Лекция №1 «Введение: микроконтроллеры серии PIC и AVR»

Основные особенности микроконтроллеров серии PIC:

- состав и назначение семейств PIC-контроллеров
- микроконтроллеры семейств PIC16CXXX и PIC17CXXX
- особенности архитектуры микроконтроллеров семейства PIC16CXXX

Микроконтроллеры подгруппы PIC16F8X:

- основные характеристики
- особенности архитектуры

I. Общие сведения о микроконтроллерах серии PIC.

1.1. Состав и назначение семейств PIC-контроллеров

Микроконтроллеры семейств PIC (Peripheral Interface Controller) компании Microchip объединяют все передовые технологии микроконтроллеров: электрически программируемые пользователем ППЗУ, минимальное энергопотребление, высокую производительность, хорошо развитую RISC-архитектуру, функциональную законченность и минимальные размеры. Широкая номенклатура изделий обеспечивает использование микроконтроллеров в устройствах, предназначенных для разнообразных сфер применения.

Первые микроконтроллеры компании Microchip PIC16C5x появились в конце 1980-х годов и благодаря своей высокой производительности и низкой стоимости составили серьезную конкуренцию производившимся в то время 8-разрядным МК с CISC-архитектурой.

Высокая скорость выполнения команд в PIC-контроллерах достигается за счет использования двухшинной гарвардской архитектуры вместо традиционной одношинной фон-неймановской. Гарвардская архитектура основывается на наборе регистров с разделенными шинами и адресными пространствами для команд и данных. Все ресурсы микроконтроллера, такие как порты ввода/вывода, ячейки памяти и таймер, представляют собой физически реализованные аппаратные регистры.

Микроконтроллеры PIC содержат RISC-процессор с симметричной системой команд, позволяющей выполнять операции с любым регистром, используя произвольный метод адресации. Пользователь может сохранять результат операции в самом регистре-аккумуляторе или во втором регистре, используемом для операции.

В настоящее время компания Microchip выпускает пять основных семейств 8-разрядных RISC-микроконтроллеров, совместимых снизу вверх по программному коду:

- PIC12CXXX – семейство микроконтроллеров, выпускаемых в миниатюрном 8-выводном исполнении. Эти микроконтроллеры выпускаются как с 12-разрядной (33 команды), так и с 14-разрядной (35 команд) системой команд. Содержат встроенный тактовый генератор, таймер/счетчик, сторожевой таймер, схему управления прерываниями. В составе семейства есть микроконтроллеры со встроенным 8-разрядным

четырехканальным АЦП. Способны работать при напряжении питания до 2,5 В;

- PIC16C5X – базовое семейство микроконтроллеров с 12-разрядными командами (33 команды), выпускаемое в 18-, 20- и 28-выводных корпусах. Представляют собой простые недорогие микроконтроллеры с минимальной периферией. Способность работать при малом напряжении питания (до 2 В) делает их удобными для применения в переносных конструкциях. В состав семейства входят микроконтроллеры подгруппы PIC16HV5XX, способные работать непосредственно от батареи в диапазоне питающих напряжений до 15 В;
- PIC16CXXX – семейство микроконтроллеров среднего уровня с 14-разрядными командами (35 команд). Наиболее многочисленное семейство, объединяющее микроконтроллеры с разнообразными периферийными устройствами, в число которых входят аналоговые компараторы, аналогово-цифровые преобразователи, контроллеры последовательных интерфейсов SPI, USART и I2C, таймеры-счетчики, модули захвата/сравнения, широтно-импульсные модуляторы, сторожевые таймеры, супервизорные схемы и так далее;
- PIC17CXXX – семейство высокопроизводительных микроконтроллеров с расширенной системой команд 16-разрядного формата (58 команд), работающие на частоте до 33 МГц, с объемом памяти программ до 16 Кслов. Кроме обширной периферии, 16-уровневого аппаратного стека и векторной системы прерываний, почти все микроконтроллеры этого семейства имеют встроенный аппаратный умножитель 8x8, выполняющий операцию умножения за один машинный цикл. Являются одними из самых быстродействующих в классе 8-разрядных микроконтроллеров;
- PIC18CXXX – семейство высокопроизводительных микроконтроллеров с расширенной системой команд 16-разрядного формата (75 команд) и встроенным 10-разрядным АЦП, работающие на частоте до 40 МГц. Содержат 31-уровневый аппаратный стек, встроенную память команд до 32 Кслов и способны адресовать до 4 Кбайт памяти данных и до 2 Мбайт внешней памяти программ. Расширенное RISC-ядро микроконтроллеров данного семейства оптимизировано под использование нового Си-компилятора.

