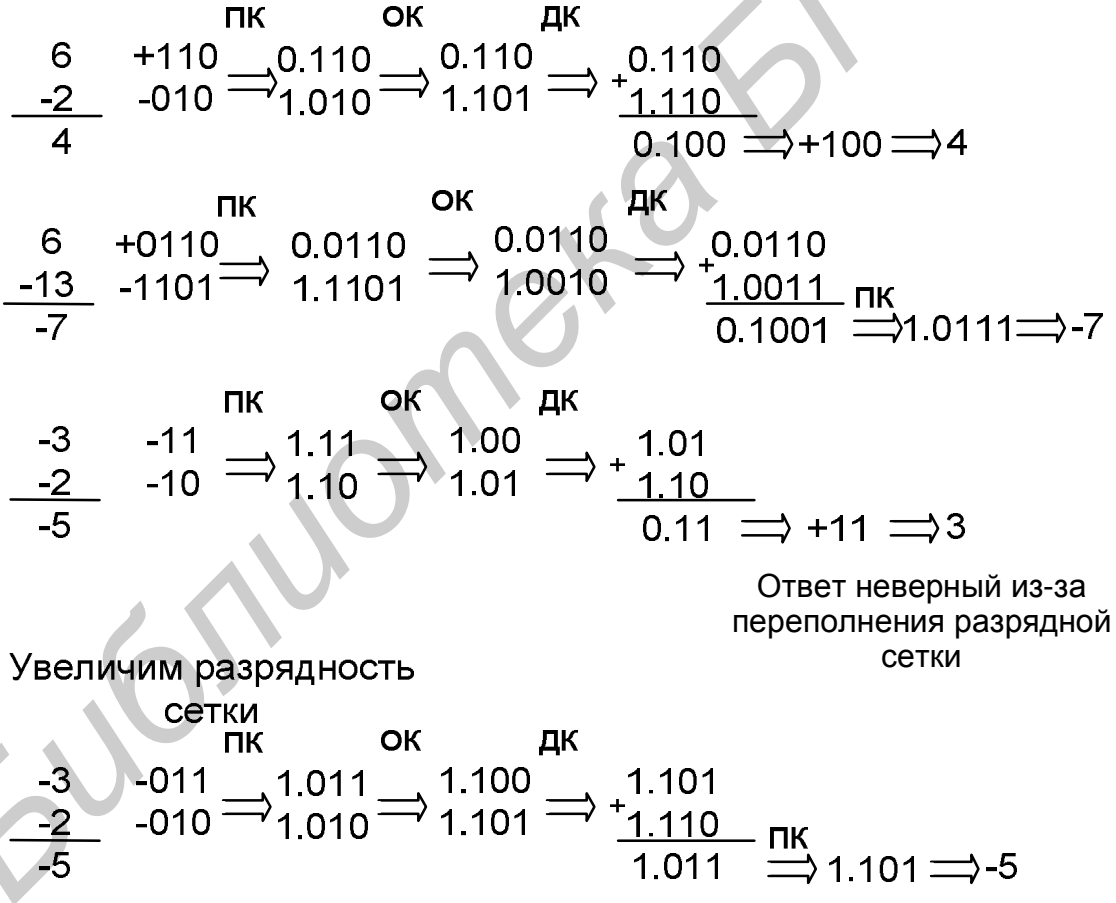


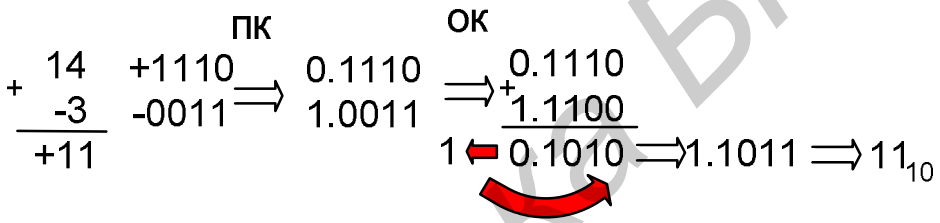
Рис. 3.2. Примеры выполнения арифметических операций и контроля переполнения разрядной сетки

Возможный перенос из старшего разряда отбрасывается, ответ также получается в дополнительном коде и будет правильным, если ответ лежит в

пределах от $-(2^{n-1}-1)$ до $(2^{n-1}-1)$. Если оба кода слагаемых имеют одинаковый знак, сумма может выйти за пределы сетки. При этом происходит переполнение разрядной сетки. Его необходимо учитывать, так как результат вычисления будет неверным. Самый распространенный способ контроля переполнения заключается в анализе значений переноса в знаковый разряд и из знакового разряда. Переполнение имеет место, когда значения различны.

Сложение в обратном коде

Коды разрядов, включая знаковый, суммируются по правилам сложения двоичных чисел. Если возникает перенос из старшего знакового разряда, он прибавляется к младшему разряду результата. Такой перенос называют круговым , а сложение с ним – циклическим . Факт переполнения определяется по переносам из знакового и в знаковый разряд до выполнения кругового переноса.



Сложение и вычитание действительных чисел

Для чисел с фиксированной запятой при порядке, равном нулю, операции сложения и вычитания осуществляются над числами в пределах от – 1 до +1. Для таких чисел методика выполнения операций сложения и вычитания в дополнительном и обратном кодах, включая признаки переполнения, аналогична методике сложения и вычитания целых чисел.

Сложение и вычитание чисел в двоично-кодированной десятичной системе

Если в результате сложения возникает недопустимая комбинация (10 – 16, т.е. 1010 – 1111), то должен произойти перенос в следующий разряд с коррекцией недопустимой комбинации.

$$
\begin{array}{rr}
+\,27_{10} & +\,0010\ 0111 \\
36_{10} & 0011\ 0110 \\
\hline
63_{10} & 0101\ 1101 \\
 & \underbrace{}_{5}\ \underbrace{}_{13}
\end{array}
$$

Возникла недопустимая комбинация

Эти требования удовлетворяются, если прибавить корректирующее слагаемое 0110 к каждой недопустимой комбинации, разрешив при этом перенос единицы из одной тетрады в другую.

$$
\begin{array}{r}
{\scriptstyle 1}\ \ \\
0101\ 1101 \\
+\quad\ 0110 \\
\hline
0110\ 0011 \\
\underbrace{}_{6}\ \underbrace{}_{3}
\end{array}
$$

Как видно из примера, если все верно проделать, то ответ также будет верным.

При сложении коррекция результата требуется также тогда, когда был совершен перенос из младшей тетрады в старшую. Корректирующее слагаемое 0110 прибавляется к той тетраде, из которой был перенос.

$$
\begin{array}{rr}
 & {\scriptstyle 1}\ \ \\
28_{10} & +\,0010\ 1000 \\
59_{10} & 0101\ 1001 \\
\hline
87_{10} & 1000\ 0001 \\
 & \underbrace{}_{8}\ \underbrace{}_{1}
\end{array}
$$

$$
\begin{array}{r}
1000\ 0001 \\
+\quad\ 0110 \\
\hline
1000\ 0111 \\
\underbrace{}_{8}\ \underbrace{}_{7}
\end{array}
$$

Вычитание реализуется через сложение путем замены знака. Используется представление операндов в дополнительном и обратном кодах. При этом обратный код получается из прямого по формуле

$$Y = y_0 y_1 y_2 \ldots y_i \ldots y_n,$$

$$Y[\text{ОК}] = y_0' y_1' y_2' \ldots y_i' \ldots y_n',$$

$$y_i' = 9 - y_i\,.$$

Умножение и деление чисел с фиксированной запятой

Умножение

Умножение двоичных чисел реализуется через операцию сложения и сдвига. Перемножение выполняют в прямом коде. Знак произведения определяют по знаковым разрядам множимого и множителя. Знак произведения не влияет на алгоритм выполнения операции умножения.

На практике чаще используют другой способ. Результат получают суммированием частичных произведений, являющихся результатом умножения множимого на значения очередных разрядов множителя. При этом каждое следующее произведение вдвое превышает предыдущее, что соответствует его сдвигу на один разряд влево.

$$\begin{array}{r} \times 13_{10} \\ \times 11_{10} \\ \hline + 13 \\ 13 \\ \hline 143 \end{array} \qquad \begin{array}{r} \times 1101 \\ \times 1011 \\ \hline 1101 \\ + 1101 \\ 0000 \\ \hline 1101 \\ \hline 10001111 = 143_{10} \end{array}$$

При перемножении мантисс (правильных дробей) младшие разряды, выходящие за пределы разрядной сетки, могут быть с округлением или без округления.

Наибольшую скорость умножения получают в статических множительных устройствах, хранящих таблицы умножения многоразрядных чисел. Однако эти устройства сложны и реализуемы для сравнительно небольшого количества разрядов.

Деление

Операцию деления можно выполнить способом, аналогичным делению вручную. Знак определяется аналогично алгоритму умножения.

Наибольшее распространение получили два алгоритма деления – с *восстановлением* и *без восстановления остатка*.

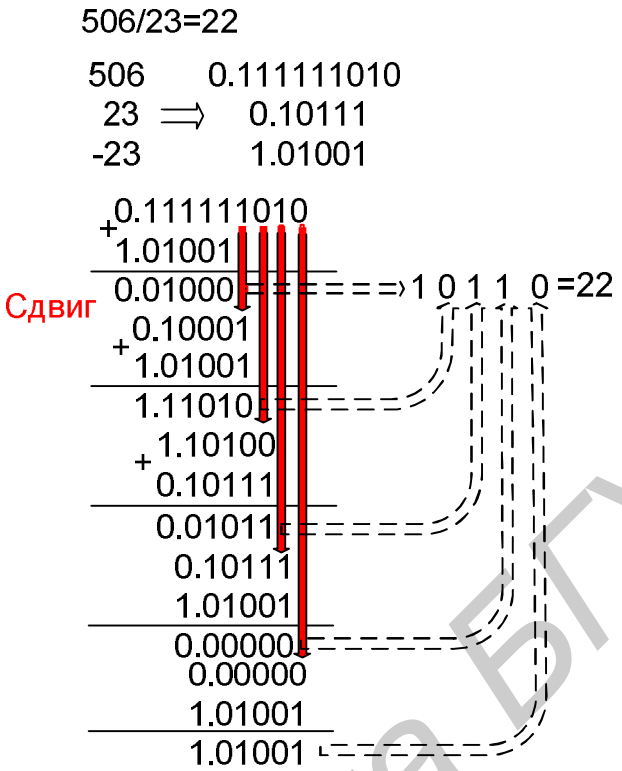
Суть алгоритма деления без восстановления остатка:

1. Из делимого вычитается делитель.

2. Если остаток положителен, первая цифра частного равна единице, если отрицателен – нулю.

3. Остаток сдвигается влево на один разряд и к нему прибавляется делитель со знаком, обратным знаку остатка. Знак следующего остатка определяет следующую цифру частного.

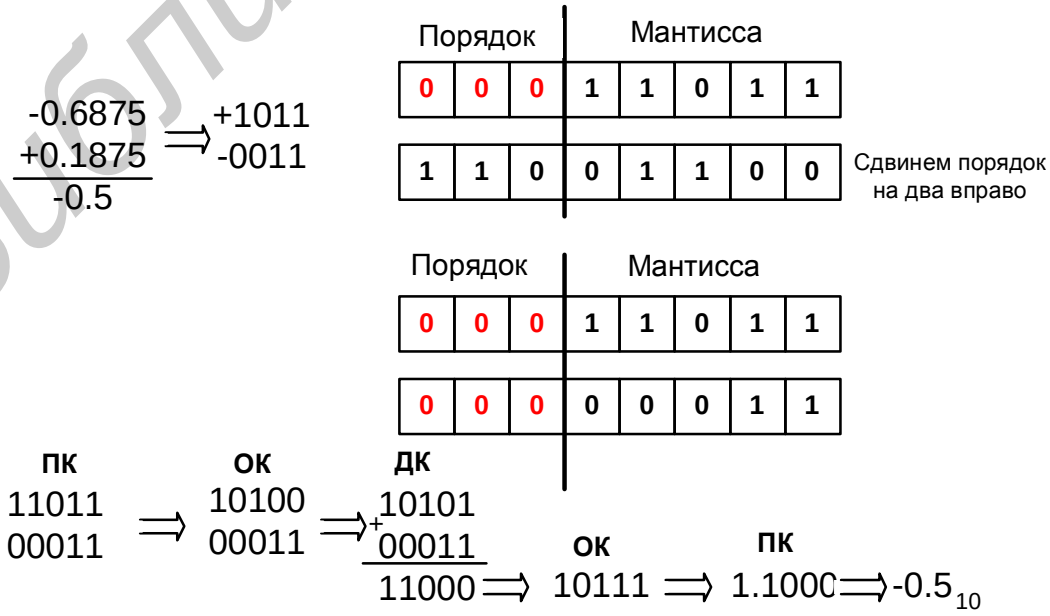
Действие п. 2 проводится до тех пор, пока не получится требуемое число разрядов частного или нулевой остаток.



3.3. Операции над числами с плавающей запятой

3.3.1. Сложение и вычитание чисел с плавающей запятой

Сложение и вычитание чисел с плавающей запятой с равными порядками также ничем не отличается от аналогичных операций для чисел с фиксированной запятой.



В случае если порядки исходных чисел не равны, перед проведением операций производится выравнивание порядков, при этом осуществляется денормализация одной из мантисс.

3.3.2. Умножение и деление двоичных чисел с плавающей запятой

Операция умножения чисел в форме с плавающей занятой производится в четыре этапа:

1. Определяется знак произведения.
2. Перемножаются мантиссы сомножителей.
3. Устанавливается порядок произведения алгебраическим сложением порядков сомножителей.
4. При необходимости производится нормализация результата.

Деление чисел в устройствах с плавающей запятой производится так же, как и умножение – в четыре этапа. Особенность заключается в том, что поскольку все числа в этом случае не должны превышать единицу, мантисса делимого должна быть выбрана меньше мантиссы делителя. Вторая особенность – вычитание порядка делителя из порядка делимого. В случае необходимости производится нормализация результата.

4. ЦИФРОАНАЛОГОВОЕ И АНАЛОГО–ЦИФРОВОЕ ПРЕОБРАЗОВАНИЕ

Большинство сигналов, с которыми мы непосредственно сталкиваемся в окружающем мире, аналоговые. Для того чтобы обрабатывать их в микропроцессорных устройствах, необходимо преобразовать их из аналоговых в цифровые. Эту операцию выполняют аналого-цифровые преобразователи. Обработанные в МПУ сигналы выдаются затем получателю, причем зачастую в аналоговой форме. Для выполнения этой задачи используются цифро-аналоговые преобразователи.

4.1. Цифроаналоговое преобразование

Цифроаналоговым преобразователем (ЦАП) называется устройство для получения выходной аналоговой величины, соответствующей цифровому коду, поступающему на вход преобразователя. ЦАП применяют в трех случаях:

1) для согласования цифрового устройства с аналоговым;
2) в качестве внутренних узлов в АЦП;

3) в качестве самостоятельных функциональных узлов и блоках радио-и электроаппаратуры.

При создании ЦАП применяют два метода:

1) суммирование единичных эталонных величин;

2) суммирование эталонных величин, веса которых отличаются.

Выражение для нахождения выходной аналоговой величины ЦАП:

$$X = P(a_1 \cdot 2^{-1} + a_2 \cdot 2^{-2} + \mathbf{L} + a_b \cdot 2^{-b}),$$

где X – выходная аналоговая величина;

P – некий опорный уровень;

a – коэффициенты, соответствующие двоичным разрядам, они либо «1», либо «0»;

b – количество разрядов преобразуемого двоичного числа.

Эта запись отображает представление величины в виде правильной дроби.

В качестве опорного сигнала P можно принимать постоянные либо переменные напряжения.

Если в качестве опорного сигнала используется переменное напряжение, то ЦАП реализует также перемножение аналоговой величины в цифровой код. В ЦАП формируются эталонные величины, соответствующие значениям разрядов входного кода, которые затем суммируются и образуют дискретные значения выходной аналоговой величины.

ЦАП классифицируются по следующим признакам:

1. По способу формирования выходного сигнала:
   - суммированием напряжений;
   - делением напряжений;
   - суммированием токов.
2. По роду выходного сигнала:
   - с выходом по току;
   - с выходом по напряжению.
3. По полярности выходного сигнала:
   - униполярные;
   - биполярные.
4. По конструктивно-логическому исполнению:
   - модульные;
   - гибридные;
   - интегральные.
5. По типу элементов, использованных для суммирования:
   - резистивные;
   - емкостные;

- оптоэлектронные.

В ЦАП интегрального исполнения в основном используется формирование выходного сигнала с суммированием токов. При этом применяются два основных схемотехнических способа:

- ЦАП со взвешенными резисторами;
- ЦАП с цепочкой типа R-2R.

4.1.1. ЦАП со взвешенными резисторами

Наиболее простой ЦАП со взвешенными резисторами (рис. 4.1) состоит из двух блоков. Резистивная матрица выполнена на резисторах $R_0 ... 2^{b-1}R_0$. Суммирующий усилитель включает в себя операционный усилитель (ОУ) и резистор обратной связи $R_{ос}$. Опорное напряжение подключается к резисторам матрицы переключателями $K_1 \ldots K_b$, управляемыми коэффициентами преобразуемого кода.

Недостатком ЦАП со взвешенными резисторами является необходимость подбора резисторов с различными номиналами; при этом сопротивления резисторов изменяются в широких пределах и должны быть выдержаны с высокой точностью.

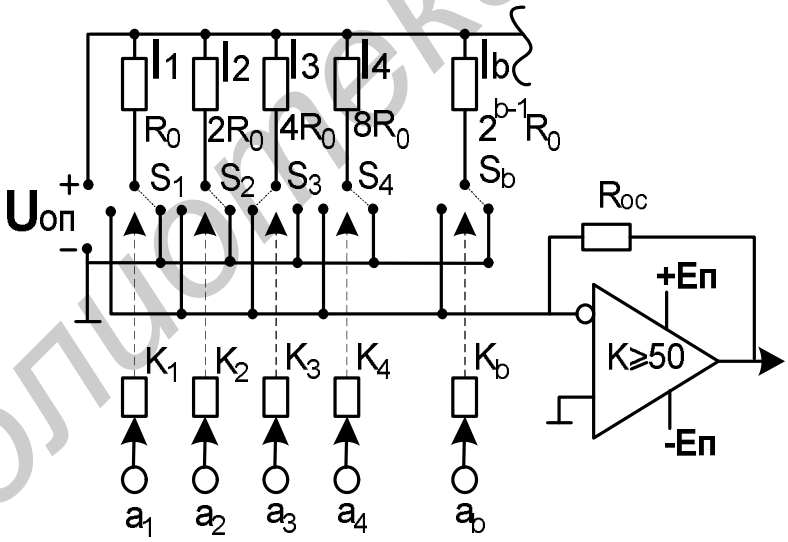


Рис. 4.1. ЦАП со взвешенными резисторами

Это создает большие трудности, особенно при реализации ЦАП в интегральном исполнении.

