Michael Hofmann

Mikrocontroller für Einsteiger

FRANZIS
PC&ELEKTRONIK

Michael Hofmann

Mikrocontroller für Einsteiger

Schaltungen entwerfen und Software programmieren

Mit 102 Abbildungen

Михаэль Хофманн

МИКРОКОНТРОЛЛЕРЫ для начинающих

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 621.382
ББК 32.85
Х86

Хофманн М.

Х86 Микроконтроллеры для начинающих: Пер. с нем. — СПб.: БХВ-Петербург, 2010. — 304 с.: ил. + CD-ROM — (Электроника)

ISBN 978-5-9775-0551-2

Рассмотрено программирование микроконтроллеров на примере PIC16F876A компании Microchip. Подробно описаны основные команды языка ассемблер, а также среда разработки MPLAB. Показано программирование с помощью отладчика-программатора ICD 2, а также через последовательный интерфейс. На практических примерах рассмотрено управление светодиодами и дисплеем, представление аналоговых сигналов в цифровой форме, сохранение/запись данных во внешнюю EEPROM-память, управление выходами микроконтроллера с помощью ИК-пульта дистанционного управления и др. На компакт-диске приведены примеры программ, чертеж для изготовления монтажной платы, электрические схемы, техническая документация, справочная информация и программное обеспечение.

Для радиолюбителей

УДК 621.382
ББК 32.85

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Перевод с немецкого	*Виктора Букирева*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.10.
Формат 70×100 $^{1}/_{16}$. Печать офсетная. Усл. печ. л. 24,51.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-3-7723-4318-6 (нем.)
ISBN 978-5-9775-0551-2 (рус.)

Оглавление

Предисловие

Микроконтроллеры находятся почти во всех электронных устройствах. В этой книге показано, что запрограммировать и применить микроконтроллер не так уж сложно. Хотя у некоторых при упоминании слова *"Ассемблер"* (Assembler) в связи с программированием "волосы встают дыбом". После прочтения этой книги вы поймете, что это вовсе не так уж сложно, как это иногда кажется.

Возможности микроконтроллера рассмотрены на различных примерах, в которых используется микроконтроллер *PIC16F876A* производства компании Microchip, имеющий различные интерфейсы и обладающий достаточно широкими возможностями. На примерах будет показано, как осуществить опрос входов и переключение выходов микросхемы. Вы также научитесь управлять дисплеем для отображения текста и данных. Узнаете, как измерить аналоговые сигналы, сохранить их в *EEPROM*-памяти (Electrically Erasable Programmable Read-Only Memory — электрически стираемое программируемое постоянное запоминающее устройство) и затем прочитать с помощью персонального компьютера. На другом примере будет показано управление выходами с помощью инфракрасного управления. Часть примеров может моделироваться с помощью среды разработки *MPLAB*, поэтому в принципе не потребуются какие-либо аппаратные средства. Но поскольку любой микроконтроллер должен когда-нибудь хоть раз использоваться для схемы, представляется маленькая монтажная плата, на которой все примеры могут испытываться в реальной среде. Для облегчения работы и отладки монтажа вы можете приобрести неукомплектованную (без деталей) печатную плату, если посетите мой сайт (**www.edmh.de**). Если вы захотите сами дополнить ее, то найдете расположение схемных элементов в Eagle-формате на приложенном CD-ROM. На нем находится также различная документация, например, обзор команд и описания регистров микроконтроллера. Для программирования микроконтроллера используется внутрисхемный отладчик *ICD 2* (In-Circuit Debugger) от компании Microchip. Однако на CD-ROM находится очень простая схема, с которой PIC-контроллер может программироваться через последовательный интерфейс.

Я желаю вам большого удовольствия и успеха при программировании микроконтроллера. Я всегда открыт для критики, похвалы и рационализаторских предложений и рад корреспонденции, поступающей на адрес моей электронной почты **info@edmh.de**.

Михаэль Хофманн

1. Обзор микроконтроллеров

Сегодня микроконтроллеры имеются почти в каждом электронном устройстве. Они применяются в термометрах для отображения температуры и управляют требуемым составом кофе в кофеварках. Микроконтроллеры автоматически открывают и закрывают ворота гаражей и выручают шофера в опасных ситуациях с помощью противоблокировочных устройств (ABS) и системы антивибрационной защиты (ESP).

Микроконтроллер (сокращенно МК, англ. MC) или также *микрокомпьютер* — это электронная микросхема, которая может осуществлять вычисления и управление техническими объектами и технологическими процессами подобно персональному компьютеру. Разумеется, МК чаще предназначен для управления конкретным устройством, а персональный компьютер способен выполнять обработку всех данных и обычно управляет многими разными устройствами. Персональный компьютер более универсален, поскольку используется для обработки текста, для компьютерных игр и многого другого, а поэтому должен быть очень гибок и предоставлять в распоряжение пользователя достаточное пространство памяти и необходимую вычислительную мощность. Напротив, микроконтроллер имеет узконаправленное предназначение. Если он используется, например, только для управления температурой в помещении, тогда для вычислений не требуется высокая производительность и не нужен большой объем памяти. При этом должна измеряться и обрабатываться лишь температура. Разумеется, строгую границу между микроконтроллером и персональным компьютером провести нельзя. Микроконтроллер в мобильном телефоне необходим для выбора номера абонента и обеспечения телефонного разговора. Однако при помощи современных карманных компьютеров можно отправлять электронные письма, записывать видео и слушать музыку.