Большинство PIC-контроллеров выпускаются с однократно программируемой памятью программ (OTP), с возможностью внутрисхемного программирования или масочным ПЗУ. Для целей отладки предлагаются более дорогие версии с ультрафиолетовым стиранием и Flash-памятью. Полный список выпускаемых модификаций PIC-контроллеров включает порядка пятисот наименований. Поэтому продукция компании перекрывает почти весь диапазон применений 8-разрядных микроконтроллеров.

Из программных средств отладки наиболее известны и доступны различные версии ассемблеров, а также интегрированная программная среда

MPLAB. Российские производители программаторов и аппаратных отладочных средств также уделяют внимание PIC-контроллерам. Выпускаются как специализированные программаторы, такие как PICPROG, программирующие почти весь спектр PIC-микроконтроллеров, так и универсальные: UNIPRO и СТЕРХ, поддерживающие наиболее известные версии PIC-контроллеров.

Наиболее распространенными семействами PIC-контроллеров являются PIC16CXXX и PIC17CXXX.

1.2. Микроконтроллеры семейств PIC16CXXX и PIC17CXXX

Основным назначением микроконтроллеров семейств PIC16 и PIC17, как следует из аббревиатуры PIC (Peripheral Interface Controller), является выполнение интерфейсных функций. Этим объясняются особенности их архитектуры:

- RISC-система команд, характеризующаяся малым набором одноадресных инструкций (33, 35 или 58), каждая из которых имеет длину в одно слово (12, 14 или 16 бит) и большинство выполняется за один машинный цикл. В системе команд отсутствуют сложные арифметические команды (умножение, деление), предельно сокращен набор условных переходов;
- высокая скорость выполнения команд: при тактовой частоте 20 МГц время машинного цикла составляет 200 нс (быстродействие равно 5 млн. операций/сек);
- особая организация ввода-вывода т.е. на линиях портов ввода/вывода, имеется возможность подключать довольно мощную нагрузку (до 25 мА), например, светодиоды.
- низкая потребляемая мощность;
- ориентация на ценовую нишу предельно низкой стоимости, определяющая использование дешевых корпусов с малым количеством выводов (8, 14, 18, 28), отказ от внешних шин адреса и данных (кроме PIC17C4X), использование упрощенного механизма прерываний и аппаратного (программно недоступного) стека.

1.3. Особенности архитектуры микроконтроллеров семейства PIC16CXXX

Микроконтроллеры семейства PIC16CXXX, выполненные по технологии HCMOS представляют собой 8-разрядные микроконтроллеры на основе RISC-процессора, выполненные по гарвардской архитектуре. Имеют встроенное ПЗУ команд объемом от 0,5 до 4 Кслов (разрядность слова команд равна 12 – 14 бит). Память данных PIC-контроллеров организована в виде регистрового файла объемом 32 – 128 байт, в котором от 7 до 16 регистров отведено для управления системой и обмена данными с внешними устройствами.

Одним из основных достоинств этих устройств является очень широкий диапазон напряжений питания (2 – 6 В). Ток потребления на частоте 32768 Гц составляет менее 15 мкА, на частоте 4 МГц – 1 – 2 мА, на частоте 20 МГц 5 – 7 мА и в режиме микропотребления (режим SLEEP) – 1 – 2 мкА.

Выпускаются модификации для работы в трех температурных диапазонах: от 0 до +70°С, от -40 до +85°С и от -40 до +125°С.

Каждый из контроллеров содержит универсальные (от 1 до 3) и сторожевой таймеры, а также надежную встроенную систему сброса при включении питания. Частота внутреннего тактового генератора задается либо кварцевым резонатором, либо RC-цепочкой в диапазоне 0 – 25 МГц. PIC-контроллеры имеют от 12 до 33 линий цифрового ввода-вывода, причем каждая из них может быть независимо настроена на ввод или вывод.

В устройство PIC16C64 входит широтно-импульсный модулятор, с помощью которого можно реализовать ЦАП с разрешением до 16 разрядов. Здесь есть и последовательный двунаправленный синхронно-асинхронный порт, обеспечивающий возможность организации шины I^2C. Приборы PIC16C71 и PIC16C74 содержат встроенный многоканальный 8-разрядный АЦП с устройством выборки-хранения.