4.1.2. ЦАП с цепочкой резисторов типа *R*-2*R*

ЦАП с цепочкой резисторов R-2R (рис. 4.2) содержит резистивную матрицу *R*-2*R* и суммирующий усилитель.

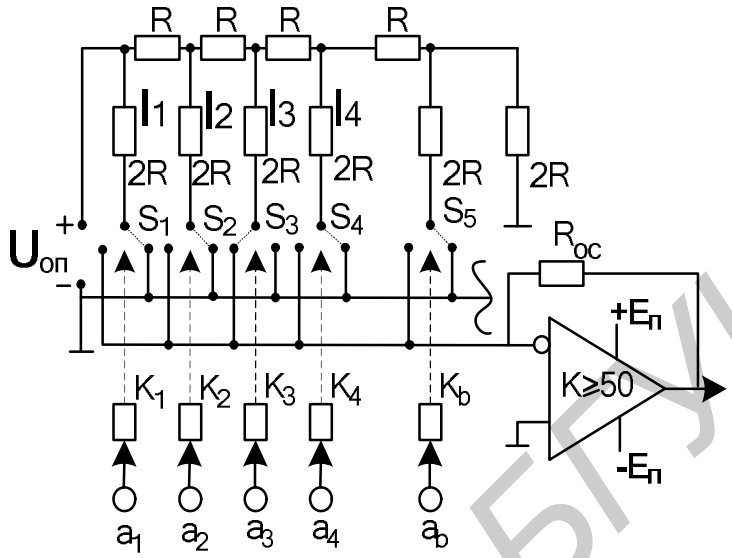


Рис. 4.2. ЦАП с цепочкой резисторов *R*-2*R*

Преимущество такой матрицы в том, что используются резисторы только двух номиналов, отличающиеся в два раза.

В каждом узле напряжение в два раза меньше предыдущего.

ЦАП, выполненный на цепочке типа *R*-2*R*, является быстродействующим, так как источник опорного напряжения нагружен на постоянное сопротивление, равное *R*, что уменьшает длительность переходных процессов. В отличие от ЦАП со взвешенными резисторами ЦАП с цепочкой резисторов типа *R*-2*R* не требует широкого диапазона сопротивлений, что упрощает задачу получения интегральной матрицы сопротивлений.

4.2. АНАЛОГО-ЦИФРОВОЕ ПРЕОБРАЗОВАНИЕ

4.2.1. АЦП с ЦАП в цепи обратной связи следящего типа

Структурная схема АЦП приведена на рис. 4.3. Принцип работы такого АЦП основан на сравнении напряжения на входе АЦП с напряжением на выходе ЦАП в цепи обратной связи. На входе сравнивающего элемента (интегрального компаратора) вырабатывается сигнал добавления или вычитания, который

поступает на реверсивный счетчик. На цифровых выходах счетчика формируется цифровой код, поступающий на ЦАП в цепь обратной связи и на выход АЦП.

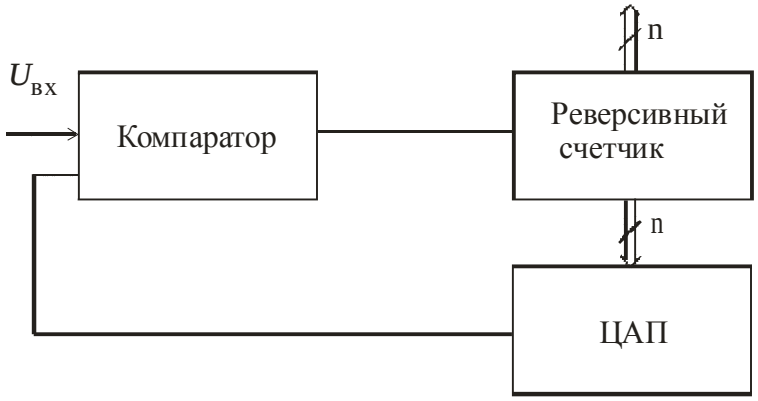


Рис. 4.3. АЦП с ЦАП в цепи обратной связи следящего типа

4.2.2. АЦП с ЦАП в цепи обратной связи последовательного счета

Принцип работы такого АЦП показан на диаграммах (рис. 4.4). Если на прямом и инверсном входах сравнивающего элемента (интегрального компаратора) действуют соответственно входной ($U_{вх}$ – постоянное напряжение, равное мгновенному значению преобразуемого сигнала) и эталонный ($U_{эт}$ – линейно изменяющееся эталонное напряжение) сигналы, то длительность выходного импульса $T_{изм}$ будет пропорциональна напряжению $U_{вх}$. Таким образом осуществляется преобразование типа «напряжение – интервал времени». В интервале $T_{изм}$ импульсы U_T тактовой частоты суммируются двоичным счетчиком, на выходе которого получим двоичный код, пропорциональный интервалу $T_{изм}$ и напряжению $U_{вх}$.

Эталонное напряжение $U_{эт}$ можно сформировать как с помощью высоколинейной импульсной схемы генератора линейно изменяющегося напряжения (ГЛИН), так и цифровой схемы ГЛИН. В последнем случае суммирующий счетчик и ЦАП формируют ступенчатое напряжение $U_{эт}$, которое при необходимости может быть преобразовано в линейно изменяющееся напряжение с помощью фильтра нижних частот. Вариант такой схемы приведен на рис. 4.5.

Эталонное напряжение для инверсного входа компаратора формируется суммирующим счетчиком и ЦАП. На прямом входе компаратора мгновенное значение входного сигнала: (в момент $t = T_{изм}$) переход из единичного состояния в нулевое выходного сигнала компаратора переписывает содержимое счетчика в выходной регистр. После переполнения счетчика преобразование повторяется.

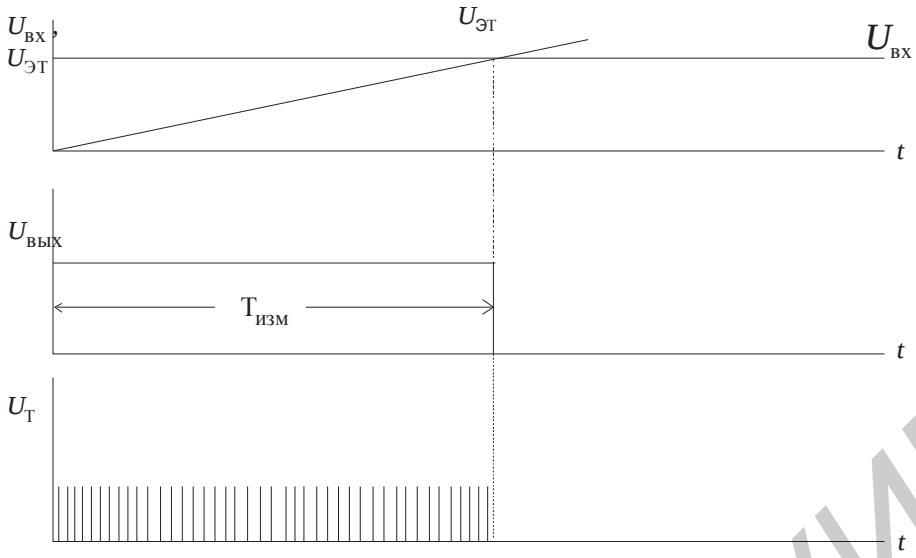


Рис. 4.4. Диаграммы работы АЦП последовательного счета

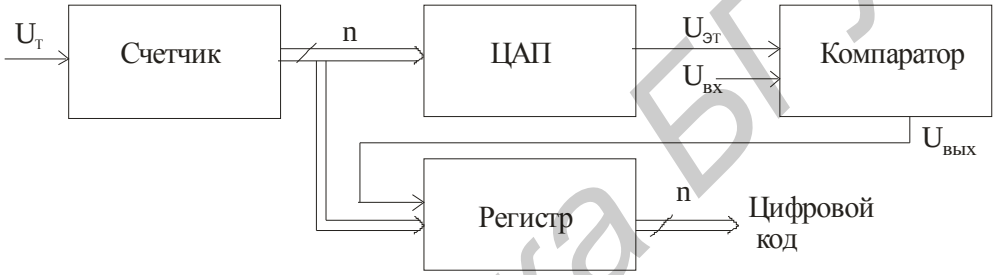


Рис. 4.5. АЦП последовательного счета

Общий недостаток АЦП с последовательным счетом – низкое быстродействие. Кроме того, время преобразования в таком АЦП зависит от преобразуемого аналогового сигнала. Поэтому данные АЦП находят применение только в устройствах с низким быстродействием, например в цифровых вольтметрах.

4.2.3. АЦП с ЦАП в цепи обратной связи последовательного приближения

Функциональная схема АЦП с последовательным приближением, показанная на рис. 4.6, отличается от схемы АЦП с последовательным счетом главным образом тем, что вместо двоичного счетчика используется регистр последовательного приближения (РПП).

В РПП прямой переход («0»→«1») тактового сигнала $U_т$ последовательно подключает к входу ЦАП D-триггеры регистра (начиная с триггера старшего разряда) и записывает в них значение 1, а обратный («1»→«0») переход производит запись текущего значения D («0» или «1») в подключенный триггер.

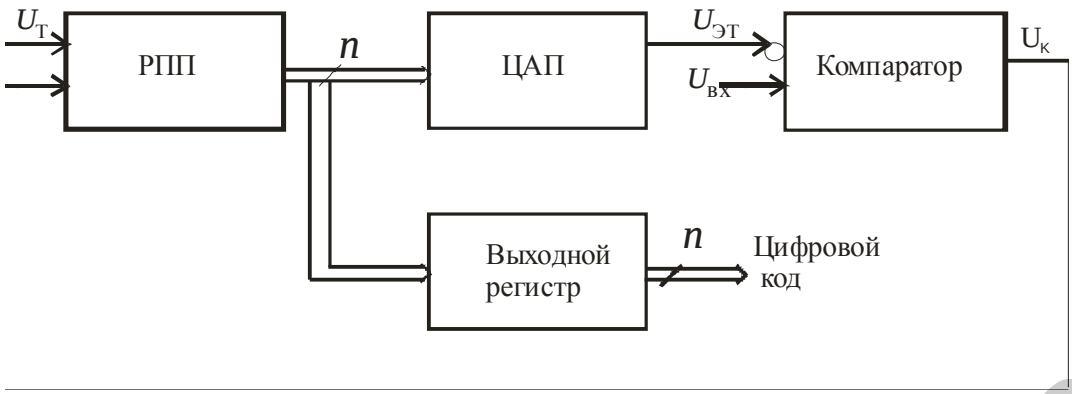


Рис 4.6. АЦП последовательного приближения

Временные диаграммы работы АЦП с РПП приведены на рис. 4.7.

Прямой переход первого тактового импульса записывает «1» в старший разряд РПП. На выходе компаратора при этом появляется 0, поскольку $U_{эт}$ (с выхода ЦАП) в данном случае превышает напряжение $U_{вх}$. С выхода компаратора сигнал «0» проходит на вход D РПП и при обратном переходе $U_т$ переписывается с этого входа в триггер старшего разряда. При этом напряжение на выходе ЦАП падает, и на выходе компаратора вновь появляется сигнал «1». При появлении второго импульса это значение присваивается триггеру второго по старшинству разряда, а на выходе ЦАП появляется напряжение, соответствующее «весу» второго разряда, который в два раза меньше первого. При $U_{эт} < U_{вх}$ на выходе компаратора сохраняется «1», и это значение присваивается второму триггеру при обратном переходе второго импульса $U_т$.

Таким образом, осуществляется поразрядное уравновешивание кода в РПП с уровнем входного сигнала. После тактового импульса с номером N (N – число разрядов РПП и всего АЦП) произойдет запоминание самого младшего разряда РПП и содержимое этого регистра можно переписывать в выходной регистр в качестве результата преобразования.

Быстродействие АЦП с РПП значительно выше, чем у АЦП последовательного счета.

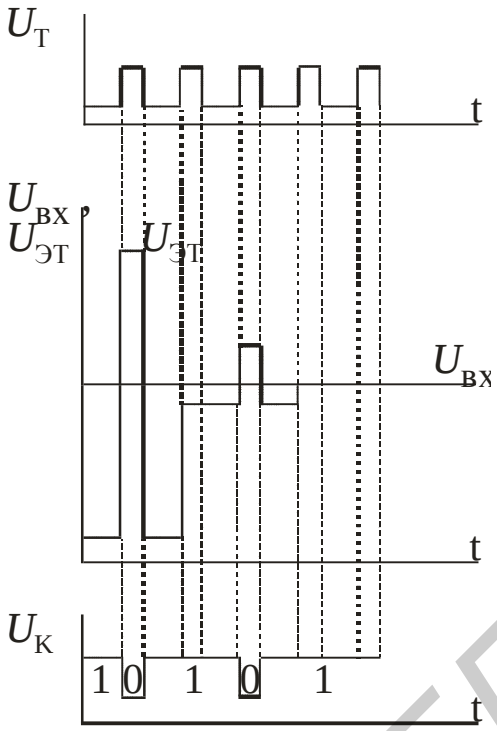


Рис 4.7. Работа АЦП последовательного приближения

Тем не менее быстродействия АЦП с РПП недостаточно для преобразования широкополосных сигналов, например стандартного телевизионного видеосигнала.

4.2.4. АЦП двойного интегрирования

Способ двойного интегрирования позволяет хорошо подавлять сетевые помехи. Кроме того, для построения схемы АЦП не требуются ЦАП с высокоточными резистивными матрицами.

Функциональная схема АЦП двойного интегрирования приведена на рис. 4.8 и напоминает схему АЦП последовательного счета, в которой вместо ЦАП применен интегратор.

Счетчик запускается от генератора в момент поступления на интегратор входного сигнала $U_{вх}$, из которого за время интегрирования делается выборка. За время выборки напряжение на выходе интегратора $U_{инт}$ увеличивается. В момент t_1 прямое интегрирование заканчивается, входной сигнал от интегратора отключается и к его входу подключается эталонное напряжение. От времени t_1 до момента t_2 продолжается разряд интегратора (обратное, второе интегрирование) с постоянной скоростью.

Интервалы времени от t_1 до нулевого значения напряжения на выходе интегратора пропорциональны уровню входного сигнала (рис. 4.9).

Существенным преимуществом преобразователя является простота компенсации наводок сети промышленного питания.

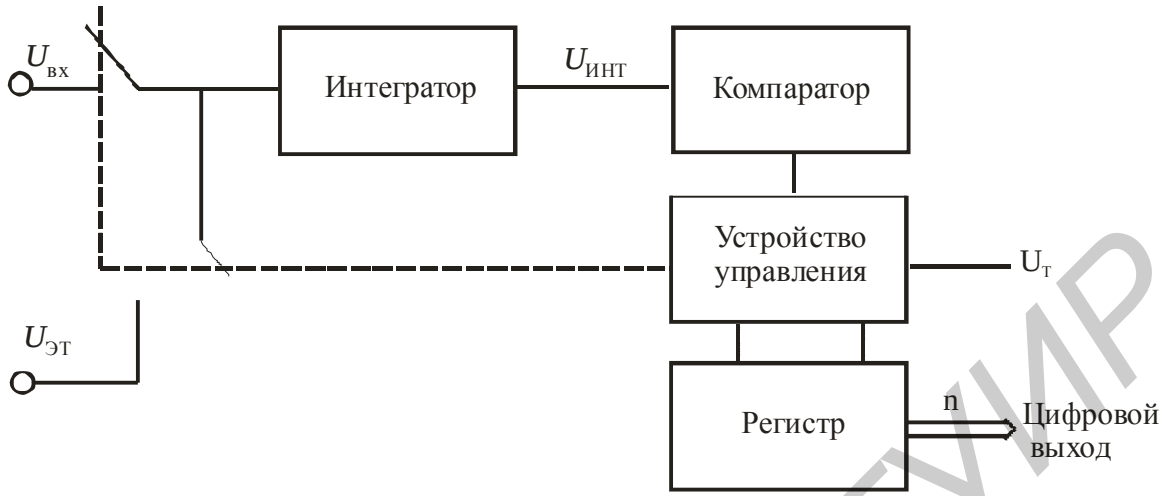


Рис. 4.8. АЦП двойного интегрирования

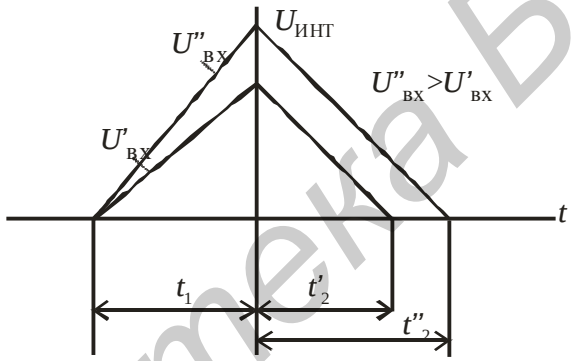


Рис 4.9. Работа АЦП двойного интегрирования

4.2.5. АЦП параллельного преобразования

Структурная схема АЦП параллельного действия показана на рис. 4.10. Она содержит набор входных компараторов, шифратор, преобразующий выходные сигналы компараторов в двоичный код, и выходной регистр, в котором этот код сохраняется.