Поскольку обычно для каждого специального задания не изготавливают узкоспециализированный микроконтроллер, поэтому из достаточно большого количества моделей универсальных контроллеров можно вполне выбирать

подходящий. Имеются разные производители, которые изготавливают много вариантов контроллеров, которые едва ли принципиально отличаются режимами функционирования. Микроконтроллеры располагают разным количеством входов и выходов, а также специальными аппаратными модулями, находящимися внутри, с которыми упрощается выполнение различных заданий. Таким образом, почти каждый микроконтроллер имеет таймер, с помощью которого можно устанавливать время и генерировать сигналы определенной длительности. Кроме того, многие микросхемы располагают встроенным *аналого-цифровым преобразователем* (АЦП), который позволяет измерять аналоговые сигналы для контроля, например, температуры или напряжения батареи.

Габариты микросхем определяются преимущественно количеством входов и выходов. Таким образом, микроконтроллер, который должен наблюдать только за напряжением батареи и выдавать сигнал при выходе за пределы порогового значения, может обходиться малым количеством выводов. Напротив, контроллер, который должен регулировать температуру в нескольких комнатах и отображать этот процесс управления на цветном индикаторе с сенсорным экраном, нуждается в существенно большем количестве входов и выходов. Поэтому у производителей можно найти микроконтроллеры как с шестью выводами, так и с несколькими сотнями выводов. Кроме того, можно заметить, что обычно с увеличением количества выводов микросхемы МК вычислительная мощность также возрастает, поскольку большее количество выводов позволяет решить соответственно и большее число разнообразных задач. Также количество битов, которые одновременно обрабатываются, растет с габаритами микроконтроллера. В настоящее время имеются модули с разрядностью шины от 4 до 32 битов. Для персонального компьютера уже применяются 64-битные процессоры. В скором времени микроконтроллер тоже наверняка сможет оперировать 64-битными значениями. Будет ли это теперь преимуществом или скорее недостатком, каждый разработчик должен решать сам. Микроконтроллеры с длиной слова 4 бита можно найти преимущественно в музеях и очень старых устройствах. Однако они продаются еще и сегодня для восстановления старых устройств. В современных разработках эти микросхемы больше не используются. В настоящее время больше всего распространены микроконтроллеры с длиной слова от 8 до 16 битов. Выбор среди них очень велик. А вот 32-битные микроконтроллеры применяются преимущественно для устройств, которые управляют цветным графическим индикатором (дисплеем), поддерживают различные носители информации, такие как USB- или SD-карты, и предназначены для подключения к персональному компьютеру для обмена данными...

В большинстве случаев 8-битных микроконтроллеров бывает вполне достаточно и к тому же их приобретение менее затратно. Кроме того, небольшие

электронные схемы могут быть реализованы без изготовления печатной платы. Так как многоразрядные микроконтроллеры функционируют практически аналогично контроллерам с меньшим числом разрядов, то в дальнейшем преимущественно будут приведены схемы с использованием 8-битного микроконтроллера. Представленные схемы и примеры программ достаточно просто реализовать, смоделировать и испытать.

В дальнейшем рассматриваются только контроллеры фирмы *Microchip*. Все примеры были запрограммированы для PIC-микроконтроллеров (*PIC* — Programmable Integrated Circuit — программируемая интегральная схема) этой фирмы. Контроллеры других производителей, например Atmel, Motorola, Renesas и т. д., функционируют по такому же принципу и отличаются преимущественно возможностями и системой команд.

Почему выбор в этой книге пал на продукты Microchip, зависело, в том числе, оттого, что фирмой Microchip предоставляется в распоряжение пользователей полностью бесплатная среда разработки. Контроллеры можно купить у известных поставщиков электронных комплектующих изделий и они, обладая большой популярностью, имеют много подробных примеров и обсуждений в Интернете.

Представленные в этой книге примеры программировались и испытывались при помощи встроенного отладчика In-Circuit-Debugger (*ICD2*) от компании Microchip. Поэтому все рисунки и описания относятся к этому программатору. Однако имеются также другие программаторы разных производителей, при помощи которых можно программировать и отлаживать микроконтроллер. Также в Интернете находится много инструкций для изготовления самодельных программаторов. Руководство по разработке простого PIC-программатора имеется на приложенном CD-ROM. Но он подходит только для программирования контроллера, отладка (поиск ошибок) с этим простым программным адаптером не возможна.

Программирование осуществлялось на языке программирования ассемблер. В этом случае лучше усваивается принцип работы микроконтроллера, оптимально используются все его системные ресурсы. Также в Интернете можно найти несколько бесплатных компиляторов для языка программирования C. На прилагаемом компакт-диске имеется полный, рабочий программный код с комментариями к каждому примеру. Но чтобы гарантировать оптимальное обучение, требуется пытаться самостоятельно программировать примеры и только в крайнем случае использовать версии, находящиеся на компакт-диске. Программы не оптимизированы на самое короткое время выполнения или самый короткий программный код. Они должны лишь демонстрировать многостороннюю функциональность микроконтроллера.

Все примеры программ в этой книге представлены для микроконтроллера PIC16F876A. У этой микросхемы есть преимущество в том, что она имеет все

основные характерные элементы современных микроконтроллеров. Количество контактов ввода/вывода I/O этого контроллера вполне достаточно для реализации практически любого проекта, однако при необходимости возможен очень простой переход и на другие типы. При этом нужно обязательно проверять названия и адреса регистров, поскольку не все PIC-контроллеры используют одни и те же адреса и обозначения.

1.1. Сравнительные характеристики микроконтроллеров

Следующий краткий обзор представляет только маленькую часть имеющихся в настоящее время микроконтроллеров. Перечень только поясняет, насколько различны микроконтроллеры:

- *PIC10F222* — очень маленький микроконтроллер с шестью выводами и размером 2×3 мм);
- *PIC16F876* — 8-битный микроконтроллер с 28 выводами;
- *PIC24FJ128* — 16-битный микроконтроллер с 64—100 выводами;
- *PIC32MX460* — 32-битный контроллер с 64—100 выводами;
- *ARM9* — 32-разрядный микроконтроллер с ядром ARM (Advanced RISC Machines) в корпусе *BGA* (Ball Grid Array — конструкция корпуса микросхемы с выводами в виде крошечных металлических шариков. Они расположены в виде сетки на его нижней поверхности, которые прижимаются к контактным площадкам на печатной плате без применения пайки. Преимущество — более низкая стоимость изготовления и уменьшение размеров при наличии более трехсот выводов).