Помимо памяти программ в PIC предусмотрено несколько индивидуально прожигаемых перемычек, с помощью которых можно на этапе программирования кристалла выбрать тип тактового генератора, отключить сторожевой таймер или систему сброса, включить защиту памяти программ от копирования, а также записать серийный номер кристалла (16 бит).

С программной точки зрения PIC-контроллер представляет собой 8-разрядный RISC-процессор с гарвардской архитектурой. Число команд небольшое — от 33 до 35. Все команды имеют одинаковую длину и, кроме команд ветвления, выполняются за четыре периода тактовой частоты (в отличие, например, от 12 периодов для i87C51). Поддерживаются непосредственный, косвенный и относительный методы адресации, можно эффективно управлять отдельными битами в пределах всего регистрового файла. Стек реализован аппаратно. Его максимальная глубина составляет два или восемь уровней в зависимости от типа контроллера. Почти во всех микросхемах PIC есть система прерываний, источниками которых могут быть таймер и внешние сигналы. Система команд практически симметрична и, как следствие, легка в освоении.

Применение PIC-контроллеров целесообразно в несложных приборах с ограниченным током потребления (автономные устройства, приборы с питанием от телефонной линии и т.п.). Благодаря малому количеству компонентов, используемых при построении таких приборов, их размеры уменьшаются, а надежность увеличивается.

Типичным представителем микроконтроллеров семейства PIC16CXXX являются микроконтроллеры подгруппы PIC16F8X.

II. Архитектура и характеристики PIC16F8X.

2.1. Основные характеристики

Микроконтроллеры подгруппы PIC16F8X относятся к семейству 8-разрядных КМОП микроконтроллеров группы PIC16CXXX, для которых характерны низкая стоимость, полностью статическая КМОП-технология и высокая производительность.

В состав подгруппы входят МК PIC16F83, PIC16CR83, PIC16F84 и PIC16CR84. Основные характеристики МК подгруппы PIC16F8X приведены в табл. 1.1.

Таблица 1.1. Основные характеристики МК подгруппы PIC16F8X.

Параметр	**PIC16F83**	**PIC16CR83**	**PIC16F84**	**PIC16CR84**
Максимальная частота, МГц	10	10	10	10
Flash-память программ, слов	512	-	1K	-
ПЗУ программ, слов	-	512	-	1K
Память данных, байт	36	36	68	68
Память данных в РПЗУ (EEPROM), байт	64	64	64	64
Таймеры	TMR0	TMR0	TMR0	TMR0
Число источников прерываний	4	4	4	4
Число линий ввода/вывода	13	13	13	13
Диапазон напряжений питания, В	2,0 – 6,0	2,0 – 6,0	2,0 – 6,0	2,0 – 6,0
Число выводов и тип корпуса	18 DIP, SOIC	18 DIP, SOIC	18 DIP, SOIC	18 DIP, SOIC

Все микроконтроллеры подгруппы PIC16F8X используют гарвардскую архитектуру с RISC-процессором, обладающую следующими основными особенностями:

- используются только 35 простых команд;
- все команды выполняются за один цикл (400 нс при частоте 10 МГц), кроме команд перехода, которые требуют 2 цикла;
- рабочая частота 0 Гц ... 10 МГц;
- раздельные шины данных (8 бит) и команд (14 бит);
- 512 x 14 или 1024 x 14 память программ, выполненная на ПЗУ или электрически перепрограммируемой Flash- памяти;
- 15 восьмиразрядных регистров специальных функций (SFR);
- восьмиуровневый аппаратный стек;
- прямая, косвенная и относительная адресация данных и команд;
- 36 или 68 восьмиразрядных регистров общего назначения (GPR) или ОЗУ;
- четыре источника прерывания:
  - внешний вход RB0/INT;
  - переполнение таймера TMR0;
  - изменение сигналов на линиях порта B;
  - завершение записи данных в память EEPROM;
- 64 x 8 электрически перепрограммируемая EEPROM память данных с возможностью выполнения 1000000 циклов стирания/записи;
- сохранение данных в EEPROM в течение как минимум 40 лет.