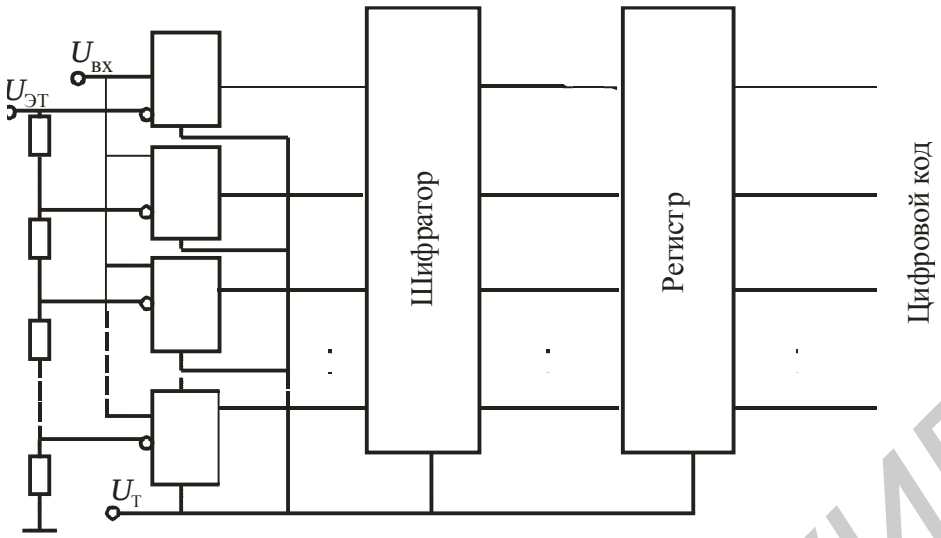


Рис. 4.10. АЦП параллельного преобразования

При $U_т = 0$ выходы всех компараторов закрыты. На их прямых входах постоянно действует входной аналоговый сигнал $U_{вх}$, а на инверсные входы подано постоянное опорное напряжение с резисторов делителя, подключенных к источнику опорного напряжения $U_{эт.}$ При $U_т = 1$ разрешается работа компараторов и на их выходах появляются сигналы, уровни которых при некотором значении $U_{вх}$ равны «0» для компараторов, для которых $U_{вх} < U_{эт}$, и равны «1» для компараторов, у которых $U_{вх} > U_{эт}$.

Сигналы с выходов компараторов поступают в шифратор, который при $U_т = 1$ закрыт по инверсному входу разрешения. При обратном переходе («1» → «0») сигнала $U_т$ компараторы выключаются, а шифратор выдает двоичный код на входы выходного регистра. При следующем прямом переходе («0» → «1») сигнала $U_т$ происходит запись двоичного числа (N-го отсчета) в регистр и это число заменяет предыдущее (N–1 значение). В это же время ($U_т = 1$) в закрытом шифраторе хранится код (N+1)-го отсчета, а открытые компараторы обрабатывают (N+2)-й отсчет сигнала.

Таким образом, в параллельном АЦП действует конвейер, при этом период следования отсчетов может быть равен времени задержки одного из трех узлов (худшего по быстродействию), а не сумме этих времен.

Достоинством АЦП параллельного преобразования является высокое быстродействие, а недостатком – резкое увеличение аппаратной сложности с увеличением разрядности, а также высокие требования к точности изготовления резисторов в делителе, формирующем опорные напряжения для компараторов.

5. АРХИТЕКТУРА ОДНОКРИСТАЛЬНОГО МИКРОКОНТРОЛЛЕРА PIC16F628

5.1. Общие сведения

Однокристальные микроконтроллеры (ОМК) содержат в одном корпусе интегральной микросхемы все функциональные блоки электронной вычислительной машины: арифметико-логическое устройство (АЛУ), устройство управления (УУ), операционные регистры, управляющие регистры, память программ, память данных, порты ввода/вывода. Кроме этого, на кристалле микросхемы часто размещают и периферийные устройства, такие, как таймеры, аналого-цифровой преобразователь (АЦП), цифроаналоговый преобразователь (ЦАП), компараторы, энергонезависимая память, драйверы стандартных последовательных интерфейсов и др. Однокристальные микроконтроллеры предназначены для работы по одной программе, записываемой в энергонезависимую память программ и не меняемой в течение всего цикла эксплуатации. Они, как правило, являются встроенными в аппаратуру вычислительными машинами и выполняют функции управления, а также формирования и обработки сигналов и информации.

В настоящее время ОМК производят несколько десятков фирм. Среди них микроконтроллеры PICmicro американской фирмы Microchip Technology Inc. отличаются высокой производительностью, низким энергопотреблением, гибкой и развитой архитектурой, широкими функциональными возможностями, развитой периферией, простотой в освоении, низкой стоимостью. В зависимости от производительности и функциональных возможностей PICmicro подразделяются на семейства: PIC12, PIC16, PIC17 и PIC18. Самые простые модели выпускаются в 8-выводных, а сложные – в 80-выводных корпусах. При этом все семейства имеют общую базовую архитектуру и систему команд, что упрощает их изучение. Такое сочетание потребительских свойств сделало PICmicro одними из самых популярных ОМК у разработчиков радиоэлектронной аппаратуры.

Для учебных целей нами выбрана модель PIC16F628. Она отличается простотой, но при этом несет в себе важнейшие черты всех семейств, что облегчает освоение других моделей микроконтроллеров. Наличие встроенной электрически перепрограммируемой памяти программ позволяет легко тестировать разрабатываемые схемы и программы.

Ядро однокристального микроконтроллера PIC16F628 разработано в соответствии с модифицированной гарвардской RISC-архитектурой и изготавливается по высокоскоростной КМОП-технологии. Оно имеет

внутреннюю энергонезависимую электрически перепрограммируемую память программ (FLASH) емкостью 2048 14-разрядных слов, 8-битную длину машинного слова и 224-байтную внутреннюю память данных. Система команд включает 35 инструкций. Все команды имеют длину в одно слово шириной 14 бит и исполняются за один машинный цикл (200 нс при максимальной тактовой частоте 20 МГц), кроме команд перехода, которые выполняются за два цикла (400 нс).

PIC16F628 имеет прерывание, срабатывающее от десяти источников, и восьмиуровневый аппаратный стек.

Периферия включает в себя три таймера, два аналоговых компаратора, источник опорного напряжения, модуль широтно-импульсной модуляции, последовательный синхронно-асинхронный приемопередатчик USART, энергонезависимую память данных EEPROM, 15 линий двунаправленного ввода/вывода и др.

В настоящей работе целесообразно ограничиться изучением лишь цифровых портов ввода/вывода и 8-битного таймера/счетчика TMR0 с 8-битным программируемым предварительным делителем.

Особенность портов состоит в высокой нагрузочной способности: втекающий и вытекающий токи могут достигать 25 мА. Это упрощает схемы внешних устройств, за счет чего уменьшается общая стоимость разрабатываемых систем.

Микроконтроллер способен работать в широком диапазоне тактовых частот – от 0 до 20 МГц, широком диапазоне питающих напряжений – от 2 до 5,5 В, в широком температурном диапазоне – от –40 до +125 °С. Он отличается также низким энергопотреблением: менее 2 мА при напряжении питания 5 В и тактовой частоте 4 МГц, 15 мкА при питании 3 В и тактовой частоте 32 кГц.

Разработки на базе PIC-контроллеров поддерживаются ассемблером, программным симулятором, внутрисхемным эмулятором и программатором.

5.2. Структурная организация

Укороченная структурная схема PIC16F628 показана на рис. 5.1. В ней отражены функциональные блоки, составляющие ядро микроконтроллера, а также порты ввода/вывода PORTA и PORTB, счетчик-таймер TMR0 и некоторые блоки, повышающие надежность работы в реальных условиях: схема сброса по включению питания POR, таймер включения питания PWRT, таймер запуска генератора OST и сторожевой таймер WDT.

Назначение выводов микроконтроллера PIC16F628 отражено в табл. 5.1.

Архитектура микроконтроллера основана на концепции раздельных шин и областей памяти для данных и для программ (гарвардская архитектура). Это увеличивает скорость обмена по сравнению с традиционной принстонской архитектурой, в которой команды и данные передаются по одной и той же шине. Разделение шин команд и данных позволяет увеличить разрядность команды по сравнению с разрядностью данных. Шина данных и память данных (ОЗУ) имеют ширину 8 битов, а программная шина и программная память (ПЗУ) – 14 биты. Такая концепция обеспечивает простую, но мощную систему однословных команд, разработанную так, что битовые, байтовые и регистровые операции работают с высокой скоростью и с перекрытием по времени выборок команд и циклов выполнения. 14-битная ширина программной памяти обеспечивает выборку 14-битной команды за один цикл. Двухступенчатый конвейер обеспечивает одновременную выборку следующей и исполнение текущей команд. Все команды выполняются за один машинный цикл, исключая команды переходов, которые выполняются за два цикла. В PIC16F628 программная память расположена внутри кристалла. Исполняемая программа может находиться только во внутреннем ПЗУ.

Регистры PIC16F628 разделяются на две функциональные группы: специальные и общего назначения. Специальные регистры используются для управления режимами работы функциональных блоков контроллера, регистры общего назначения (память данных или ОЗУ) – для хранения переменных.

Микроконтроллер PIC16F628 использует прямую и косвенную адресацию всех регистров и ячеек памяти. Все специальные регистры и счетчик команд также адресуются как память данных. Ортогональная (симметричная) система команд позволяет выполнять любую операцию с любым регистром, используя любой из названных выше методов адресации. Это облегчает программирование для PIC16F628 и значительно уменьшает время, необходимое на обучение работе с микроконтроллером.

В микроконтроллере PIC16F628 имеется 8-разрядное арифметико-логическое устройство (АЛУ) и рабочий регистр (аккумулятор) W. АЛУ выполняет сложение, вычитание, сдвиг, битовые и логические операции. В командах, обрабатывающих два операнда, один из операндов содержится в рабочем регистре W. Второй операнд может быть константой или содержимым любого регистра ОЗУ. В командах с одним операндом операнд может быть содержимым рабочего регистра или любого регистра ОЗУ. Для выполнения всех операций АЛУ используется рабочий регистр W, который не может быть прямо адресован. Результат операции в АЛУ помещается либо в рабочий регистр W, либо в регистр ОЗУ.

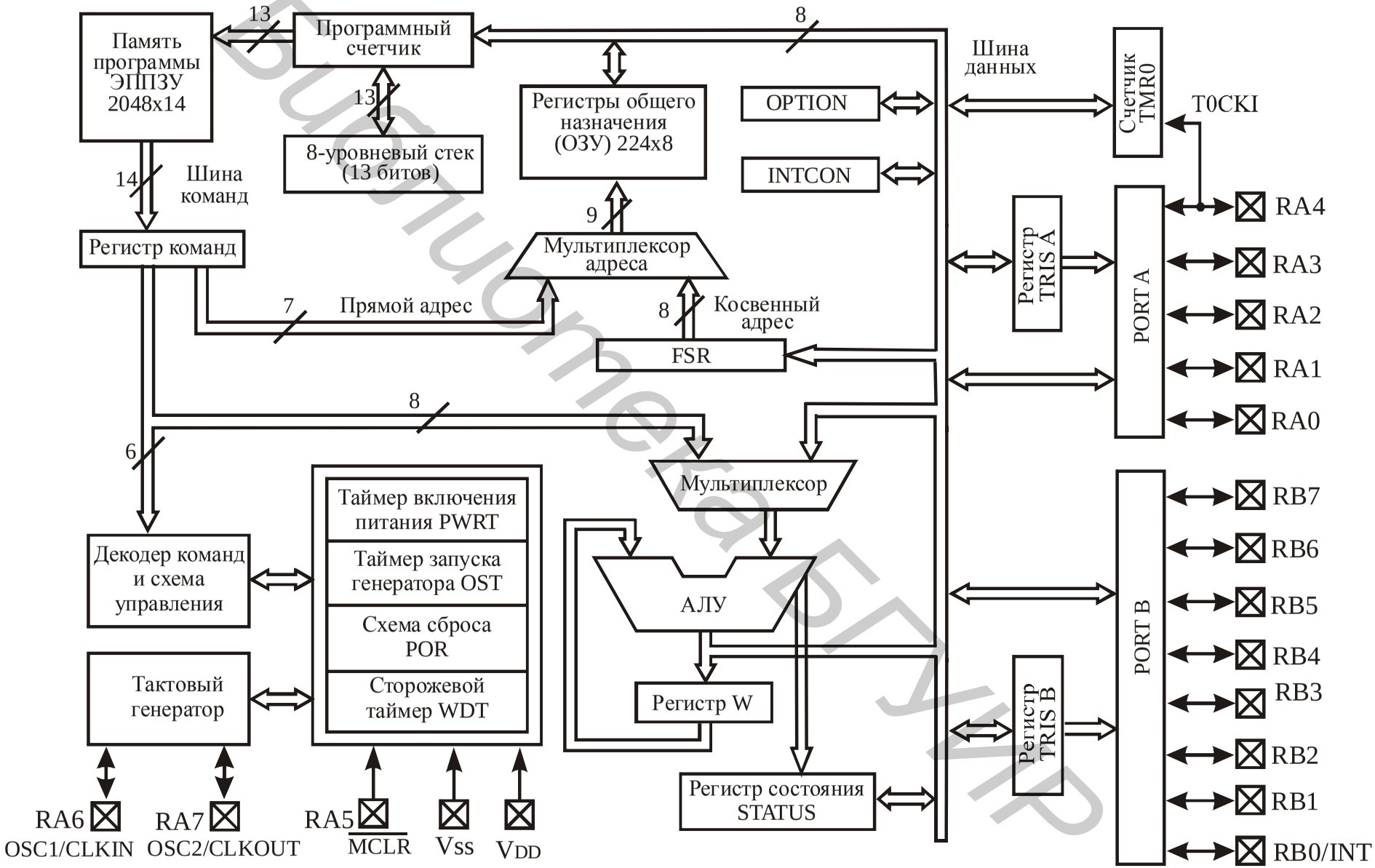


Рис. 5.1. Укороченная структурная схема PIC16F628

В зависимости от результата выполнения операции изменяются значения битов переноса (C), десятичного переноса (DC) и нулевого результата (Z) в регистре состояния STATUS. При вычитании биты C и DC работают как биты заема и десятичного заема соответственно.

Таблица 5.1

Обозначение вывода	*№ вывода DIP*	*Тип 1/O/P*	*Тип буфера в режиме ввода*	*Описание*
1	*2*	*3*	*4*	*5*
RA0/AN0	17	I/O	ST	Двунаправленный порт ввода/вывода, аналоговый вход компаратора
RA1/AN1	18	I/O	ST	Двунаправленный порт ввода/вывода, аналоговый вход компаратора
RA2/AN2/ V_{REF}	1	I/O	ST	Двунаправленный порт ввода/вывода, аналоговый вход компаратора, выход источника опорного напряжения V_{REF}
RA3/AN3/ CPM1	2	I/O	ST	Двунаправленный порт ввода/вывода, аналоговый вход компаратора, выход компаратора
RA4/T0CKI/ CPM2	3	I/O	ST	Двунаправленный порт ввода/вывода (выход с открытым стоком), вход внешнего тактового сигнала для TMR0, выход компаратора
RA5/–MCLR/ THV	4	I	ST	Вход сброса микроконтроллера, вход напряжения программирования. Когда вывод настроен как –MCLR, то по низкому уровню сигнала производится сброс микроконтроллера. При нормальной работе напряжение на выводе не должно превышать V_{DD}
RA6/OSC2/ CLKOUT	15	I/O	ST	Двунаправленный порт ввода/вывода, выход генератора для подключения резонатора. В режиме ER-генератора на выводе формируется сигнал с частотой ¼ OSC1, обозначая циклы команд
RA7/OSC1/ CLKIN	16	I/O	ST	Двунаправленный порт ввода/вывода, вход генератора для подключения резонатора, вход внешнего тактового сигнала, вывод ER-смещения
RB0/INT	6	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора, вход внешнего прерывания
RB1/RX/DT	7	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора, вход приемника USART, линия данных в синхронном режиме USART
RB2/TX/CK	8	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора, выход передатчика USART, линия тактового сигнала в синхронном режиме USART

Окончание табл. 5.1

1	*2*	*3*	*4*	*5*
RB3/CCP1	9	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора, вывод модуля CCP
RB4/PGM	10	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора. Изменение сигнала на входе может вывести микроконтроллер из режима SLEEP
RB5	11	I/O	TTL	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора. Изменение сигнала на входе может вывести микроконтроллер из режима SLEEP
RB6/T1OSO/ T1CKI	12	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора. Изменение сигнала на входе может вывести микроконтроллер из режима SLEEP. Выход генератора таймера 1
RB7/T1OSI	13	I/O	TTL/ST	Двунаправленный порт ввода/вывода с программным включением подтягивающего резистора. Изменение сигнала на входе может вывести микроконтроллер из режима SLEEP. Вход генератора таймера 1
V_{SS}	5	P	-	Общий вывод для внутренней логики и портов ввода/вывода
V_{DD}	14	P	-	Положительное напряжение питания для внутренней логики и портов ввода/вывода

Обозначения: I – вход, O – выход, I/O – вход/выход, P – питание,
TTL – входной буфер ТТЛ, ST – вход с триггером Шмитта

Входная тактовая частота, поступающая с вывода OSC1/CLKIN, внутри делится на четыре, и из нее формируются четыре циклические неперекрывающиеся тактовые последовательности Q1, Q2, Q3 и Q4. Счетчик команд увеличивается в такте Q1, команда считывается из памяти программы и защелкивается в регистре команд в такте Q4. Команда декодируется и выполняется в течение последующего цикла в тактах Q1...Q4. Временные диаграммы тактирования и выполнения команд изображены на рис. 5.2.

Цикл выполнения команды состоит из четырех тактов: Q1...Q4. Выборка команды и ее выполнение совмещены по времени таким образом, что выборка команды занимает один цикл, а выполнение – следующий цикл. Эффективное время выполнения команды составляет один цикл. Если команда изменяет счетчик команд (например команда CALL), то для выполнения этой команды потребуется два цикла. Цикл выборки начинается с увеличения счетчика команд в такте Q1. В цикле выполнения команды выбранная команда

защелкивается в регистр команд в такте Q1. В течение тактов Q2, Q3 и Q4 происходит декодирование и выполнение команды. В такте Q2 считывается память данных (чтение операнда), а запись происходит в такте Q4.