1.2. Структура и принцип работы PIC16F876A

1.2.1. Блок-схема

Для обеспечения работы микроконтроллера требуется много маленьких функциональных блоков, которые должны оптимально взаимодействовать. На рис. 1.1 показана упрощенная функциональная схема микроконтроллера PIC16F876A.

Самой важной составной частью микроконтроллера является арифметико-логическое устройство — *АЛУ* (англ. *ALU* — Arithmetic Logic Unit). С помощью АЛУ, которое является *операционным устройством* контроллера,

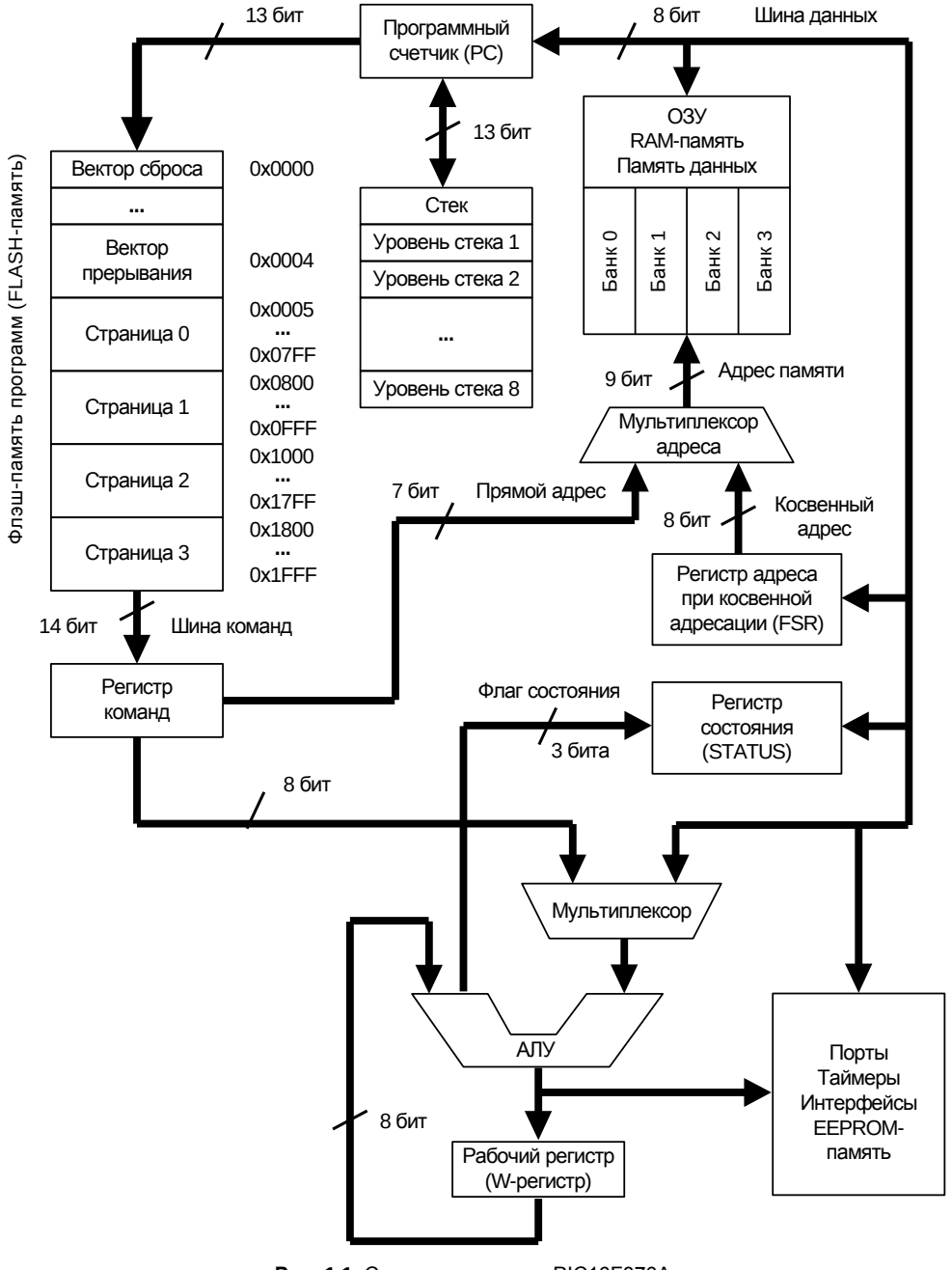

Рис. 1.1. Структурная схема PIC16F876A

выполняются арифметические и логические операции. Чтобы АЛУ могло оперировать с соответствующими значениями, они должны предоставляться в распоряжение в определенное время из соответствующей области памяти.