Микроконтроллеры подгруппы PIC16F8X обладают развитыми возможностями ввода/вывода:

- 13 линий ввода-вывода с индивидуальной установкой направления обмена;

- высокий втекающий/вытекающий ток, достаточный для управления светодиодами:
- максимальный втекающий ток – 25 мА;
- максимальный вытекающий ток – 20 мА;
- 8-битный таймер/счетчик TMR0 с 8-битным программируемым предварительным делителем.

Специализированные микроконтроллерные функции включают следующие возможности:

- автоматический сброс при включении (Power-on-Reset);
- таймер включения при сбросе (Power-up Timer);
- таймер запуска генератора (Oscillator Start-up Timer);
- сторожевой (Watchdog) таймер WDT с собственным встроенным генератором, обеспечивающим повышенную надежность;
- EEPROM бит секретности для защиты кода;
- экономичный режим SLEEP;
- выбираемые пользователем биты для установки режима возбуждения встроенного генератора;
- последовательное встроенное устройство программирования Flash/EEPROM памяти программ и данных с использованием только двух выводов.

КМОП технология обеспечивает МК подгруппы PIC16F8X дополнительные преимущества:

- статический принцип работы;
- широкий диапазон напряжений питания: 2,0 ... 6,0 В;
- низкое энергопотребление:
- менее 2 мА при 5В и 4МГц;
- порядка 15 мкА при 2В и 32КГц;
- менее 1 мкА для SLEEP-режима при 2В.

Микроконтроллеры подгруппы PIC16F8X различаются между собой только объемом ОЗУ данных, а также объемом и типом памяти программ. Наличие в составе подгруппы МК с Flash-памятью программ облегчает создание и отладку прототипов промышленных образцов изделий.

2.2. Особенности архитектуры

Упрощенная структурная схема МК подгруппы PIC16F8X приведена на рис. 1.1.

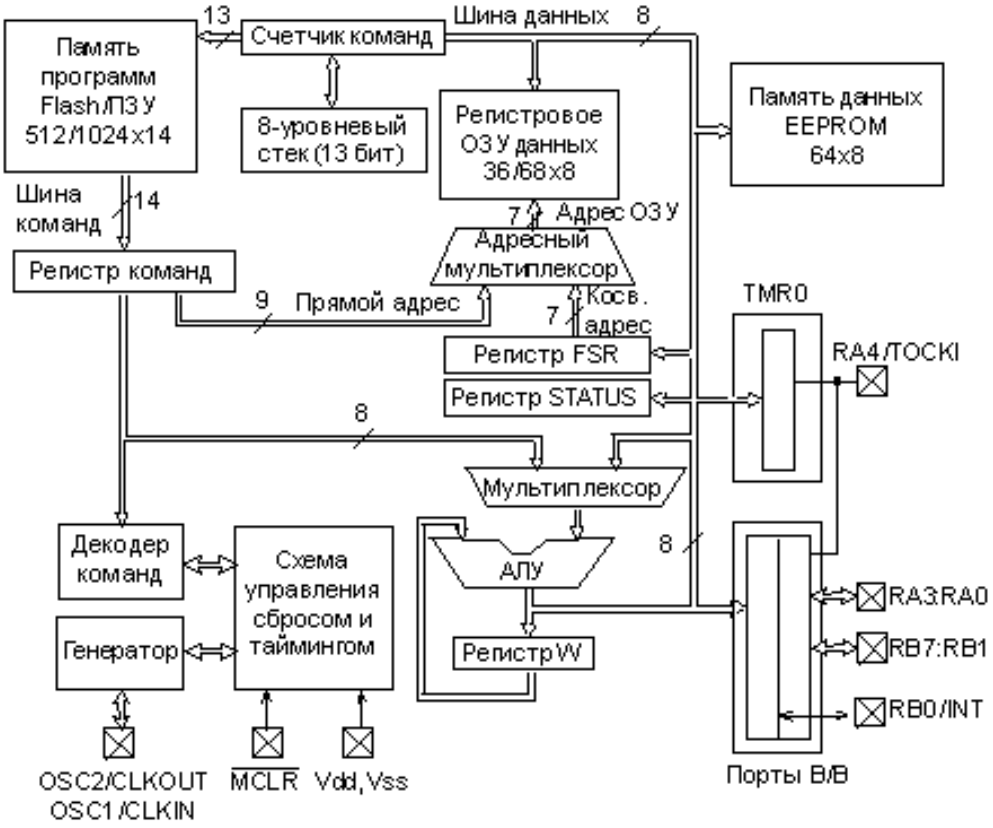

Рис. 1.1. Структурная схема МК подгруппы PIC16F8X.