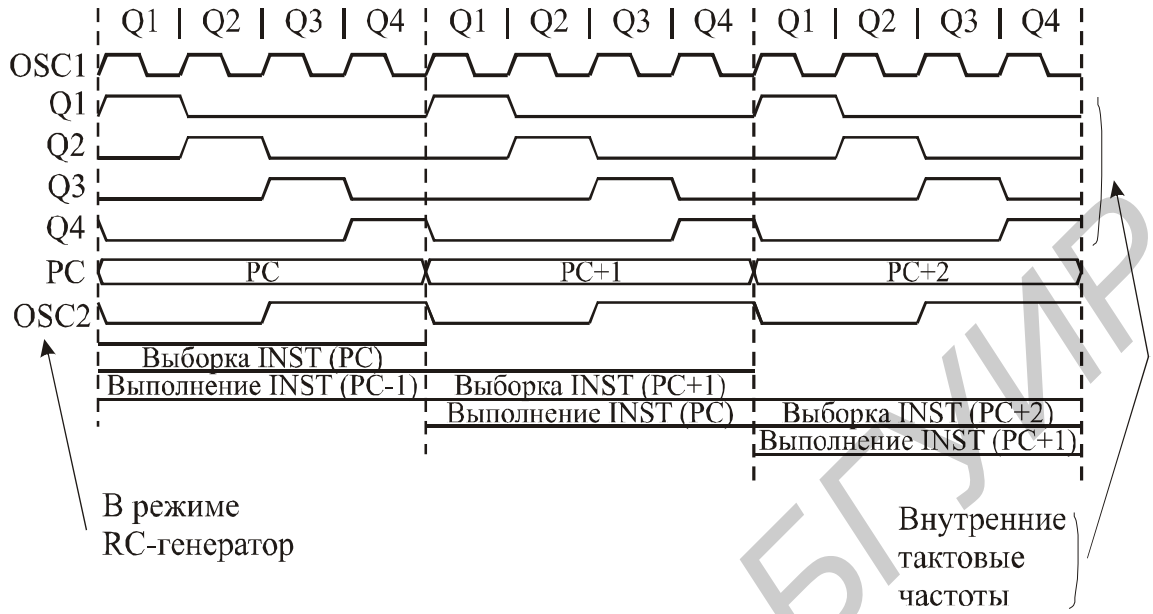


Рис. 5.2. Временные диаграммы тактирования и выполнения команд

5.3. Организация памяти

Внутренняя память в микроконтроллере PIC16F628 состоит из двух частей: памяти программы и памяти данных. Память программы и память данных имеют раздельные шины, поэтому доступ к ним может происходить одновременно. Память данных делится на регистры общего назначения (ОЗУ) и специальные регистры.

5.3.1. Организация памяти программы

Микроконтроллер PIC16F628 имеет 13-разрядный счетчик команд, способный адресовать до 8Кх14 слов памяти программы. В PIC16F628 присутствуют только первые 2Кх14 (0000-07FFh) слов памяти программы, остальные адреса зарезервированы для будущих модификаций. При сбросе процессор запускается с адреса 0000h, вектор прерывания расположен по адресу 0004h.

Счетчик команд PC указывает адрес выполняемой инструкции (команды). Младший байт счетчика команд PCL доступен для чтения и записи. Старший байт PCH, содержащий <12:8> биты счетчика команд PC, не доступен для чтения и записи. Все операции с регистром PCH происходят через дополнительный регистр PCLATH. При любом виде сброса микроконтроллера счетчик команд PC очищается.

На рис. 5.3 показаны две ситуации загрузки значения в счетчик команд PC. В примере сверху запись в счетчик команд PC происходит при записи значения в регистр PCL (PCLATH <4:0> —> PCH). В примере снизу запись значения в счетчик команд PC происходит при выполнении команды CALL или GOTO (PCLATH <4:3> —>PCH).

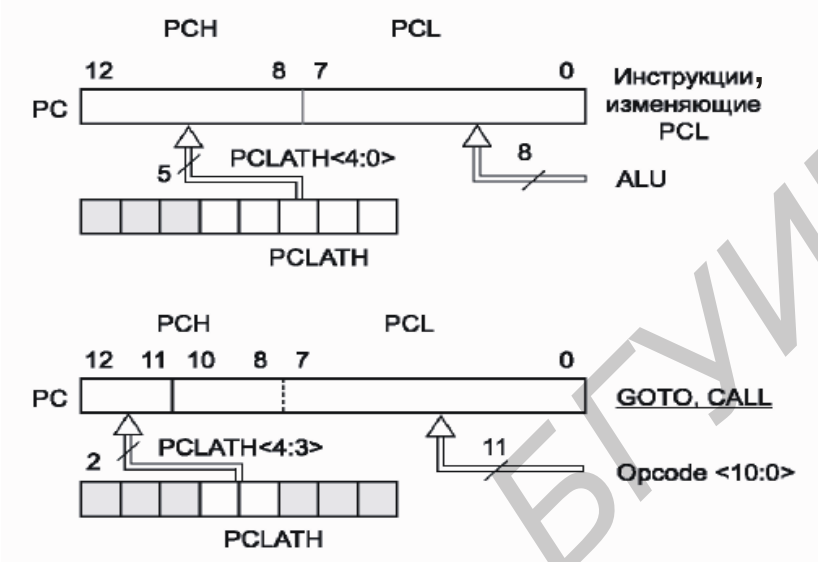


Рис. 5.3. Запись значения в счетчик команд PC

Вычисляемый переход может быть выполнен командой приращения к регистру PCL (например ADDWF PCL). При выполнении табличного чтения вычисляемым переходом следует заботиться о том, чтобы значение PCL не пересекло границу блока памяти (каждый блок – 256 байтов).

PIC16F628 имеет 8-уровневый 13-разрядный аппаратный стек. Стек не имеет отображения на память программ и память данных, нельзя записать или прочитать данные из стека. Значение счетчика команд заносится в вершину стека при выполнении инструкций перехода на подпрограмму (CALL) или обработки прерываний. Чтение из стека и запись в счетчик команд PC происходит при выполнении инструкций возвращения из подпрограммы или обработки прерываний (RETURN, RETLW, RETFIE), при этом значение регистра PCLATH не изменяется.

Стек работает как циклический буфер. После восьми записей в стек девятая запишется на место первой, а десятая заменит вторую и т.д.

В микроконтроллере не имеется никаких указателей о переполнении стека.

5.3.2. Организация памяти данных

Память данных разделяется на две области. Первая представляет собой регистры специальных функций, вторая – регистры общего назначения (ОЗУ). Специальные регистры включают в себя регистр таймера/счетчика (TMR0), счетчик команд (PC), регистр состояния (STATUS), регистры ввода/вывода (PORTA и PORTB), регистр косвенной адресации (FSR) и др. Специальные регистры TRISA и TRISB управляют конфигурацией (направлением передачи) портов ввода/вывода, а OPTION – режимами работы предварительного делителя.

Регистры общего назначения используются для хранения переменных по усмотрению пользователя.

Память данных разделена на четыре банка, которые содержат регистры общего и специального назначения.

Биты RP1 (STATUS<6>) и RP0 (STATUS<5>) предназначены для управления банками данных.

В табл. 5.2 приведена организация памяти данных PIC16F628.

Все регистры могут быть адресованы прямо или косвенно с использованием регистра косвенной адресации FSR. Непосредственная адресация поддерживается специальными командами, загружающими данные из памяти программы в рабочий регистр W.

5.4. Система команд

Каждая команда микроконтроллера PIC16F87X состоит из одного 14-разрядного слова, содержащего код операции (OPCODE), определяющий тип команды и один или несколько операндов, указывающих операцию команды. Описание полей команд дается в табл. 5.3. Полный список команд приведен в табл. 5.4. Команды разделены на следующие группы: байт-ориентированные команды, бит-ориентированные команды, команды управления и операций с константами.

Для байт-ориентированных команд f является указателем регистра, а d – указателем адреса результата. Указатель регистра определяет, какой регистр должен использоваться в команде. Указатель адресата определяет, где будет сохранен результат. Если d = 0, результат сохраняется в регистре W. Если d = 1, результат сохраняется в регистре, который используется в команде.

В бит-ориентированных командах b определяет номер бита, участвующего в операции, а f – указатель регистра, который содержит этот бит.

Таблица 5.2

Регистр	*Адрес*	*Регистр*	*Адрес*	*Регистр*	*Адрес*	*Регистр*	*Адрес*
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION REG	81h	TMR0	101h	OPTION REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
	07h		87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
	0Dh		8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh		8Fh		10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
	13h		93h		113h		193h
	14h		94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1 H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah	EEDATA	9Ah		11Ah		19Ah
	1Bh	EEADR	9Bh		11Bh		19Bh
	1Ch	EECON1	9Ch		11Ch		19Ch
	1Dh	EECON2	9Dh		11Dh		19Dh
	1Eh		9Eh		11Eh		19Eh
CMCON	1Fh	VRCON	9Fh		11Fh		19Fh
Регистры общего назначения 96 байт	20h	Регистры общего назначения 80 байт	A0h	Регистры общего назначения 48 байт	120h		1A0h
					14Fh		
					150h		
			EFh		16Fh		1EFh
	70h	Доступ к 70h-7Fh	F0h	Доступ к 70h-7Fh	170h	Доступ к 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Банк 0		***Банк 1***		***Банк 2***		***Банк 3***	

Примечание. Регистр косвенной адресации INDF – не физический. Обращение к нему по содержимому FSR= 00h дает нулевой результат.

Таблица 5.3

Поле	**Описание**
f	Адрес регистра (от 0x00 до Ox7F)
w	Рабочий регистр (аккумулятор)
b	Номер бита в 8-разрядном регистре
k	Константа (данные или метка)
d	Указатель адресата результата операции: d = 0 – результат сохраняется в регистре w, d = 1 – результат сохраняется в регистре f , По умолчанию d = 1
C	Флаг переполнения или заема
Z	Флаг нулевого результата
DC	Флаг десятичного переноса (из 3-го в 4-й разряд)
PC	Счетчик команд
GIE	Бит глобального разрешения прерываний
WDT	Сторожевой таймер
-TO	Флаг переполнения WDT
-PD	Флаг сброса по включению питания

В командах управления или операциях с константами k представляет восемь или одиннадцать битов константных значений или значения литералов.

Таблица 5.4

Мнемоника команды	*Описание*	*Кол-во циклов*	*Изм. флаги*	*Прим.*
1	*2*	*3*	*4*	*5*
Байт-ориентированные команды				
ADDWF f,d	Сложение W и f	1	C, DC, Z	1,2
ANDWF f,d	Побитное 'И' W и f	1	Z	1,2
CLRF f	Очистить f	1	Z	2
CLRW	Очистить W	1	Z	
COMF f,d	Инвертировать f	1	Z	1,2
DECF f,d	Вычесть 1 из f	1	Z	1,2
DECFSZ f,d	Вычесть 1 из f и пропустить следующую команду, если результат 0	1(2)		1,2,3
INCF f,d	Прибавить 1 к f	1	Z	1,2
INCFSZ f,d	Прибавить 1 к f и пропустить следующую команду, если результат 0	1(2)		1,2,3
IORWF f,d	Побитное 'ИЛИ' W и f	1	Z	1,2
MOVF f,d	Переслать f	1	Z	1,2
MOVWF f	Переслать W в f	1		
NOP	Нет операции	1		
RLF f,d	Циклический сдвиг f влево через перенос	1	C	1,2
RRF f,d	Циклический сдвиг f вправо через перенос	1	C	1,2
SUBWF f,d	Вычесть W из f	1	C, DC, Z	1,2
SWAPF f,d	Поменять местами полубайты в регистре f	1		1,2
XORWF f,d	Побитное 'исключающее ИЛИ' W и f	1	Z	1,2
Бит-ориентированные команды				
BCF f,b	Очистить бит b в регистре f	1		1,2
BSF f,b	Установить бит b в регистре f	1		1,2

Окончание табл. 5.4

1	*2*	*3*	*4*	*5*
BTFSC f,b	Проверить бит b в регистре f, пропустить следующую команду, если результат 0	1(2)		3
BTFSS f,b	Проверить бит b в регистре f, пропустить следующую команду, если результат 1	1(2)		3
Команды управления и операций с константами				
ADDLW k	Сложить константу с W	1	C, DC, Z	
ANDLW k	Побитное 'И' константы и W	1	Z	
CALL k	Вызов подпрограммы	2		
CLRWDT	Очистить WDT	1	-TO.-PD	
GOTO k	Безусловный переход	2		
IORLW k	Побитное 'ИЛИ' константы и W	1	Z	
MOVLW k	Переслать константу в W	1		
RETFIE	Возврат из подпрограммы с разрешением прерываний	2	GIE	
RETLW k	Возврат из подпрограммы с загрузкой константы k в W	2		
RETURN	Возврат из подпрограммы	2		
SLEEP	Перейти в режим SLEEP	1	-TO, -PD	
SUBLW k	Вычесть W из константы k	1	C, DC, Z	
XORLW k	Побитное 'исключающее ИЛИ' константы и W	1	Z	

Примечания:

1. При выполнении операции «чтение – модификация – запись» с портом ввода/вывода исходные значения считываются с выводов порта, а не из выходных защелок. Например, если в выходной защелке была записана '1', а на соответствующем выходе низкий уровень сигнала, то обратно будет записано значение '0'.

2. При выполнении записи в TMR0 (и d=1) предделитель TMR0 сбрасывается, если он подключен к модулю TMR0.

3. Если условие истинно или изменяется значение счетчика команд PC, то инструкция выполняется за два цикла. Во втором цикле выполняется команда NOP.

Все команды выполняются за один машинный цикл, кроме команд условия, в которых получен истинный результат, и инструкций, изменяющих значение счетчика команд PC. В случае выполнения команды за два машинных цикла во втором цикле выполняется инструкция NOP. Один машинный цикл состоит из четырех тактов генератора. Для тактового генератора с частотой 4 МГц все команды выполняются за 1мкс; если условие истинно или изменяется счетчик команд PC, команда выполняется за 2 мкс.

5.5. Регистр состояния STATUS

Регистр STATUS доступен по адресам 03h, 83h, 103h или 183h.

Структура регистра отражена в табл. 5.5. В регистре STATUS содержатся флаги состояния АЛУ, флаги причины сброса микроконтроллера и биты управления банками памяти данных.

Таблица 5.5

Номер бита	*Имя бита*	*Доступ и состояние после сброса*	*Назначение*
7	IRP	R/W-0	Выбор банка при косвенной адресации: I банк 2, 3(100h-1FFh) 0 банк 0, 1 (000h - 0FFh)
6, 5	RP1, RP0	R/W-0	Выбор банка при непосредственной адресации: 11 банк 3 (180h-1FFh) 10 банк 2 (100h-17Fh) 01 банк 1 (080h - 0FFh) 00 банк 0 (000h - 07Fh)
4	–TO	R-1	Флаг переполнения сторожевого таймера: 1 после POR или выполнения команд CLRWDT, SLEEP, 0 после переполнения WDT
3	–PD	R-1	Флаг включения питания: 1 после POR или выполнения команды CLRWDT, 0 после выполнения команды SLEEP
2	Z	R/W-x	Флаг нулевого результата: 1 нулевой результат выполнения арифметической или логической операции, 0 ненулевой результат выполнения арифметической или логической операции
1	DC	R/W-x	Флаг десятичного переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL), заем имеет инверсное значение: 1 был перенос/заем из младшего полубайта, 0 не было переноса/заема из младшего полубайта
0	C	R/W-x	Флаг переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL): 1 был перенос/заем из старшего бита, 0 не было переноса/заема из старшего бита. Заем имеет инверсное значение. Вычитание выполняется путем прибавления дополнительного кода второго операнда. При выполнении команд сдвига (RRF, RLF) бит C загружается старшим или младшим битом сдвигаемого регистра

Регистр STATUS может быть адресован любой командой, как и любой другой регистр памяти данных. Если обращение к регистру STATUS выполняется командой, которая воздействует на флаги Z, DC и C, то изменение этих трех битов командой заблокировано. Эти биты сбрасываются или устанавливаются согласно логике ядра микроконтроллера. Команды изменения регистра STATUS также не воздействуют на биты –TO и –PD. Поэтому результат выполнения команды с регистром STATUS может отличаться от ожидаемого. Например, команда CLRF STATUS сбросит три старших бита и

установит бит Z (состояние регистра STATUS после выполнения команды *000uu1uu*, где *u* – неизменяемый бит).

При изменении битов регистра STATUS рекомендуется использовать команды, не влияющие на флаги АЛУ (SWAPF, MOVWF, BCF и BSF).

Флаги C и DC используются как биты заема и десятичного заема соответственно, например, при выполнении команд вычитания SUBLW и SUBWF.

5.6. Регистр OPTION_REG

Регистр OPTION_REG доступен для чтения и записи, содержит биты управления предварительным делителем TMR0/WDT, активным фронтом внешнего прерывания RB0/INT, подтягивающими резисторами на входах PORTB.

Если предварительный делитель включен перед WDT, то коэффициент деления тактового сигнала для TMR0 равен 1:1.

Регистр OPTION_REG доступен по адресам 81h и 181h.

Структура регистра отражена в табл. 5.6.

Таблица 5.6

Номер бита	*Имя бита*	*Доступ и состояние после сброса*	*Назначение*
1	2	3	4
7	–RBPU	R/W-1	Включение подтягивающих резисторов на входах PORTB: 1 подтягивающие резисторы отключены 0 подтягивающие резисторы включены
6	INTEDG	R/W-1	Выбор активного фронта сигнала на входе внешнего прерывания INT: 1 прерывания по фронту сигнала 0 прерывания по срезу сигнала
5	T0CS	R/W-1	Выбор тактового сигнала для TMR0: 1 внешний тактовый сигнал с вывода RA4/T0CKI 0 внутренний тактовый сигнал CLKOUT
4	T0SE	R/W-1	Выбор фронта приращения TMR0 при внешнем тактовом сигнале: 1 приращение по срезу сигнала (с высокого к низкому уровню) на выводе RA4/TOCKI 0 приращение по фронту сигнала (с низкого к высокому уровню) на выводе RA4/TOCKI
3	PSA	R/W-1	Выбор включения предделителя: 1 предделитель включен перед WDT 0 предделитель включен перед TMR0

Окончание табл. 5.6

1	2	3	4		
2 1 0	PS2 PS1 PS0	R/W-1	Установка коэффициента деления предделителя:		
				для TMR0	для WDT
			000	1:2	1:1
			001	1:4	1:2
			010	1:8	1:4
			011	1:16	1:8
			100	1:32	1:16
			101	1:64	1:32
			110	1:128	1:64
			111	1:256	1:128

5.7. Регистр INTCON

Регистр INTCON доступен для чтения и записи, содержит биты разрешений и флаги прерываний по переполнению TMR0, по изменению уровня сигнала на выводах PORTB и по внешнему источнику прерываний RB0/INT. Флаги прерываний устанавливаются при возникновении условий прерываний вне зависимости от соответствующих битов разрешения и бита общего разрешения прерываний GIE (INTCON<7>).