1.2.2. Флэш-память программ

Сначала разберемся, как данные поступают в микроконтроллер и как они образуются. Микроконтроллер не понимает, к сожалению, такой язык программирования, как Basic, C или ассемблер, а воспринимает только двоичные значения, т. е. нули или единицы. Они записываются во *флэш-память программ* (англ. Flash Program Memory), начиная с адреса 0x0000. Поскольку Flash-память — это энергонезависимая память, в которой данные сохраняются и после выключения питания, то именно здесь и хранят программный код микроконтроллера. Если бы программу поместили в оперативное запоминающее устройство (*ОЗУ*) (англ. *RAM*), то устройство перестало бы функционировать после выключения. Поэтому ОЗУ это память, которая предназначена для хранения промежуточных данных, а не программ. Чтобы сформировать цифровые данные, которые понимает PIC-контроллер, необходимо выполнить несколько последовательных операций. Принципиально можно было бы написать программный код непосредственно в двоичном виде и загрузить его при помощи программатора в микроконтроллер. Однако любой разработчик понимает, что это очень трудоемкая работа, и может восприниматься как "штрафная". Поэтому для программирования микроконтроллеров, как правило, используют различные языки программирования. Наиболее употребительным является язык C или ассемблер. Оба языка очень близки к аппаратным средствам и поэтому могут непосредственно обмениваться сообщениями с физически имеющимися регистрами. Для сложных многоразрядных микроконтроллеров почти всегда применяется язык C, а вот относительно простой контроллер, который рассматривается в этой книге, достаточно часто программируют на ассемблере. Следующий пример показывает программу обратного счета от 9 до 0, написанную на языке C, ассемблере и в машинном коде, который в итоге должен находиться в PIC-контроллере.

Язык C	Ассемблер	Машинный код
`for(i=9; i>0; i--);`	`movlw 0x09`	`000C 3009`
	`movwf 0x20`	`000D 00A0`
	`counter`	`000E 0BA0`
	`decfsz 0x20, 1`	`000F 280E`
	`goto counter`	

Как можно видеть на представленном примере, было бы слишком непонятно и затруднительно программировать микроконтроллер непосредственно в машинных кодах. Поэтому применяют специальную программу, которая транслирует программу, написанную на языке С или ассемблере, в машинный язык. Называют эту программу *компилятор* (англ. compiler). Компилятор проходит шаг за шагом программный код и интерпретирует запрограммированные разработчиком команды. Это часто выполняется не за один, а за несколько проходов. После компилирования отдельные модули еще должны связываться между собой с помощью другой специальной программы — *компоновщиком* (англ. linker). Если все правильно работает и ошибки отсутствуют, то после этого получают машинный код, который может быть применен принципиально только на предусмотренном контроллере. Во время процесса компиляции генерируется файл с расширением hex, в котором записано, в каком месте должны находиться в памяти отдельные команды. Далее этот файл может быть загружен при помощи программатора в микроконтроллер.

Теперь программа записана во флэш-память программ. Если просмотреть вышеупомянутый машинный код, то можно определить место в программном коде по первым четырем шестнадцатеричным цифрам. Код начинался бы в памяти с адреса 0x000C и завершался бы на адресе 0x000F. Теперь для выполнения программы должна определяться исходная точка. Это — адрес 0x0000, с которого выполняется программа после запуска (`Start`) или после сброса (`Reset`). С помощью внешнего или внутреннего тактирования *программный счетчик* PC (англ. Program Counter) (или иначе *счетчик команд*) будет постепенно прибавлять 1 и запускать очередную команду после предыдущей. При отсутствии команд перехода и циклов в программе процесс продолжался бы максимально до адреса 0x1FFF (что соответствует 8292 командам) и повторялся бы после этого опять сначала. Счетчик команд имеет разрядность 13 битов (2^{13} = 8 192) и может обращаться поэтому максимально к 8292 командам. Команда имеет разрядность 14 битов (например, `movlw 0x09` = 0x3009 = 11 0000 0000 1001_2). После обработки команды из регистра команд данные и адреса ОЗУ через внутренние линии передаются соответственно в АЛУ или оперативную память (ОЗУ).

1.2.3. Обработка данных в АЛУ

Данные обрабатываются после поступления в АЛУ. Команда `movlw 0x09` означает: "Загрузить шестнадцатеричное значение 0x09 в рабочий регистр W". Без использования рабочего регистра W в PIC-контроллере практически ничего не работает. Через него должны проходить почти все значения. Например, чтобы записать то или иное значение в ячейку оперативной памяти или иначе в тот или иной регистр ОЗУ (*File Register* — сокращенно регистр F),

оно должно быть сначала загружено в рабочий регистр W (`movlw 0x09`) и только после этого может передаваться дальше в регистр ОЗУ. Это может происходить, например, по команде `movwf 0x20`, которая означает не что иное, как: "Переслать значение из регистра W в регистр ОЗУ по адресу памяти 0x20".

1.2.4. Регистр состояния

Поскольку после выполнения некоторых команд необходимо знать, стал ли результат нулевым после выполнения математической или логической операции или же произошел перенос, в регистре состояния (STATUS) в зависимости от команды устанавливаются определенные биты. Речь идет о так называемых *флагах или битах состояния*. *Флаг* (Flag) — это признак или индикатор для определения того, что произошло после той или иной операции. В регистре состояния микроконтроллера имеются следующие флаги (рис. 1.2): C — флаг переноса (от англ. Carry), DC — флаг десятичного переноса (от англ. Digit Carry) и Z — флаг нулевого результата (от англ. Zero). Флаг переноса указывает на перенос из старшего бита, если, например, после сложения результат больше не может представляться 8-ю битами. Флаг десятичного переноса похож на флаг переноса, но показывает перенос из младшего полубайта, т. е. из четвертого бита. И, наконец, флаг нулевого результата показывает, является ли результат нулем после выполнения арифметической или логической операции или нет.