Архитектура основана на концепции раздельных шин и областей памяти для данных и для команд (гарвардская архитектура). Шина данных и память данных (ОЗУ) – имеют ширину 8 бит, а программная шина и программная память (ПЗУ) имеют ширину 14 бит. Такая концепция обеспечивает простую, но мощную систему команд, разработанную так, что битовые, байтовые и регистровые операции работают с высокой скоростью и с перекрытием по времени выборок команд и циклов выполнения. 14-битовая ширина программной памяти обеспечивает выборку 14-битовой команды в один цикл. Двухступенчатый конвейер обеспечивает одновременную выборку и исполнение команды. Все команды выполняются за один цикл, исключая команды переходов.

Микроконтроллеры PIC16F83 и PIC16CR83 адресуют 512x14 памяти программ, а PIC16F84 и PIC16CR84 – 1Kx14 памяти программ. Вся память программ является внутренней.

Микроконтроллер может прямо или косвенно обращаться к регистрам или памяти данных. Все регистры специальных функций, включая счетчик команд, отображаются на память данных. Ортогональная (симметричная) система команд позволяет выполнять любую команду над любым регистром с использованием произвольного метода адресации. Ортогональная архитектура и отсутствие специальных исключений делает программирование МК группы PIC16F8X простым и эффективным.

Назначение выводов МК подгруппы PIC16F8X приведено в табл. 1.2.

Таблица 1.2. Назначение выводов МК подгруппы PIC16F8X

Обозначение	**Тип**	**Буфер**	**Описание**
OSC1/CLKIN	I	ТШ/КМОП	Вход кристалла генератора, RC-цепочки или вход внешнего тактового сигнала
OSC2/CLKOUT	O	-	Выход кристалла генератора. В RC-режиме – выход 1/4 частоты OSC1
/MCLR	I/P	ТШ	Сигнал сброса/вход программирующего напряжения. Сброс низким уровнем.
RA0	I/O	ТТЛ	PORTA – двунаправленный порт ввода/вывода RA4/T0CKI может быть выбран как тактовый вход таймера/счетчика TMR0. Выход с открытым стоком.
RA1	I/O	ТТЛ	
RA2	I/O	ТТЛ	
RA3	I/O	ТТЛ	
RA4	I/O	ТШ	
/T0CKI			
RB0/INT	I/O	ТТЛ/ТШ	PORTB – двунаправленный порт ввода/вывода. Может быть запрограммирован в режиме внутренних активных нагрузок на линию питания по всем выводам. Вывод RB0/INT может быть выбран как внешний вход прерывания. Выводы RB4...RB7 могут быть программно настроены как входы прерывания по изменению состояния на любом из входов. При программировании МК RB6 используется как тактовый, а RB7 как вход/выход данных.
RB1	I/O	ТТЛ	
RB2	I/O	ТТЛ	
RB3	I/O	ТТЛ	
RB4	I/O	ТТЛ	
RB5	I/O	ТТЛ	
RB6	I/O	ТТЛ/ТШ	
RB7	I/O	ТТЛ/ТШ	
Vdd	P	-	Положительное напряжение питания
Vss	P	-	Общий провод (земля)
В таблице использованы следующие обозначения: I — вход; O — выход; I/O — вход/выход; P — питание; "–" — не используется; ТТЛ — ТТЛ вход; ТШ — вход триггера Шмитта.			

Микроконтроллер содержит 8-разрядное АЛУ и рабочий регистр W. АЛУ является арифметическим модулем общего назначения и выполняет арифметические и логические функции над содержимым рабочего регистра и любого из регистров контроллера. АЛУ может выполнять операции сложения, вычитания, сдвига и логические операции. Если не указано иное, то арифметические операции выполняются в дополнительном двоичном коде.
В зависимости от результата операции, АЛУ может изменять значения бит регистра STATUS: C (Carry), DC (Digit carry) и Z (Zero).

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

Микроконтроллеры подгруппы PIC16F8X:

- схема тактирования и цикл выполнения команды
- организация памяти программ и стека
- организация памяти данных
- регистры специального назначения
- счетчик команд
- прямая и косвенная адресации

I. Принципы работы (временная диаграмма тактирования и циклов выполнения программы).

Входная тактовая частота, поступающая с вывода OSC1/CLKIN, делится внутри на четыре, и из нее формируются четыре циклические не перекрывающиеся 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. Схема тактирования и выполнения команды изображена на рис. 2.1.