Регистр INTCON доступен по адресам 0Bh, 1Bh, 10Bh и 18Bh.

Структура регистра отражена в табл. 5.7.

Таблица 5.7

Номер бита	*Имя бита*	*Доступ и состояние после сброса*	*Назначение*
1	2	3	4
7	GIE	R/W-0	Глобальное разрешение прерываний: 1 разрешены все немаскированные прерывания, 0 все прерывания запрещены
6	PEIE	R/W-0	Разрешение прерываний от периферийных модулей: 1 разрешены все немаскированные прерывания периферийных модулей, 0 прерывания от периферийных модулей запрещены
5	T0IE	R/W-0	Разрешение прерывания по переполнению TMR0: 1 прерывание разрешено, 0 прерывание запрещено
4	INTE	R/W-0	Разрешение внешнего прерывания INT: 1 прерывание разрешено, 0 прерывание запрещено
3	RBIE	R/W-0	Разрешение прерывания по изменению сигнала на входах RB7:RB4 PORTB: 1 прерывание разрешено, 0 прерывание запрещено

Окончание табл. 5.7

1	2	3	4
2	T0IF	R/W-0	Флаг прерывания по переполнению TMR0: 1 произошло переполнение TMR0 (сбрасывается программно), 0 переполнения TMR0 не было
1	INTF	R/W-0	Флаг внешнего прерывания INT: 1 выполнено условие внешнего прерывания на выводе RB0/INT (сбрасывается программно), 0 внешнего прерывания не было
0	RBIF	R/W-x	Флаг прерывания по изменению уровня сигнала на входах RB7:RB4 PORTB: 1 зафиксировано изменение уровня сигнала на одном из входов (сбрасывается программно), 0 не было изменения уровня сигнала

5.8. Косвенная адресация данных

Для выполнения косвенной адресации необходимо обратиться к физически не реализованному регистру INDF. Обращение к регистру INDF фактически вызовет действие с регистром, адрес которого указан в FSR. Косвенное чтение регистра INDF (FSR=0) даст результат 00h. Косвенная запись в регистр INDF вызывает только воздействия на флаги АЛУ в регистре STATUS. 9 битов косвенного адреса IRP сохраняется в регистре STATUS<7>. Пример 9-разрядной косвенной адресации показан на рис. 5.4.

Приведем пример использования косвенной адресации для очистки памяти данных в диапазоне адресов 20h-2Fh.

```
    BCF     STATUS, IRP; Установить банк 0,1
    MOVLW   0x20        ; Указать первый регистр в ОЗУ
    MOVWF   FSR
NEXT:
    CLRF    INDF        ; Очистить регистр
    INCF    FSR, F      ; Увеличить адрес
    BTFSS   FSR, 4      ; Завершить?
    GOTO    NEXT        ; Нет, продолжить очистку
CONTINUE:               ; Да
```

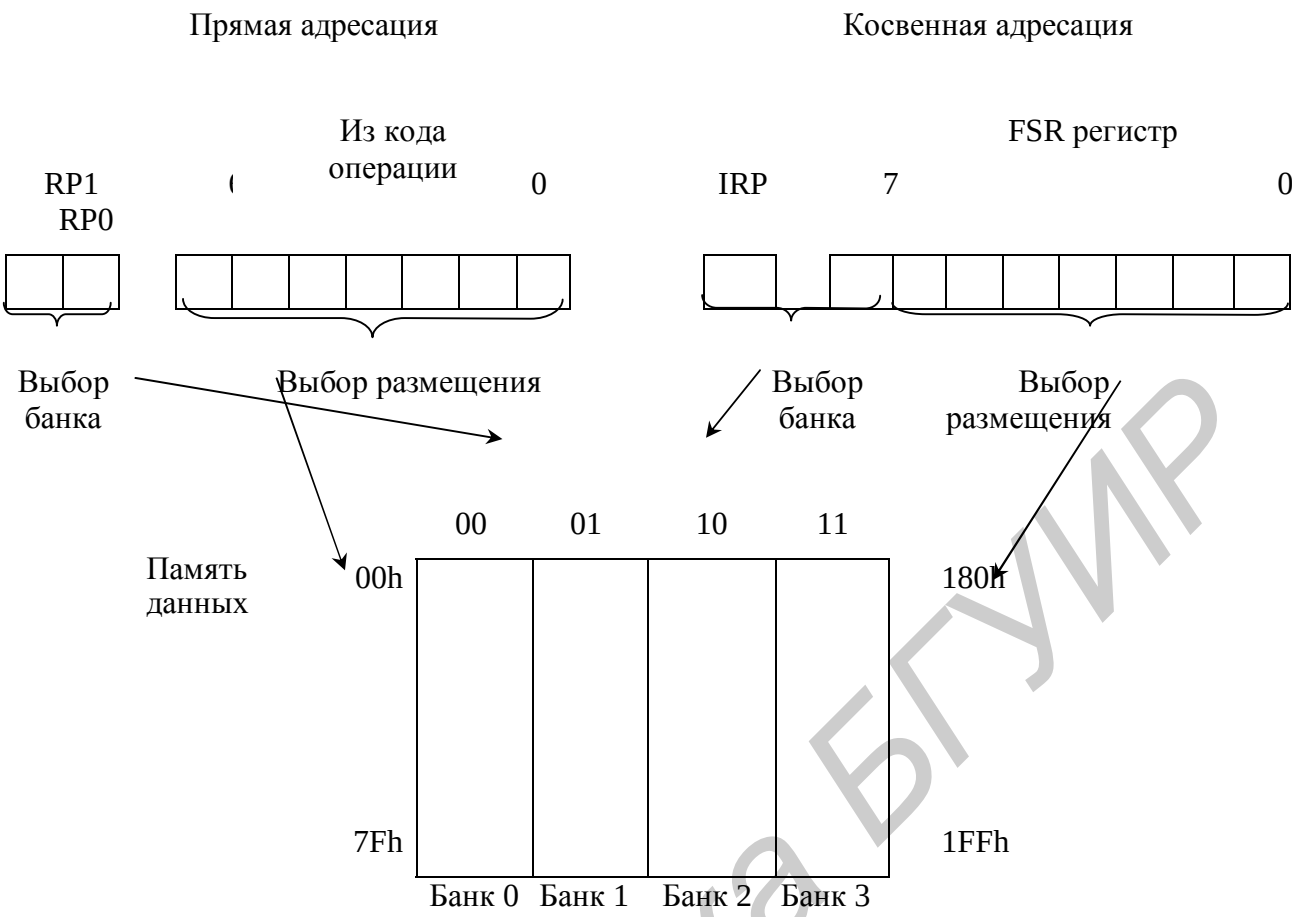


Рис. 5.4. Схема прямой и косвенной адресации в PIC16F628

5.9. Прерывания

PIC16F628 имеет 10 источников прерываний. Регистр INTCON содержит флаги отдельных прерываний, биты разрешения этих прерываний и бит глобального разрешения прерываний.

Если бит GIE (INTCON<7>) установлен в '1', то разрешены все немаскированные прерывания. Если GIE=0, то все прерывания запрещены. Каждое прерывание в отдельности может быть разрешено или запрещено установкой/сбросом соответствующего бита в регистрах INTCON и PIE1. При сбросе микроконтроллера бит GIE сбрасывается в '0'.

При возвращении из подпрограммы обработки прерывания по команде RETFIE бит GIE аппаратно устанавливается в '1', разрешая все немаскированные прерывания.

В регистре INTCON находятся флаги следующих прерываний: внешнего сигнала INT, изменения уровня сигнала на входах RB7:RB4, переполнения TMR0.

В регистре INTCON находится бит разрешения прерываний от периферийных модулей. Упрощенная структурная схема логики прерываний (без прерываний от периферийных модулей) показана на рис. 5.5.

При переходе на подпрограмму обработки прерываний бит GIE аппаратно сбрасывается в '0', запрещая прерывания, текущее значение счетчика команд (адрес возврата из подпрограммы обработки прерываний) помещается в стек, а в счетчик команд PC загружается вектор прерывания 0004h. Источник прерываний может быть определен проверкой флагов прерываний, которые должны быть сброшены программно перед разрешением прерываний, чтобы избежать повторного вызова.

Для внешних источников прерываний (сигнал INT, изменения уровня сигнала на входах RB7:RB4) время перехода на подпрограмму обработки прерываний будет составлять 3-4 машинных цикла. Точное время перехода зависит от конкретного случая, оно одинаково для одного и двух цикловых команд. Флаги прерываний устанавливаются независимо от состояния соответствующих битов маски и бита GIE.

Индивидуальные флаги прерываний устанавливаются независимо от состояния соответствующих битов маски и бита GIE.

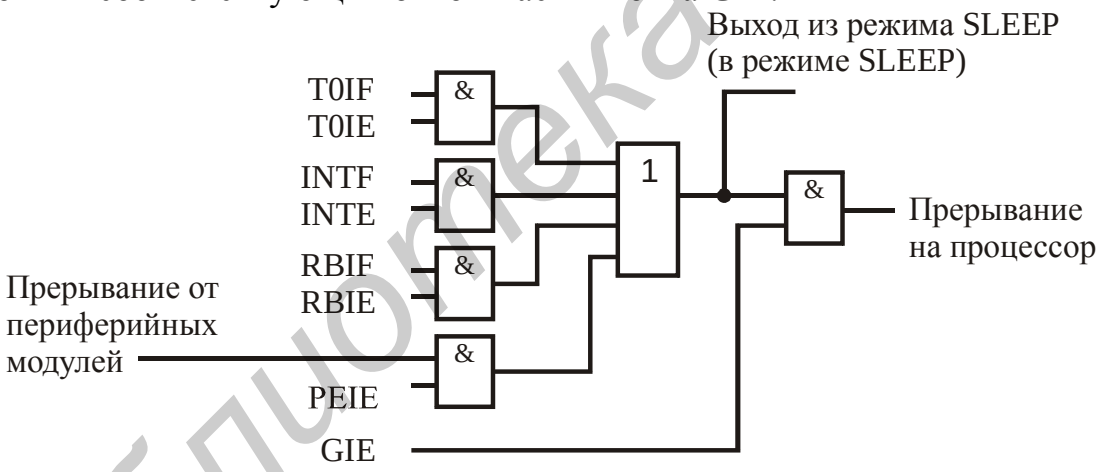


Рис. 5.5. Логика прерываний

5.9.1. Внешнее прерывание с входа RB0/INT

Внешнее прерывание с входа RB0/INT происходит следующим образом: по переднему фронту сигнала, если бит INTEDG (OPTION_REG<6>) установлен в '1'; по заднему фронту сигнала, если бит INTEDG сброшен в '0'. Когда активный фронт сигнала появляется на входе RB0/INT, бит INTF (INTCON<1>) устанавливается в '1'. Прерывание может быть запрещено сбросом бита INTE (INTCON<4>) в '0'. Флаг прерывания INTF должен быть сброшен программно в подпрограмме обработки прерываний. Прерывание INT

может вывести микроконтроллер из режима SLEEP, если бит INTE=1 до перехода в режим SLEEP. Состояние бита GIE определяет, переходить ли на подпрограмму обработки прерываний после выхода из режима SLEEP.

5.9.2. Прерывание по переполнению TMR0

Переполнение таймера TMR0 (FFh –> 00h) устанавливает флаг T0IF (INTCON<2>) в '1'. Прерывание от TMR0 можно разрешить/запретить установкой/сбросом бита T0IE (INTCON<5>).

5.9.3. Прерывание по изменению уровня сигнала на входах RB7:RB4

Изменение уровня сигнала на входах RB7:RB4 вызывает установку флага RBIF (INTCON<0>). Прерывание можно разрешить/запретить установкой/сбросом бита RBIE (INTCON<4>).

5.9.4. Сохранение контекста при обработке прерываний

При переходе на подпрограмму обработки прерываний в стеке сохраняется только адрес возврата. Как правило, необходимо сохранять значения ключевых регистров при обработке прерываний (например регистр W и STATUS), что выполняется программным способом.

Так как старшие 16 байтов каждого банка микроконтроллера PIC16F628 доступны во всех банках, то регистры STATUS_TEMP, PCLATH_TEMP и W_TEMP могут быть размещены в этой области. Ниже показан пример текста программы сохранения контекста.

Пример: Сохранение и восстановление регистров STATUS, W и PCLATH

```
MOVWF     W_TEMP          ;Сохранить W в регистре текущего банка
SWAPF     STATUS,W        ;Поменять местами полубайты и сохранить в W
CLRF      STATUS          ;Выбрать банк 0
MOVWF     STATUS_TEMP     ;Сохранить регистр STATUS
MOVF      PCLATH,W        ;
MOVWF     PCLATH_TEMP     ;Сохранить регистр PCLATH
…
…                         ;Код программы обработки прерываний
…
MOVF      PCLATH_TEMP,W   ;
MOVWF     PCLATH          ;Восстановить регистр PCLATH
SWAPF     STATUS_TEMP,W   ;Прочитать регистр STATUS_TEMP
                          ; в W, восстанавливая банк памяти программ
MOVWF     STATUS          ;Переписать W в регистр STATUS
SWAPF     W_TEMP,F        ;Поменять местами полубайты в W_TEMP
SWAPF      W_TEMP,W       ;Поменять местами полубайты в W_TEMP и
                           ;записать в W
```

В примере для пересылки содержимого регистра в рабочий регистр W используется команда SWAPF. Это единственный способ пересылки без искажения состояния регистра STATUS.

5.10. Порты ввода/вывода

Микроконтроллеры PIC16F628 имеют два порта ввода/вывода, PORTA и PORTB. Некоторые выводы портов мультиплексированы с периферийными модулями микроконтроллера. Когда периферийный модуль включен, вывод не может использоваться как универсальный канал ввода/вывода.

Программа может считывать и записывать данные в регистры ввода/вывода аналогично регистрам общего назначения. При чтении всегда считывается действительное состояние выводов независимо от того, запрограммированы отдельные биты как входы или как выходы. После сброса все разряды программируются как входы (выводы находятся в высокоимпедансном состоянии), поскольку регистры управления портами TRISA и TRISB устанавливаются в '1'. Разряд порта ввода-вывода определяется как выход, если соответствующий бит в регистре управления портом установлен в '0'.

5.10.1. PORTA

Регистр ввода-вывода PORTA имеет разрядность 8 битов. Разряд RA4 имеет вход с триггером Шмитта и выход с открытым стоком. Он объединен с входом таймера TMR0. Все остальные разряды PORTA имеют триггеры Шмитта на входах и выходные КМОП-буферы.

Описание выводов регистра PORTA приведено в табл. 5.1. Упрощенная функциональная организация разрядов RA0…RA3 регистра показана на рис. 5.6, а разряда RA4 – на рис. 5.7.

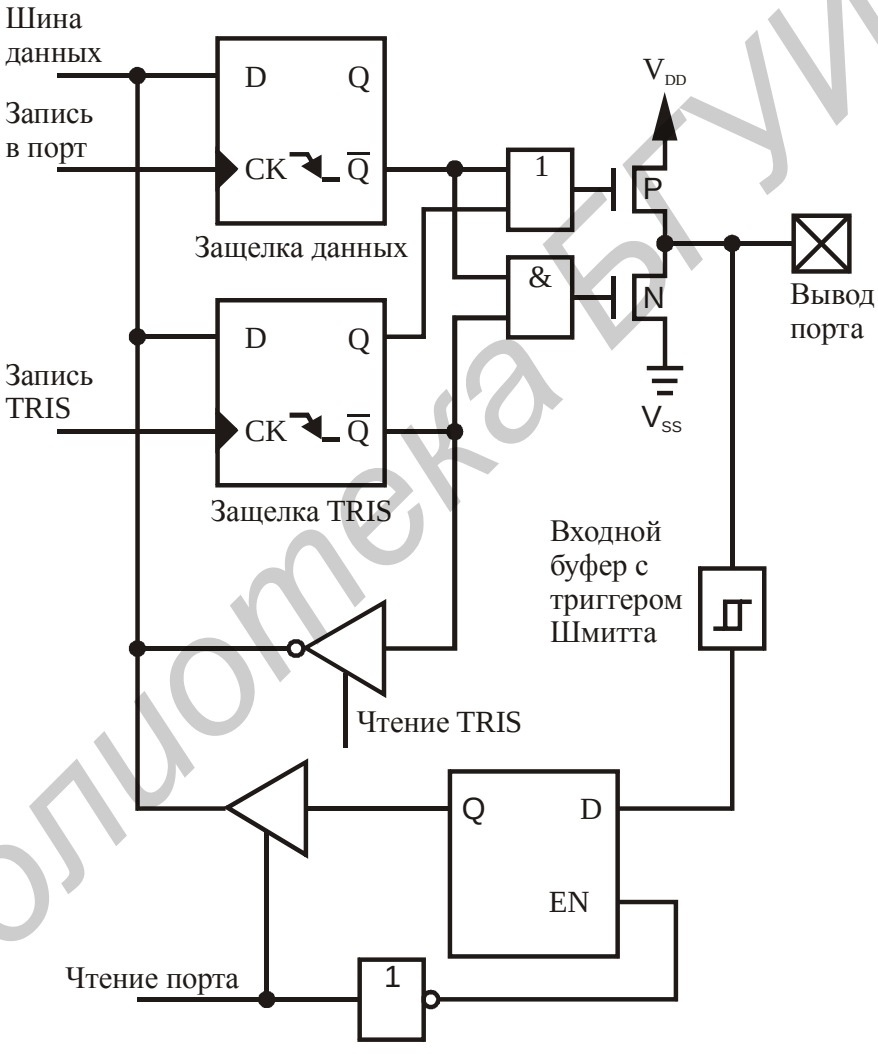
Ниже приведем пример программы инициализации (настройки) PORTA.

```
CLRF    PORTA           ;Очистка (сброс в «0») выходных защёлок PORTA.
BSF     STATUS,RP0      ;Выбор банка 1.
MOVLW   0x 0F           ;Значение константы для выбора режимов работы
                        ;разрядов.
MOVWF   TRISA           ;Установить RA<3:0> как входы и RA4 как
                        ;выход.
BCF     STATUS,RP0      ;Выбор банка 0.
```