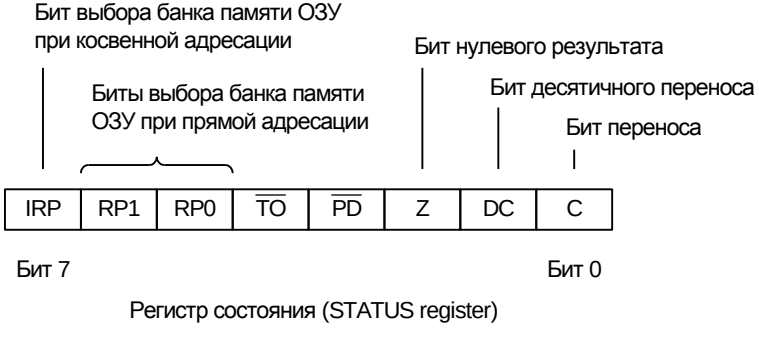

Рис. 1.2. Формат регистра состояния

1.2.5. Адресация оперативной памяти или регистров ОЗУ

При обращении к регистрам ОЗУ имеется небольшая проблема. Адресация осуществляется с помощью 9-разрядного адреса оперативной памяти (RAM-

адреса). С помощью 9 битов можно обращаться максимум к 2^9 = 512 байтам (0x000—0x1FF). Разумеется, при прямой адресации в PIC-контроллере используется только 7 битов, что позволяет обратиться лишь к 2^7 = 128 байтам (0x00—0x7F). Для решения этой проблемы прибегли к следующему трюку: всю память данных (ОЗУ) разделили на *4 банка*, при этом каждый банк состоит из 128 регистров, среди которых имеются *регистры общего назначения* (англ. *GPR* — General Purpose Registers) и *специального назначения* (англ. *SFR* — Special Function Registers). Таким образом, чтобы обратиться к конкретному регистру, вначале нужно выбрать банк, а затем уже в нем найти требуемый регистр. *Выбор банка памяти* осуществляется с помощью двух битов в регистре состояния RP0 и RP1 (рис. 1.3). С помощью этих двух дополнительных битов и получают 9-разрядный адрес оперативной памяти: младшие 7 разрядов определяются из регистра команд при прямой адресации, а два старших бита устанавливаются посредством регистра состояния (рис. 1.4). Здесь также может подстерегать еще одна проблема, которая может возникнуть при программировании: если во время программирования добавляются строки кода, которые обращаются к регистру другого банка, можно легко забыть о переключении банка. Программа в таком случае транслируется правильно, но может повести себя не так, как требуется. Особенно эта опасность возникает, если реализуется выбор банков памяти в пределах *макрокоманды* (инструкция, директива как элемент типичного языка программирования, при обработке развертывающаяся в определенную последовательность простых команд) или подпрограммы. Тогда после возврата к выполнению основной программы можно не заметить, что будет активен уже другой банк.

RP0 = 1 RP1 = 1	RP0 = 0 RP1 = 1	RP0 = 1 RP1 = 0	RP0 = 0 RP1 = 0
0x180	0x100	0x080	0x000
0x181	0x101	0x081	0x001
Банк 3	Банк 2	Банк 1	Банк 0
0x1F0	0x17E	0x0FE	0x07E
0x1FF	0x17F	0x0FF	0x07F

Рис. 1.3. Организация банков памяти

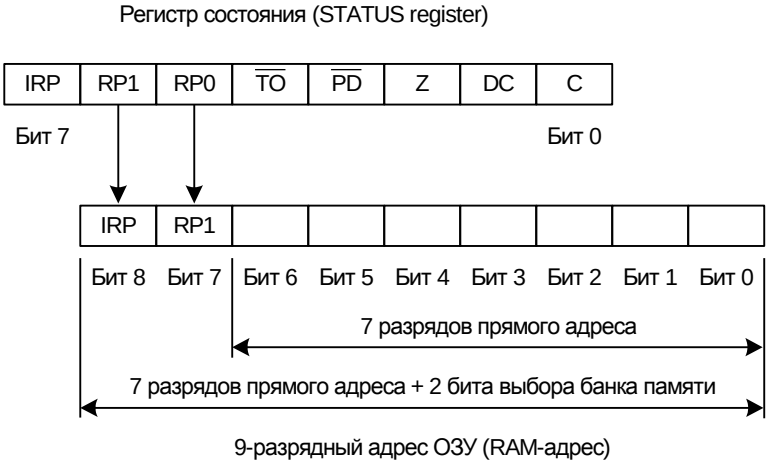

Рис. 1.4. Формирование RAM-адреса. Диапазон адресов банков памяти

1.2.6. Вызов подпрограмм

Схожее действие с макрокомандами выполняют и подпрограммы. Они служат для упрощения выполнения программы и пишутся для часто используемых последовательностей команд (например, "Передать данные через последовательный интерфейс"). Подпрограмма программируется только раз и поэтому занимает одни и те же ячейки в памяти программ. В этом случае подпрограмма может легко вызываться всякий раз, когда это необходимо. Чтобы вызвать подпрограмму, требуется небольшая процедура. Поскольку подпрограмма только раз сохраняется в памяти программ, но, как правило, неоднократно должна вызываться, обратиться к ячейке памяти начала подпрограммы можно с помощью программного счетчика. Чтобы это сделать, нужно выполнить команду перехода к подпрограмме или иначе — команду вызова подпрограммы — `call` (вызов) (рис. 1.5). При этом программный счетчик устанавливается на адрес начала подпрограммы и продолжает работу как обычно. После выполнения подпрограммы нужно вновь вернуться продолжить основную программу. Это происходит по команде `return` (возврат). Только какое значение нужно загрузить в программный счетчик? Для этого в PIC-контроллере при вызове подпрограммы значение программного счетчика временно сохраняется в стеке. *Стек* (англ. stack) — это память, организованная по принципу *LIFO* (англ. Last In — First Out), т. е. "последним пришел — первым вышел". Новое значение в стеке всегда помещается в верхнюю ячейку, которая называется *вершиной стека*. Вызывается это значение из стека опять-таки из вершины стека, причем в первую очередь. Аналогичным образом это функционирует со значением программного счетчика и при вызове подпрограммы. По команде `call` следующее за текущим значение про-