5.10.2. PORTB

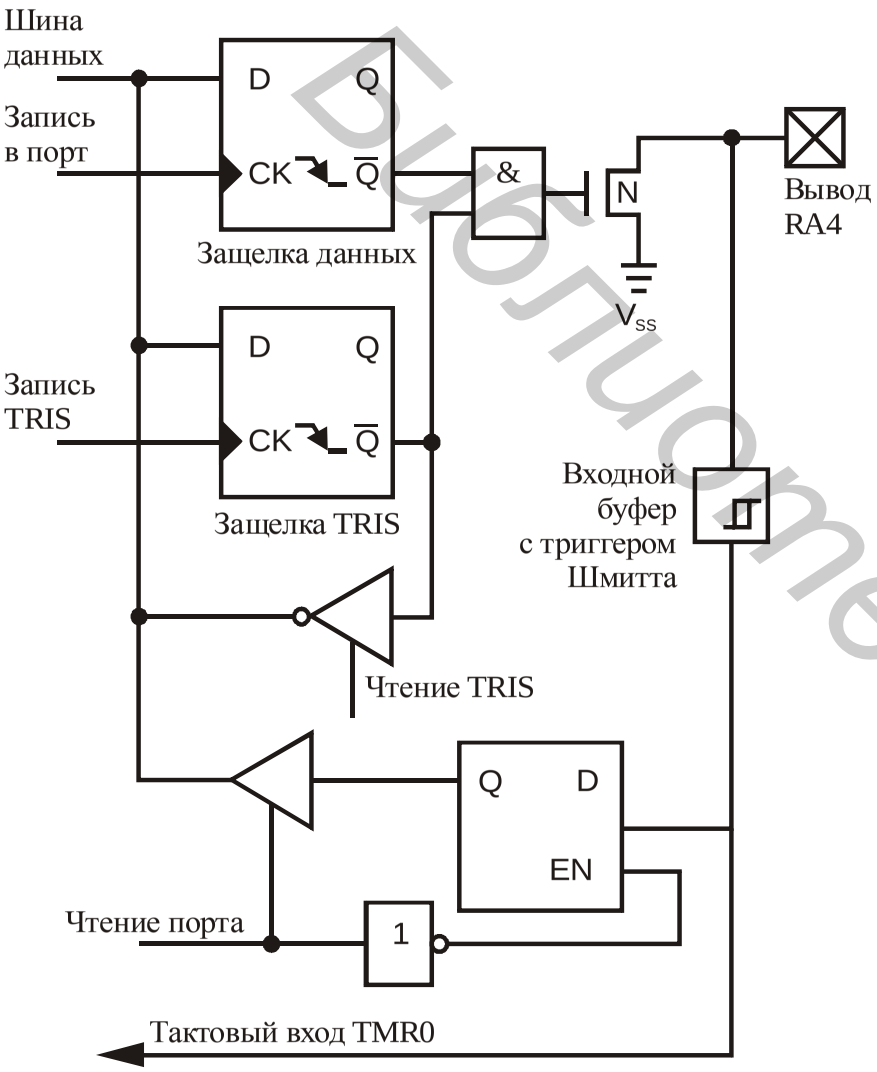
Регистр ввода/вывода PORTB 8-разрядный. Все разряды PORTB имеют внутренние подтягивающие резисторы, которые могут быть включены установкой в '0' бита RBPU (OPTION). Подтягивающие резисторы автоматически отключаются, если соответствующий разряд программируется как выход. По включении питания подтягивающие резисторы отключаются.

Упрощенная функциональная организация выводов RB7…RB4 PORTB показана на рис. 5.8, а выводов RB3…RB0 – на рис. 5.9.



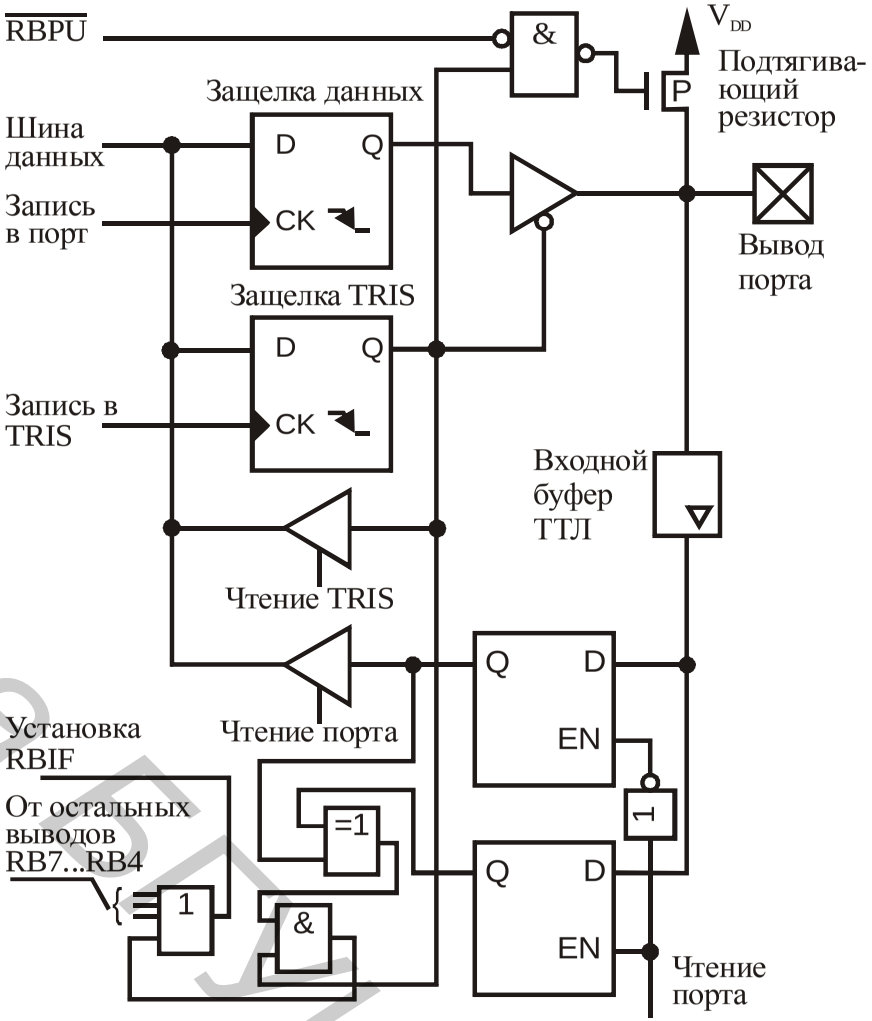
Примечание. Все выводы имеют защитные диоды на V_{DD} и V_{SS}.

Рис. 5.6. Функциональная организация выводов RA0...RA3 регистра PORTA



Примечание. Вывод имеет защитный диод только на V_{SS}.

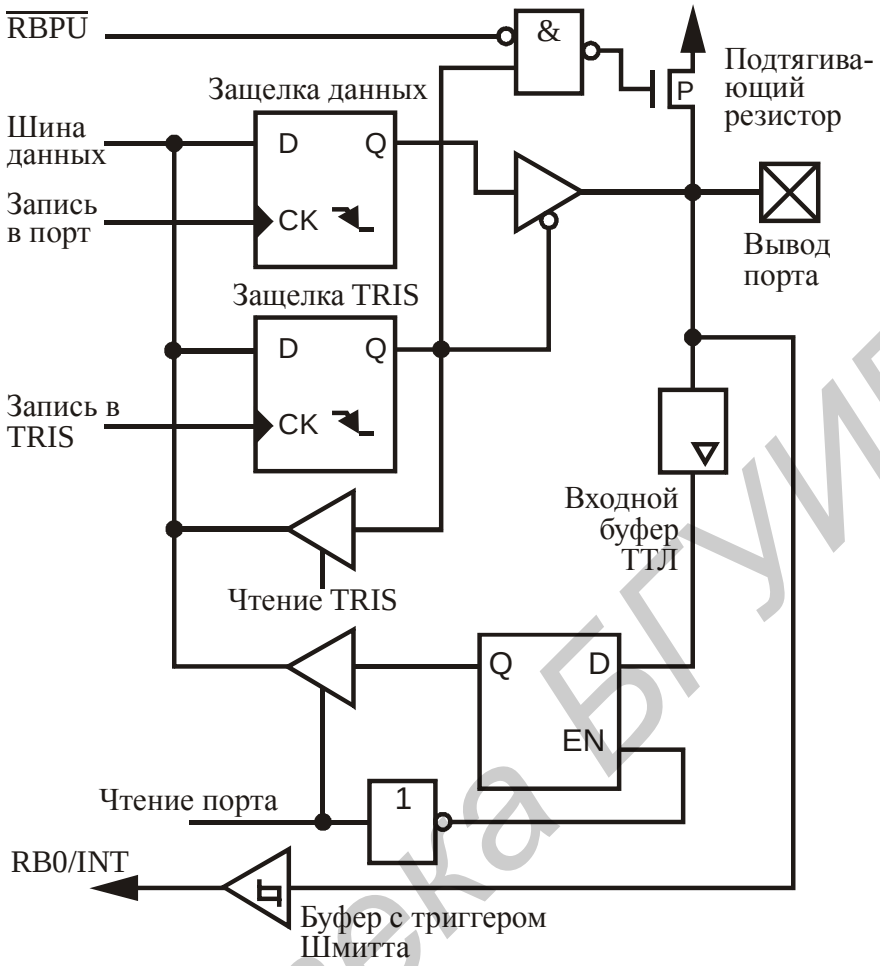
Рис. 5.7. Функциональная организация вывода RA4 регистра PORTA



Примечания:

1. Бит TRISB=’1’ разрешает подтягивающий резистор, если в регистре OPTION $\overline{RBPU}$=’0’.
2. Все выводы имеют защитные диоды на V_{DD} и V_{SS}.

Рис. 5.8. Функциональная организация выводов RB7...RB4



Примечания:
1. Бит TRISB = ’1’ разрешает подтягивающий резистор, если в регистре OPTION $\overline{\text{RBPU}}$ = ’0’.
2. Все выводы имеют защитные диоды на V_{DD} и V_{SS}.

Рис. 5.9. Функциональная организация выводов RB3...RB0

Имеется возможность прерывания по изменению состояния четырех разрядов PORTB (выводы RB7…RB4). Прерывание может возникнуть только от тех разрядов PORTB<7:4>, которые запрограммированы как входы; выходы не включаются в процедуру сравнения. Текущее состояние разрядов PORTB<7:4>, запрограммированных как входы, сравнивается с состоянием, защелкнутым в регистр PORTB при последнем считывании. При несовпадении возникает прерывание по изменению состояния. Это прерывание может вывести микроконтроллер из режима пониженного энергопотребления SLEEP. Для сброса прерывания в подпрограмме обработки необходимо выполнить следующие действия:

1) считать PORTB (это сбросит условие несовпадения);

2) сбросить флаг RBIF.

Прерывание по несовпадению совместно с программно-управляемыми подтягивающими резисторами позволяет легко реализовать интерфейс клавиатуры и обеспечить выход из режима пониженного энергопотребления по нажатию клавиши. Для возникновения прерывания по изменению состояния минимальная деятельность импульса должна быть не менее длительности цикла команды.

Ниже приведем пример инициализации (настройки) PORTB.

```
CLRF    PORTB           ;Обнуление выходных регистров PORTB.
BSF     STATUS,RP0      ;Выбор банка 1.
MOVLW   0xCF            ;Значение для задания направления.
MOVWF   TRISB           ;Установить RB<3:0> как входы, RB<5:4>
                                ; как выходы и RB<7:6> как входы
```

5.11. Особенности программирования портов

5.11.1. Организация двунаправленных портов

Некоторые команды выполняются в режиме «чтение – модификация – запись».

Например, команды BCF и BSF считывают содержимое порта целиком, модифицируют один бит и выводят результат обратно. При использовании этих команд с портами, в которых некоторые разряды запрограммированы как входы, а некоторые как выходы, необходима осторожность. Например, команда BSF для пятого бита регистра PORTB сначала считывает все восемь бит, затем выполняется установка пятого бита и новое значение байта целиком записывается в выходную защелку. Если другой бит регистра PORTB используется в качестве двунаправленного входа/выхода (скажем, бит 0) и в данный момент он определен как вход, входной сигнал на этом выводе будет считан и записан обратно в выходную защелку этого же вывода, затирая ее предыдущее состояние. До тех пор пока этот вывод остается в режиме входа, никаких проблем не возникает. Однако если позднее линия 0 переключится в режим выхода, ее состояние будет неопределенным. Команда считывания порта считывает состояние вывода, а не выходных регистров. Например, если разряд порта запрограммирован как выход и установлен в '1', но внешняя схема поддерживает низкий уровень на выводе, порт будет считываться как '0'. На вывод, работающий в режиме выхода, не должны подключаться внешние

нагрузки по схеме "монтажное И" либо "монтажное ИЛИ". Возникающие при этом большие токи могут повредить кристалл.

5.11.2. Обращение к портам ввода/вывода

Запись в порт вывода происходит в конце цикла выполнения команды. При чтении данные должны быть стабильны в начале цикла выполнения команды. Надо быть внимательным при операциях чтения, следующих сразу же за записью в тот же порт. Здесь надо учитывать инерционность установления напряжения на выводах. Может потребоваться программная задержка, чтобы напряжение на выводе успело стабилизироваться до начала исполнения следующей команды чтения. Время установления напряжения на выводе зависит от подключенной к нему нагрузки и может меняться в широких пределах.

Ниже рассмотрен пример выполнения операции «чтение – модификация – запись» с портом ввода/вывода.

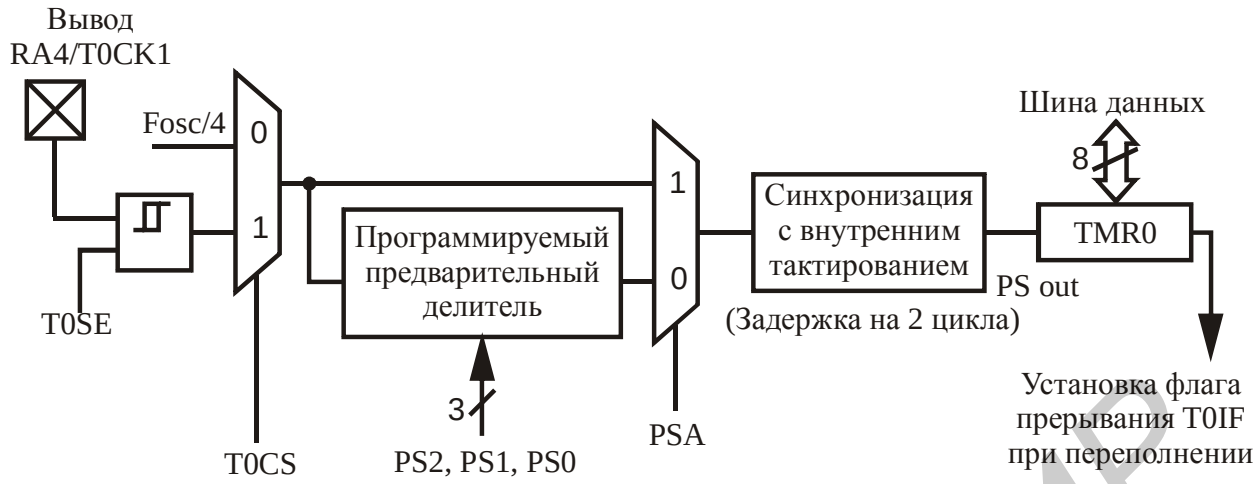
```
;Начальные установки порта: PORTB<7…4> – входы,
;                           PORTB<3…0> – выходы.
:Разряды PORTB<7…6> имеют внешние нагрузки.
;                                   Защелка PORTB   Выводы PORTB
        BCF     PORTB,7             ;01pp pppp          11pp pppp
        BCF     PORTB,6             ;10pp pppp          11pp pppp
        BSF     STATUS,RP0 ;
        BCF     TRISB,7             ;10pp pppp          11pp pppp
        BCF     TRISB,6             ;10pp pppp          10pp pppp
; Примечание. Пользователь мог бы ожидать, что результирующее значение будет
; 00pp pppp.
; Однако вторая команда BCF вызывает защелкивание в RB7 высокого уровня в
; соответствии с состоянием вывода RB7.
```

5.12. Модуль таймера TMR0

Модуль таймера TMR0 имеет следующие особенности:

- 8-разрядный таймер/счетчик, доступен по чтению и записи;
- 8-разрядный программируемый предварительный делитель;
- внутреннее или внешнее тактирование;
- прерывание по переполнению счетчика (переход от 0FFh к 00h);
- выбор фронта тактирующего импульса при внешнем тактировании.

Упрощенная структурная схема модуля таймера приведена на рис. 5.10.



Примечания.
1. Биты T0CS, T0SE, PSA, PS2, PS1 и PS0 находятся в регистре OPTION.
2. Предварительный делитель используется совместно со сторожевым таймером.

Рис. 5.10. Структурная схема таймера

Режим таймера выбирается установкой в '0' бита T0CS (OPTION<5>). В режиме таймера TMR0 увеличивается в каждом командном цикле (в отсутствии предварительного делителя). Если происходит запись в TMR0, то увеличение счетчика задерживается на два последующих цикла выполнения команды. Запись в TMR0 должна вестись с учетом этой задержки. При необходимости проверки регистра TMR0 на ноль без влияния на процесс счета рекомендуется пользоваться командой **MOVF** TMR0,W.

Режим счетчика выбирается установкой в '1' бита T0CS (OPTION<5>). В этом режиме TMR0 увеличивается по каждому перепаду 1/0 или 0/1 на выводе T0CKI. Перепад, увеличивающий значение TMR0, выбирается битом выбора фронта переключения T0SE (OPTION<4>). Установка этого бита в '0' вызывает увеличение TMR0 по перепаду 0/1.

Предварительный делитель может использоваться модулем сторожевого таймера WDT или модулем таймера. Подключение предварительного делителя задается битом PSA (OPTION<3>). Установка бита PSA в '1' подключает предварительный делитель к модулю WDT и устанавливает коэффициент деления для TMR0 1:1. Установка бита PSA в '0' подключает предварительный делитель к модулю таймера. Коэффициент деления предварительного делителя может быть установлен битами PS0-PS2 регистра OPTION_REG. Сам предварительный делитель недоступен для чтения и записи.

5.12.1. Прерывание от таймера

Прерывание от TMR0 вырабатывается при переполнении счетчика (переходе от 0FFh к 00h). При переполнении устанавливается в '1' бит T0IF (INTCON<2>). Прерывание может быть замаскировано установкой в '0' бита T0IE (INTCON<5>). Бит T0IF должен быть сброшен в '0' в процедуре обработки прерывания от TMR0 до того, как прерывания снова будут разрешены. Прерывание от TMR0 не может вывести микроконтроллер из режима пониженного энергопотребления SLEEP, поскольку в режиме SLEEP таймер TMR0 выключен.

5.12.2. Использование TMR0 с внешним сигналом

Если для тактирования TMR0 используется внешний сигнал, то он должен удовлетворять определенным требованиям для синхронизации с внутренней тактовой частотой. Кроме того, между перепадом на выводе T0CKI и реальным увеличением счетчика TMR0 есть некоторая задержка.

Если предварительный делитель не используется, внешний тактовый сигнал на входе T0CKI должен сохранять как высокий, так и низкий уровень в течение не менее двух периодов тактового генератора.

Когда используется предварительный делитель, входной сигнал TMR0 делится асинхронным счетчиком предварительного делителя, поэтому выходной сигнал делителя является симметричным. Период сигнала на входе TMR0 должен быть не менее четырех периодов тактового генератора. Сигнал же на входе T0CKI должен иметь высокие и низкие уровни длительностью не менее 10 нс каждый.

Так как выход предварительного делителя синхронизирован с внутренней тактовой частотой, то возможна небольшая задержка между перепадом сигнала на выводе T0CKI и моментом увеличения содержимого TMR0.

5.12.3. Предварительный делитель

Встроенный 8-разрядный счетчик может использоваться как предварительный делитель для TMR0 или как дополнительный делитель для сторожевого таймера WDT. Необходимо учесть, что делитель может быть использован либо с TMR0, либо со сторожевым таймером WDT, но не одновременно.

Схема использования предварительного делителя отражена на рис. 1.11.

Биты PSA и PS0-PS2 в регистре OPTION<3:0> задают режим использования предварительного делителя и его коэффициент деления.

Когда предварительный делитель используется с TMR0, все команды, производящие запись в регистр TMR0 (например **CLRF** TMR0, **MOVWF** TMR0, **BSF** TMR0,b и т.д.), обнуляют предварительный делитель.

Когда предварительный делитель используется со сторожевым таймером WDT, команда **CLRWDT** очищает предварительный делитель одновременно со сбросом сторожевого таймера WDT. Предварительный делитель не может быть считан или записан программно. По сбросу предварительный делитель содержит все '0'.

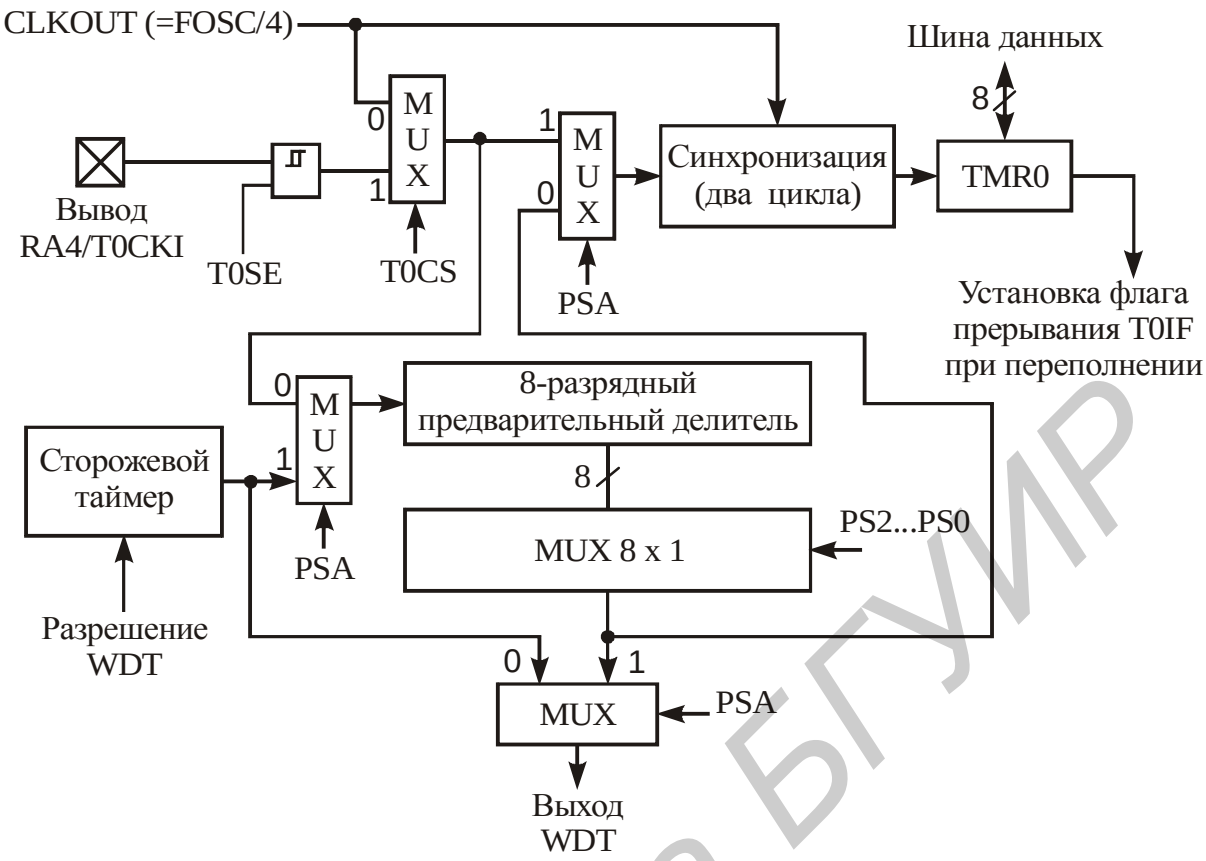
Назначение предварительного делителя задается программно и может быть изменено в процессе выполнения программы. Чтобы избежать непредусмотренного сброса контроллера, при **переключении** предварительного делителя **с TMR0 на WDT** должна быть выполнена такая последовательность команд:

```
BCF     STATUS,RP0      ;Установка банка 0.
CLRF    TMR0            ;Сброс TMR0.
BSF     STATUS,RP0       Установка банка 1.
CLRWDT                  ;Сброс WDT и предварительного делителя.
MOVLW   b'xxxx1xxx'     ;Укажите новое значение предварительного
MOVWF   OPTION          ; делителя.
BCF     STATUS,RP0      ; Установка банка 0.
```

Для **переключения** предварительного делителя **с WDT на TMR0** должна быть выполнена последовательность команд:

```
CLRWDT                  ;Сброс WDT и предварительного делителя.
BSF     STATUS,RP0;Установка банка 1.
MOVLW   b'xxxx0xxx'   ;Указать новое значение предварительного делителя,
MOVWF   OPTION        ;источник тактирования и фронт переключения TMR0.
BCF     STATUS,RP0;Установка банка 0.
```

Эта последовательность должна быть выполнена даже в том случае, если сторожевой таймер WDT запрещен.



Примечание. Биты T0CS, T0SE, PSA, PS2...PS0 – биты регистра OPTION <6...0>.

Рис. 5.11. Схема использования предварительного делителя

6. ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

6.1. Составление схем алгоритмов

Микропроцессорные устройства способны выполнять широкий круг задач по получению, передаче, хранению, переработке информации, принятию решений и т. д. Но все эти действия должны быть заранее подготовлены, запрограммированы человеком.

Создание программы основывается на алгоритме решения задачи, в соответствии с ним создается последовательность команд, которая и составляет программу.

Под алгоритмом понимается конечный набор действий для выполнения некоторой процедуры, удовлетворяющий трем основным требованиям: массовости, детерминированности и результативности.

Требование массовости предполагает, что предписание должно обеспечивать выполнение не одной конкретной процедуры, а быть пригодным

для реализации класса однородных процедур. Бессмысленно писать программу для получения суммы двух констант, но имеет смысл программа, определяющая сумму двух переменных, которые могут принимать множество значений в некотором диапазоне.

Детерминированность означает, что действия, образующие алгоритм, должны быть однозначно понимаемы, т.е. при одинаковых исходных данных независимо от исполнителя должна обеспечиваться одинаковость результатов.

Результативность обеспечивает конечность применения указаний. Результат должен быть получен за конечное число шагов либо за конечное число шагов должно быть получено указание о неприменимости данного алгоритма для решения поставленной задачи.

Алгоритм может быть описан в виде набора предложений, отражающих последовательность действий исполнителя, или в виде схемы, состоящей из ряда блоков, обозначающих действия исполнителя. Последний вариант хотя и имеет большую трудоемкость, но отличается большей наглядностью.

Для успешного решения задачи на вычислительной машине (микроконтроллере) разработчик должен пройти следующие семь этапов: 1) постановка задачи; 2) выбор приемлемого алгоритма; 3) определение типов входных и выходных данных; 4) распределение аппаратных ресурсов микроконтроллера, т.е. портов ввода/вывода и периферийных устройств на кристалле; 5) проектирование и анализ решения, в том числе составление схем, описаний и пр.; 6) кодирование алгоритма на языке программирования; 7) проверка и отладка программы.

Процесс решения задачи носит, как правило, итерационный характер. Это означает, что, получив решение, разработчик часто бывает вынужден вернуться вновь к третьему, второму и даже первому этапу.

По-видимому, первые рассмотренные выше четыре этапа относятся к тому, что принято называть «искусством разработки». Здесь успех в основном определяется опытом разработчика, его знанием объекта разработки, поэтому дать конкретные рекомендации не представляется возможным. Однако можно детальнее остановиться на составлении схем алгоритмов.

Изображение алгоритма решения задачи в виде схемы – важный этап подготовки задачи к решению на вычислительной машине. Схема позволяет разработчику адекватно представить работу программы. Кроме этого, схема алгоритма является одной из важных частей документации на разрабатываемую систему или устройство.

Схема алгоритма составляется из отдельных операторов. Различают шесть типов операторов, каждый из которых имеет один или несколько входов или один или несколько выходов (рис. 6.1). Стрелками обозначают направление хода действий.

Оператор в форме прямоугольника (см. рис. 6.1, *а*) символизирует выполнение каких-либо операций по обработке данных; текст внутри прямоугольника является кратким описанием этого процесса обработки. Например, если в схеме алгоритма содержится оператор «Очистка аккумулятора», то это означает, что на данном этапе работы машины аккумулятор должен быть обнулен. В таком случае для выполнения этой операции микроконтроллеру требуется одна машинная команда. Однако могут быть заданы сложные действия, для которых требуется целый набор команд.

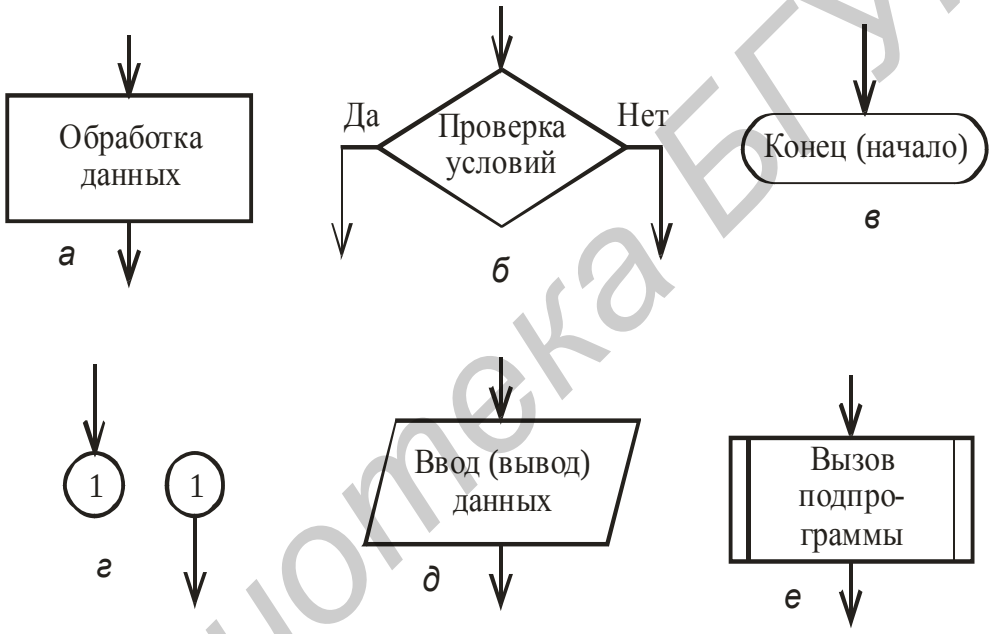


Рис. 6.1. Операторы обработки данных (*а*), проверки условия (*б*), начала или конца алгоритма (*в*), соединения (*г*), ввода или вывода данных (*д*), вызова подпрограммы (*е*)

Оператор, имеющий форму ромба (см. рис. 6.1, *б*), используется для символического обозначения проверки выполнения какого-либо условия с целью принятия решения о направлении последующего хода вычислений. Внутри ромба описывается условие, подлежащее проверке в той точке схемы алгоритма, где размещается данный оператор. Возможные результаты проверки указываются на линиях, выходящих из ромба. Для проведения подобной проверки требуется использование одной или нескольких команд микроконтроллера.

Оператор овальной формы используется для символического обозначения начала или конца алгоритма (см. рис. 6.1, *в*). Текст внутри овала, как правило, состоит из одного слова — «Начало», «Конец», «Выход», «Возврат» или имени подпрограммы.

В тех случаях, когда необходимо «разорвать» линию потока вычислений, идущую от одного оператора к другому, применяются так называемые соединители в виде окружности с указанной внутри нее цифрой или буквой (см. рис. 6.1, *г*). Наличие другого идентичного соединителя (с той же цифрой или буквой) означает, что прерванная в месте расположения первого соединителя линия продолжается с того места, где находится второй подобный соединитель. Использование соединителей упрощает внешний вид схемы алгоритма, что позволяет избежать пересечения линий и даёт возможность размещать схему алгоритма на нескольких страницах.

Для обозначения процедур ввода или вывода применяется оператор, имеющий форму параллелограмма (см. рис. 6.1, *д*). Внутри параллелограмма указывают обычно переменные, подлежащие вводу или выводу.

Оператор, изображенный на рис. 6.1, *е*, используется для обозначения вызова подпрограммы. Внутри него обычно помещается имя вызываемой подпрограммы.

Алгоритмы в зависимости от порядка следования операций бывают трех типов: линейные, ветвящиеся и циклические.

Схемы линейных алгоритмов не содержат операторов проверки условий, в них операторы располагаются строго друг за другом.

Ветвящиеся алгоритмы состоят из двух и более параллельных линейных ветвей, при этом ветвления реализуются через операторы проверки условий.

Циклический алгоритм предписывает многократные циклические проходы группы операторов.

Реальные схемы алгоритмов, как правило, представляют собой комбинации из трех перечисленных выше типов схем алгоритмов.

6.2. Общие правила программирования на языке Ассемблер

Обычно программа начинается со строки комментариев, в которых указывается, чья это программа и зачем она написана. Строки комментариев могут быть в любом месте программы.

Далее после комментариев (если они есть) и перед первыми командами программы размещаются строки директив. Директивы – это указания Ассемблеру, как именно работать. Список директив Ассемблера довольно обширный, но на данном этапе используются только самые необходимые.

Комментарии и директивы не включаются в исполняемый код программы и, следовательно, не попадают в память программ. Строки директив также могут быть в любом месте программы, но некоторые директивы обязательно должны быть заданы до первой строки команд.

Рассмотрим обязательные элементы программы.

Директива LIST с опцией P указывает, с каким микроконтроллером идет работа.

Директива EQU присваивает символическому имени определенное выражение или значение. Присвоенное значение впоследствии в программе переопределить нельзя. Эта директива позволяет программисту оперировать в программе не физическими адресами регистров (ячеек памяти), а их условными именами, которые придумывает сам программист. Если при этом в условное имя закладывается физический смысл переменной, размещающейся в этом регистре, то его легче помнить.

Директива END означает, что текст программы закончился. Эта директива должна быть в последней строке программы.

Каждая строка программы может иметь до четырех разделов (полей) и содержать до 255 символов.

С первой позиции в строке начинается поле метки. Метка – это условное символическое имя, присваиваемое конкретной строке, а также переменной или константе. Метка должна начинаться с буквы или символа подчеркивания (_) и содержать до 32 символов букв или цифр. В поле метки прописные и строчные буквы различаются. Поле метки должно заканчиваться символом пробела, табуляции или конца строки.

Далее следует второе поле – мнемонический код команды. Оно начинается со второй позиции в строке, а если перед ним стоит метка, то от метки мнемоника должна быть отделена двоеточием, одним (или более) пробелом или символом табуляции.

В третьем поле задаются операнды. От мнемоники они отделяются одним (или более) пробелом или символом табуляции. Операндов должно быть столько, сколько требует формат команды. Между собой операнды разделяются запятой. Если команда предусматривает переменное количество операндов, то они считаются до конца строки или до четвертого поля (начало комментария).

Четвертое поле – комментарии. Начинаются с символа точки с запятой. От остальных полей отделяются одним (или более) пробелом или символом табуляции. Могут занимать всю строку с первой позиции.

Программа обязательно заканчивается директивой END.

6.3. Пример простейшей программы управления портами

Пусть требуется спроектировать устройство на микроконтроллере (МК) PIC16F628, которое обеспечивает мигание светодиода на время нажатия и удержания кнопки. Другими словами: пока нажата кнопка, светодиод мигает, при отпущенной кнопке светодиод погашен.

Решение. В этой задаче один источник информации и один объект управления. Источник информации представляет собой два нормально разомкнутых контакта SB. При нажатии кнопки контакты замыкаются, при отпускании – размыкаются. Микроконтроллер должен определять состояния кнопки. Таких состояний два: «не нажата» и «нажата». Эти состояния можно условно описать одной булевой переменной x, которая может принимать два значения: 0 и 1. Таким образом, кнопка как источник даёт один бит информации. Для приёма этой информации достаточно задействовать лишь какой-либо один из входов МК. Сама по себе кнопка является электрически пассивным источником, и для определения ее состояния с помощью порта МК необходимо преобразовать состояние контактов в электрический сигнал напряжения. Наиболее просто это можно выполнить по схеме, изображенной на рис. 6.2.