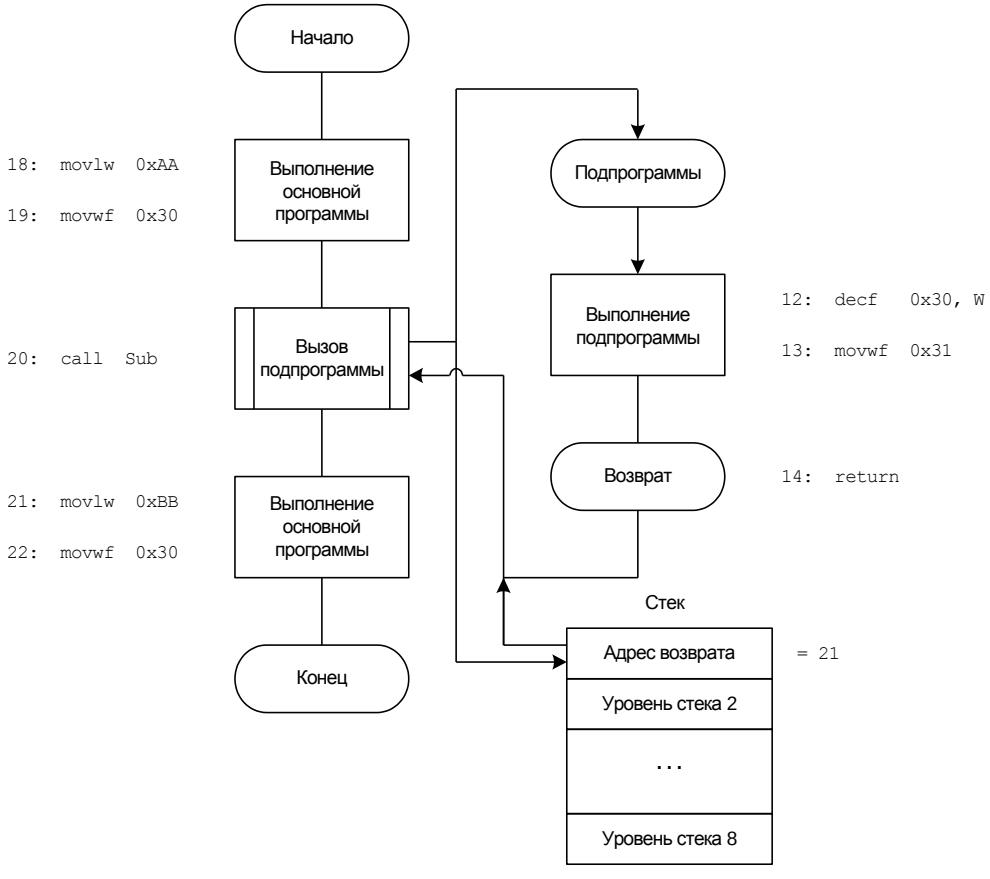

Рис. 1.5. Вызов подпрограммы

граммного счетчика помещается на вершину стека, а по команде `return` значение с вершины стека снова загружается в программный счетчик. Таким образом, работа основной программы может быть продолжена. Стековая память микроконтроллера PIC16F876A имеет глубину равную 8, т. е. она содержит 8 уровней. Другие уровни стека могут заполняться аналогично вершине стека, при условии вызова из данной подпрограммы очередной подпрограммы, которая в этом случае называется *вложенной*. Команда `call` в подпрограмме также помещает следующее за текущим значение программного счетчика на вершину стека (предыдущее значение вершины опускается ниже) и переходит на выполнение вложенной подпрограммы. Возврат к прерванной подпрограмме из вложенной осуществляется также по команде `return`. Таким образом, в данном микроконтроллере могут выполняться максимум 8 вызовов подпрограмм. После этого стековая память будет полностью заполнена, и больше сохранить адрес возврата будет невозможно.

1.2.7. Косвенная адресация

Иногда требуется рассчитывать адрес и обращаться к нему косвенно. "Косвенно" — значит не вводить непосредственно точный адрес, а указывать