В состоянии «не нажата» по порту МК будет читаться логическая единица, а в состоянии «нажата» – логический нуль. Учитывая, что в PIC16F628 все разряды порта B имеют подтягивающие внутренние резисторы, внешний резистор R можно не использовать, а кнопку подключить к выводу порта и к общему проводу. При этом в программе необходимо позаботиться о включении подтягивающих резисторов. Подключим в проектируемом устройстве кнопку SB к порту RB0.

Объектом управления в устройстве является светодиод. Его состояние «включен» обеспечивается пропусканием тока $I_Д$ силой в несколько миллиампер (обычно 5–10 мА). Поскольку нагрузочная способность портов МК 20 мА, то светодиод можно подключать без дополнительного усилителя непосредственно к порту, предусмотрев лишь токоограничивающий резистор R, как показано на рис. 6.3. Сопротивление резистора можно рассчитать по формуле $R = (E^1 – U_{VD}) / I_Д$, где $E^1 = 5$ В – уровень логической единицы, $U_{VD} = 2{,}4$ В – падение напряжения на открытом светодиоде. Зададимся $I_Д = 5$ мА, тогда $R = (5–2{,}4) / (5 \cdot 10^{-3}) = 2{,}6 / (5 \cdot 10^{-3}) = 2600 / 5 =$ $= 520$ Ом.

Выберем для управления светодиодом порт RA0.

Для обеспечения требований к временным параметрам устройства необходимо выбрать частоту и вариант тактирования работы МК. Выберем вариант синхронизации с помощью кварцевого резонатора, имеющего

резонансную частоту 4 МГц. Таким образом, длительность машинного цикла (время выполнения большинства команд) составит 1 мкс.

Составим схему алгоритма управляющей программы (рис. 6.4).

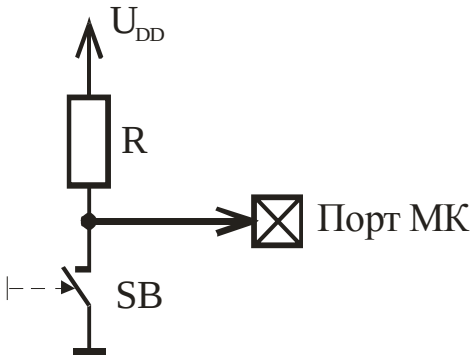


Рис. 6.2. Схема подключения кнопки к порту МК

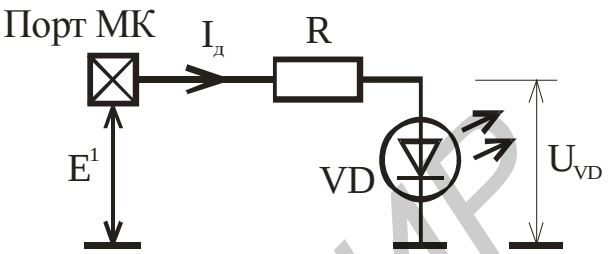


Рис. 6.3. Схема подключения светодиода к МК

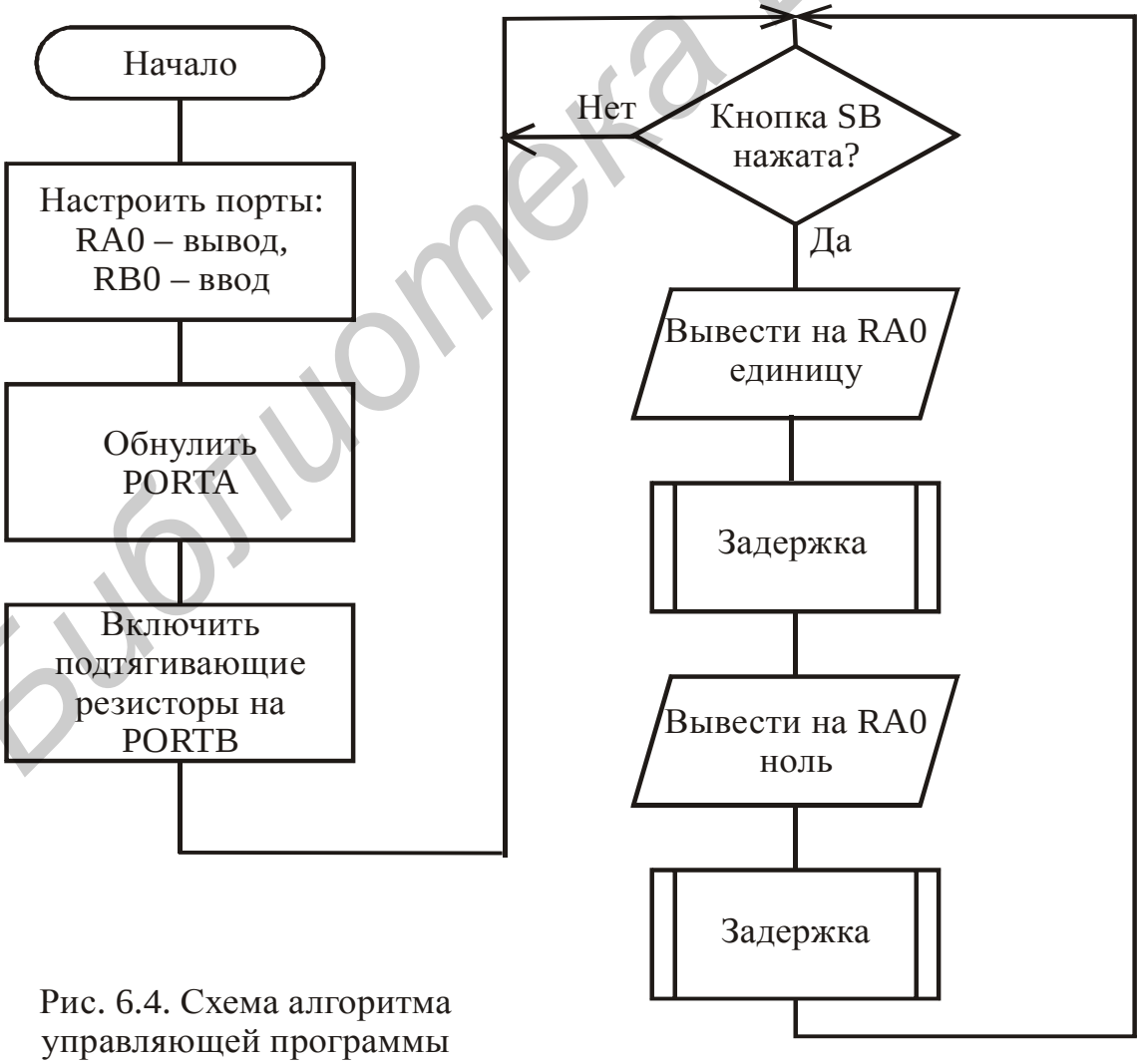


Рис. 6.4. Схема алгоритма управляющей программы

От схемы алгоритма легко перейти к программе.

```
; Учебная программа №1.
        LIST p=16f628     ; Директива Ассемблеру с указанием типа МК PIC16F628.
  ; Определяем имена констант, используемых в этой программе.
  ; После этого определения можно вместо 16-ричного адреса регистра
  ; или номера разряда в регистре указывать его имя, которое сами придумаем.
  ; Это облегчает чтение и понимание работы программы.
STATUS          EQU     03h     ; Символ  h  в конце записи числа является
RP0             EQU     05h     ; указателем 16-ричной
TRISA     EQU       05h     ; системы счисления.
TRISB     EQU       06h     ;
PORTA           EQU     05h     ;
PORTB           EQU     06h     ;
RA0             EQU     0h      ;
RB0             EQU     0h      ;
OPTION_REG      EQU     01h     ;
RBPU      EQU       07h     ;
        ; Настраиваем порты
        BSF   STATUS,RP0      ; Включаем банк памяти 1 для доступа к регистрам TRIS.
        BCF   TRISA,RA0       ; Линия RA0 – на вывод  данных.
        BSF   TRISB,RB0       ; Линия RB0 – на ввод данных.
        BCF   OPTION_REG,RBPU ; Включаем подтягивающие резисторы на входах
                              ; порта B, запрограммированных как входы.
        BCF   STATUS,RP0      ; Включаем банк памяти 0 для доступа к портам.
        CLRFPORTA             ; Обнуляем порт A (гасим светодиод).
                              ; Ожидаем нажатия кнопки SB.
Wait    BTFSC PORTB,RB0       ; В цикле проверяем линию RB0.
        GOTO      Wait        ; При нажатии кнопки на линии будет читаться ноль,
        ; что вызовет пропуск команды GOTO и выход из цикла.
        BSF   PORTA,RA0       ; Зажигаем светодиод.
        CALL      Delay       ; Вызываем подпрограмму задержки.
        BCF   PORTA,RA0       ; Гасим светодиод.
        CALL      Delay       ; Снова задержка.
        GOTO      Wait        ; Переход в точку контроля нажатия кнопки.
Delay   ;Подпрограмма задержки.
; Методы формирования задержек рассматриваются  ниже.
; В простейшем варианте это цепочка команд NOP.
        NOP
        NOP
        NOP
        RETURN    ; Возврат из подпрограммы задержки.
        END       ; Конец программы.
```

6.4. Программирование задержек

Процедура задержки является одной из самых часто используемых, особенно в алгоритмах управления, а также формирования и анализа сигналов. В рассматриваемом микроконтроллере она может быть реализована тремя способами:

- линейной цепочкой пустых команд NOP;
- многократным циклическим повторением цепочки команд, в том числе NOP;
- с применением счетчика-таймера.

Первый способ самый простой, но он пригоден только для реализации коротких задержек. Величина задержки рассчитывается по формуле $\Delta t = t_{кц} N$, где $t_{кц}$ – длительность командного цикла (при тактовой частоте 4МГц она равна 1 мкс), N – количество команд NOP в цепочке.

По второму способу организуется конечное число проходов цепочки команд. Для этого отводится одна из ячеек памяти данных, в которой организуется счетчик циклов. В ячейку заносится константа, которая определяет количество повторений. После каждого прохода из счетчика циклов вычитается единица. Выход из цикла обеспечивается по нулевому значению счетчика. Максимальное значение счетчика циклов ограничивается разрядностью ячейки. Оно не может быть больше 255. Задержка, обеспечиваемая этим способом, рассчитывается по формуле $\Delta t = t_{цк} M$, где $t_{цк}$ – время выполнения цепочки команд, M – количество повторений.

Рассмотрим пример циклической программы задержки.

```
N       EQU     20h   ; Резервируем ячейку памяти по адресу 20h под счетчик циклов.
        MOVLW   07h   ; Заносим число 7 в рабочий регистр,
        MOVWF   N     ; а затем переписываем его в счетчик циклов.
Cycl                  ; Начало циклического участка программы.
        NOP           ; Команды NOP для удлинения времени выполнения участка.
        NOP           ; В реальной программе команды NOP могут отсутствовать.
        NOP           ;
        DECFSZ  N,1   ; Вычитание из счетчика циклов единицы и проверка условия
        GOTO    Cycl  ; выхода из цикла.
```

Для организации больших задержек могут быть использованы вложенные циклы. Рассмотрим фрагмент программы, реализующий задержку с двукратным вложенным циклом.

```
N0      EQU     20h   ; Ячейка памяти по адресу 20h под счетчик внутренних циклов.
N1      EQU     21h   ; Ячейка памяти по адресу 21h под счетчик внешних циклов.
        MOVLW   0Eh   ; Задаем количество внешних циклов
        MOVWF   N1    ;       N1=14.
Cycl_1                ; Начало внешнего цикла.
        MOVLW   08h   ; Задаем количество внутренних циклов
        MOVWF   N0    ;       N0=8.
Cycl_0                ; Начало внутреннего цикла.
        DECFSZ  N0,1  ; Вычитание из счетчика внутренних циклов единицы
        GOTO    Cycl_0      ; и проверка условия выхода из цикла.
        DECFSZ  N1,1  ; Вычитание из счетчика внешних циклов единицы
        GOTO    Cycl_1      ; и проверка условия выхода из цикла.
```

Недостатком рассмотренных способов организации задержек является то, что микроконтроллер занят только задержками и не может выполнять другие полезные действия.

Наиболее совершенный способ реализации задержек – с помощью счетчика-таймера TMR0. Как и в предыдущем случае, задержка отмеряется по обнулению счетчика, в который предварительно записывается начальная константа К. Но в отличие от предыдущего случая счетчик переключается не программно, а аппаратно и обнуление счетчика фиксируется через систему прерываний. Так что в промежутках времени от запуска задержки до ее окончания микроконтроллер может выполнять любые другие полезные действия. TMR0 в процессе работы инкрементируется с частотой командных циклов. Его обнуление происходит через переполнение. Таким образом, реализуемая задержка $\Delta t = (255–K)\ t_{кц}$, где $t_{кц}$ – длительность командного цикла.

Увеличить время задержки можно, подключив к входу TMR0 предварительный делитель и задав с помощью регистра OPTION_REG его коэффициент деления N. Тогда реализуемая задержка $\Delta t = N(255–K)\ t_{кц}$.

Рассмотрим реализацию генератора прямоугольных импульсов (меандра) на базе таймера TMR0. Пусть выходной сигнал формируется на выводе RA0, а частота синхронизации микроконтроллера равна 4 МГц. Тогда длительность командного цикла $t_{кц}$ = 1 мкс.

; Пример программной реализации генератора прямоугольных импульсов.

```
LIST p=16f628
#INCLUDE        p16f628.inc     ; Включаем в программу файл p16f628.inc,
                ; в котором описаны имена регистров МК и их физические адреса.
```

```
; Определяем константы и адреса регистров, флагов и переменных.
K       EQU  0AAh       ; Начальное значение таймера TMR0=170.
W_TEMP  EQU  20h        ; Ячейка памяти для сохранения регистра W
                        ; на время обработки прерывания.
STATUS_TEMP  EQU 21h    ; Ячейка памяти для сохранения регистра
                        ; STATUS на время обработки прерывания.
        ; Начало программы.
        ORG 0           ; Директива Ассемблеру установить программный счетчик в нуль.
        GOTO    Begin   ; Перейти к началу основной программы.
        ORG  4          ; Установить программный счетчик на адрес вектора прерывания.
        GOTO    Int     ; Перейти на программу обработки прерывания.
Begin   ; Начало основной программы.

; Выполняем настройки режимов работы функциональных узлов микроконтроллера.
        BSF     STATUS,RP0          ; Включаем банк регистров 1.
        BCF     TRISA,RA0           ; Порт RA0 включаем на выдачу сигналов.
        BCF     OPTION_REG,T0CS     ;Включаем внутреннее тактирование TMR0.
        BSF     INTCON,T0IE     ; Разрешаем прерывания по переполнению TMR0.
        BCF     STATUS,RP0      ; Включаем банк регистров 0.
        CLRF    PORTA           ; Обнуление выходного сигнала.
        MOVLW   K               ; Загрузка TMR0
        MOVWF   TMR0            ; начальным значением.
        BSF     INTCON,GIE      ; Разрешаем прерывания от переполнения TMR0.
            ; Здесь могут располагаться команды,
            ; выполняющие некоторые полезные действия, например,
            ; обслуживание индикатора, ввод информации с клавиатуры и т.д.
Wait    GOTO Wait       ; Зацикливаем участок программы в ожидании прерывания.
            ; Выход отсюда возможен только по прерыванию.
Int     ; Начало подпрограммы обработки прерывания. Здесь на каждое
        ; прерывание производится инверсия порта А. Поскольку на выход
        ; настроен только вывод RA0, то инвертировать будет только он.
        MOVWF   W_TEMP      ; Сохранить W в регистре TEMP.
        SWAPF   STATUS,0
        MOVWF   STATUS_TEMP         ; Сохранить STATUS в регистре TEMP.
        BCF     STATUS,RP0          ; Включаем банк регистров 0.
        COMF    PORTA,1             ; Инверсия выходной линии.
        MOVLW   K                   ; Запись начальной константы К
        MOVWF   TMR0                ; в таймер TMR0.
        BCF     INTCON,T0IF     ; Сброс флага прерывания по переполнению TMR0.
```

```
                ; Восстанавливаем STATUS и W.
SWAPF   STATUS_TEMP,0   ;
MOVWF   STATUS          ;
SWAPF   W_TEMP,1        ;
SWAPF   W_TEMP,0        ;
RETFIE          ; Вернуться из прерывания, разрешить следующее прерывание.
END             ; Конец программы.
```

ЛИТЕРАТУРА

1. PIC16F62X. Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip technology incorporated : пер. с англ. – М. : ООО «Микрочип», 2001. – 148 с. www.microchip.ru
2. Сергеев, Н. Р. Основы вычислительной техники : учеб. пособие для вузов / Н. Р. Сергеев, Н. Р. Вашкевич. – М. : Высш. шк., 1988. – 311 с.
3. Каган, Б. М. Электронные вычислительные машины и системы : учеб. пособие для вузов / Б. М. Каган. – М. : Энергоатомиздат, 1991. – 592 с.
4. Микропроцессорные системы : учеб. пособие для вузов / Е. К. Александров, [и др.] ; под общей ред. Д. В. Пузанкова. – СПб. : Политехника, 2002. – 935 с.
5. Бродин, В. Б. Системы на микроконтроллерах и БИС программируемой логики / В. Б. Бродин, А. В. Калинин. – М. : ЭКОМ, 2002. – 400 с.
6. Предко, М. Справочник по PIC-микроконтроллерам / М. Предко : пер. с англ. – М. : ДМК Пресс, 2004. – 512 с.

СОДЕРЖАНИЕ

Св. план 2007, поз. 83

Учебное издание

Левкович Василий Николаевич,
Шабров Олег Васильевич

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Учебно-методическое пособие
для студентов радиотехнических специальностей
всех форм обучения

Редактор Т. П. Андрейченко
Корректор Е. Н. Батурчик

Подписано в печать 21.08.2007. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 5,46.
Уч.-изд. л. 5,0. Тираж 200 экз. Заказ 197.

Издатель и полиграфическое исполнение:
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6