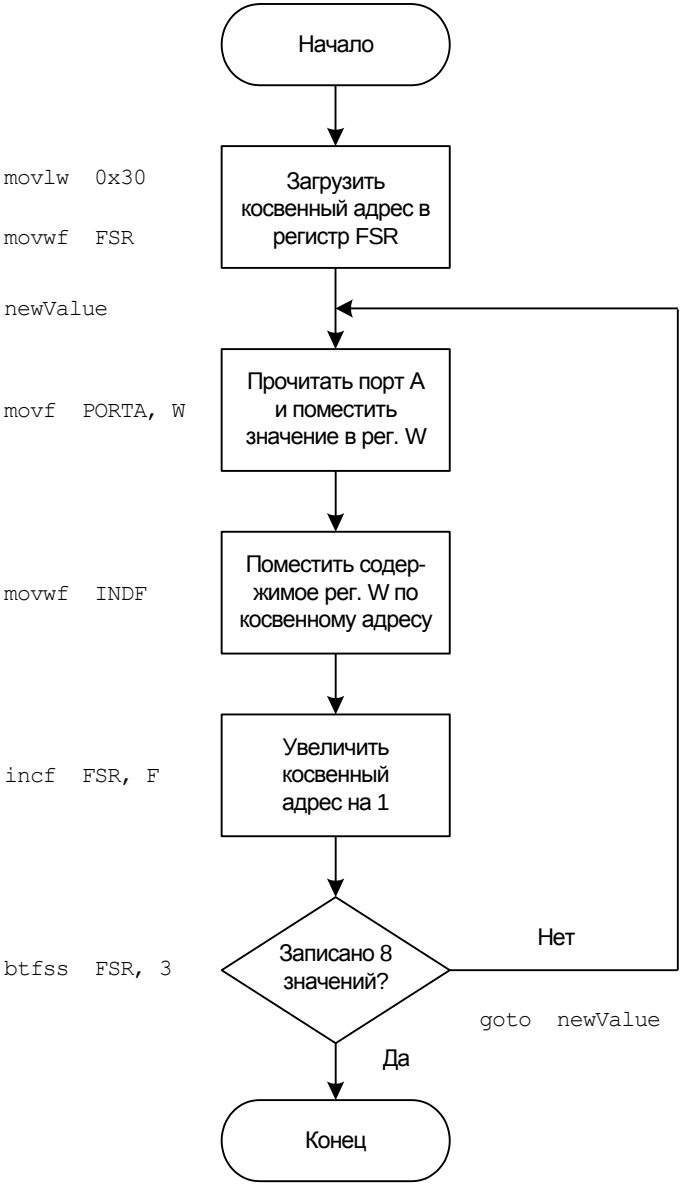

Рис. 1.6. Косвенная адресация

только ссылку на этот адрес. Для этих целей применяется только так называемый *указатель*. Этот метод может использоваться, например, чтобы последовательно записывать или очищать несколько регистров. Следующий пример показывает, как 8 значений по очереди считываются из входного порта A и заносятся в RAM-память.

В описанном примере данные считываются с входов порта A и заносятся в RAM-память по адресам от 0x30 до 0x37.

Так как при косвенной адресации для определения адреса имеются в распоряжении только 8 битов регистра FSR, то при этом можно обращаться далеко не ко всем регистрам RAM-памяти, которая, как известно, имеет 9-разрядный адрес. Поэтому при косвенной адресации снова используют регистр состояния (STATUS), а точнее один старший его бит IRP (рис. 1.7). Чтобы сделать выбор между банками 0—1 и 2—3, нужно установить или сбросить бит IRP. Так если бит IRP установить в 0, то можно обращаться к банкам 0 и 1. Если же бит IRP установить в 1, то адресуются банки 2 и 3. После этого содержимое 8-разрядного регистра FSR объединяется с битом IRP регистра состояния и таким образом образуется 9-разрядный адрес памяти данных. Для осуществления косвенной адресации используют *регистр косвенной адресации* INDF, адрес которого находится в каждом банке по адресу 0x00. Регистр в микроконтроллере физически отсутствует, он используется только как вспомогательное средство для адресации. Если пишут что-то в этот регистр, то запись осуществляется в физически имеющийся регистр, адрес которого находится в регистре FSR. Это же самое происходит и при чтении регистра INDF.

Рис. 1.7. Структура RAM-адреса для косвенной адресации

1.2.8. Чтение и запись внутренней EEPROM-памяти

Поскольку данные, которые сохраняются в регистрах RAM-памяти, пропадают после выключения микроконтроллера, часто необходимо, чтобы данные могли храниться продолжительно. При небольших объемах данных, например, при сохранении последнего рабочего состояния перед выключением, это может происходить через внутреннюю память *EEPROM* (Electronic Erasable Programmable Read Only Memory — электрически стираемое программируемое постоянное запоминающее устройство). В нее можно записывать данные, которые будут храниться так долго, сколько нужно. Запись и чтение памяти EEPROM, к сожалению, осуществляется не так просто, как в случае с регистрами ОЗУ. Судить об этом можно уже только потому, что имеются 6 регистров специального назначения для записи и чтения EEPROM- и Flash-памяти. В микроконтроллере PIC16F876A объем EEPROM-памяти данных равен 256 байтам. Если этого пространства памяти недостаточно, то можно, кроме того, задействовать неиспользованное пространство Flash-памяти программ. Разумеется, в этом случае нужно быть очень осторожным, если программа еще должна расширяться. Также может случиться, что при программировании не было обращено внимание на допущенные ошибки. Данные основной программы могут быть случайно стерты. В таком случае PIC-контроллер окажется неработоспособным, и должен будет вновь программироваться — на этот раз уже безошибочной программой.

Таким образом, лучшей альтернативой расширения EEPROM-памяти данных является добавление внешней EEPROM-памяти, в которую будут заноситься данные. Как подключить внешнюю EEPROM-память и управлять ею, объясняется далее в последующих главах. Исходя из этого, применение памяти программ для хранения данных должно быть тщательно продумано и учтены все возможные затраты. Так внешняя 32-килобитная EEPROM-память стоит примерно 0,40 € (≈17 руб.) за штуку. Поэтому в дальнейшем обсуждается только обмен данными с внутренней EEPROM-памятью. Разумеется, запись Flash-памяти программ предоставляет еще преимущество и в том, что программу при необходимости можно изменить. Таким образом, микропрограммное обеспечение, которое загружается через последовательный интерфейс, возможно совершенствовать.

Следующая блок-схема алгоритма (рис. 1.8) описывает процесс записи данных в EEPROM-память по определенному адресу. Операция записи в EEPROM-память по сравнению с оперативной памятью (RAM-памятью) длится гораздо дольше. Так для слова данных (одного байта) требуется примерно от 4 до 8 мс. При тактовом сигнале процессора 4 МГц это соответствует обработке 4000 команд. К счастью, процесс записи протекает в фоновом

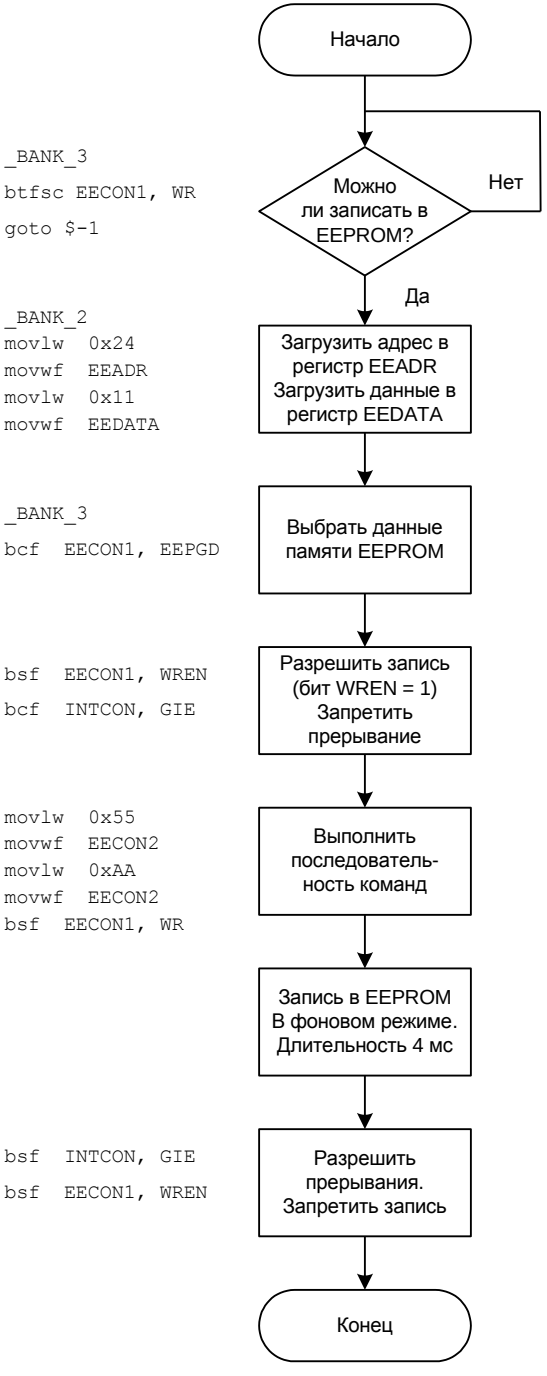

Рис. 1.8. Запись во внутреннюю EEPROM-память

режиме и не требуется ждать его окончания для продолжения работы основной программы. Длительность процесса записи также показывает, что нельзя использовать EEPROM-память для часто изменяющихся данных. Поскольку EEPROM-память подлежит старению, то имеется ограничение максимального количества записей в нее. Если производитель указывает количество минимум 100000 циклов стирания/записи, то это значит, что при сохранении каждую секунду нового значения в EEPROM-памяти, она будет непригодна уже примерно через 28 часов. Следует заметить, что для чтения EEPROM-памяти требуется то же самое время, как и для чтения при косвенной адресации.

Для записи EEPROM-памяти данных требуются только 4 из 6 специализированных регистров.

- EECON1 — регистр для управления записью в память EEPROM;
- EECON2 — регистр для запуска непосредственно записи;
- EEADR — регистр для хранения адреса для записи или чтения;
- EEDATA — регистр для данных, которые должны писаться в память EEPROM.

Регистры EEADRH и EEDATH дополнительно требуются для обращения к Flash-памяти программ. Это происходит потому, что для адресации к этой памяти требуется количество битов большее 8, т. е. обычной разрядности регистров специального назначения.

Как можно видеть при выполнении программы, чтобы выполнить процесс записи, нужно придерживаться специальной последовательности команд. Эта последовательность должна соблюдаться, т. к. иначе EEPROM-память не будет перезаписана. Также нужно обращать внимание на то, что во время выполнения этой последовательности прерывания микроконтроллера были отключены, поскольку иначе с передачей управления в программу обработки прерывания требуемая последовательность будет не соблюдена.

Чтение из EEPROM-памяти проще. На следующей блок-схеме алгоритма (рис. 1.9) представлены немногочисленные шаги, которые требуются для операции чтения.

Поскольку EEPROM-память данных это относительно медленная память, которая должна перезаписываться очень редко, то она хорошо подходит только для сохранения эталонных данных или для последовательностей значений. Поэтому объем равный 256 байтам для большинства применений также вполне достаточен.

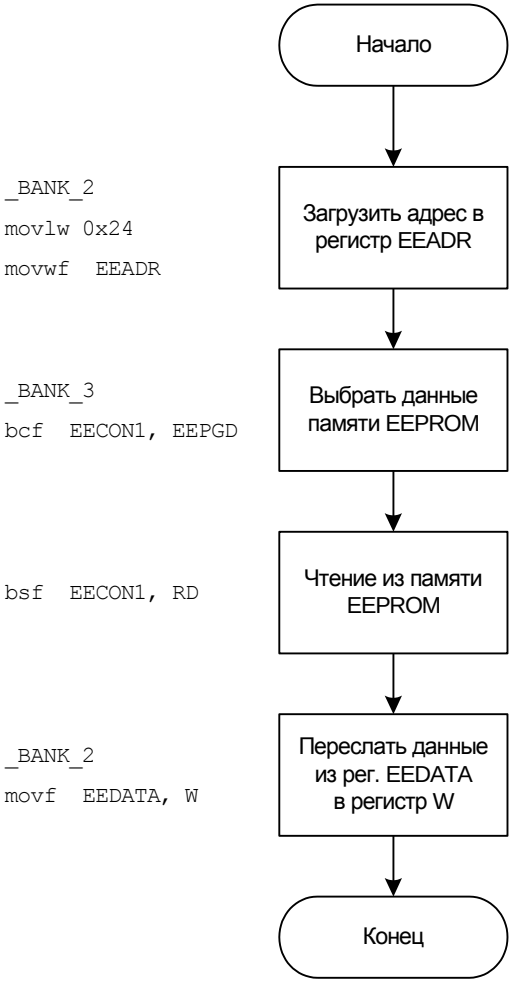

Рис. 1.9. Чтение из внутренней EEPROM-памяти