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 GOTO), то для ее выполнения потребуется два цикла, как показано на рис. 2.2.

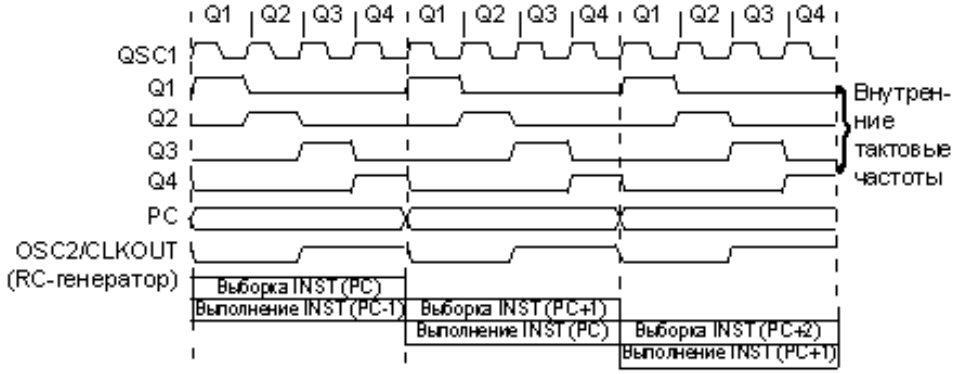

Рис. 2.1. Схема тактирования и выполнения команды.

Рис. 2.2. Выборка команд.

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.

II. Организация памяти.

2.1. Организация памяти программ и стека

Счетчик команд в МК PIC16F8X имеет ширину 13 бит и способен адресовать 8Кх14бит объема программной памяти. Однако физически на кристаллах PIC16F83 и PIC16CR83 имеется только 512x14 памяти (адреса 0000h-01FFh), а в МК PIC16F84 и PIC16CR84 – 1Кх14 памяти (адреса 0000h-03FFh). Обращение к адресам выше 1FFh (3FFh) фактически есть адресация в те же первые 512 адресов (первые 1К адресов).

Организация памяти программ и стека приведена на рис. 2.3.

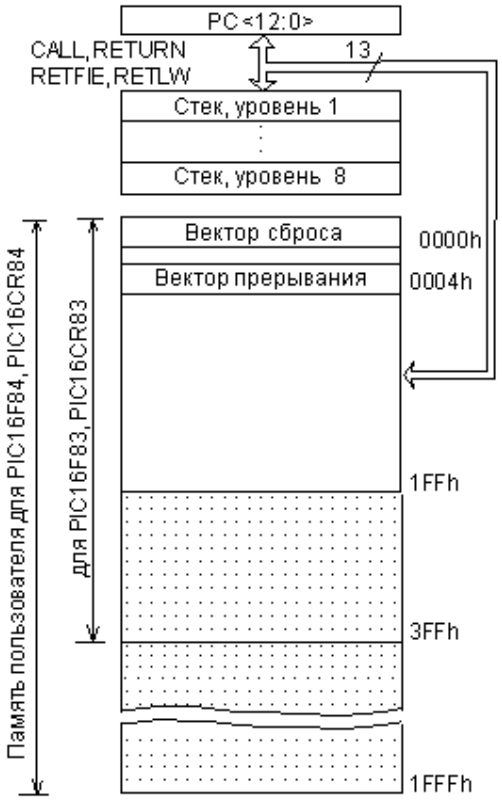

Рис. 2.3. Организация памяти программ и стека.

В памяти программ есть выделенные адреса. Вектор сброса находится по адресу 0000h, вектор прерывания – по адресу 0004h. Обычно по адресу 0004h располагается подпрограмма идентификации и обработки прерываний, а по адресу 0000h – команда перехода на метку, расположенную за подпрограммой обработки прерываний.

2.2. Организация памяти данных

Память данных МК разбита на две области. Первые 12 адресов – это область регистров специальных функций (SFR), а вторая – область регистров общего назначения (GPR). Область SFR управляет работой прибора.

Обе области разбиты в свою очередь на банки 0 и 1. Банк 0 выбирается обнулением бита RP0 регистра статуса (STATUS). Установка бита RP0 в единицу выбирает банк 1. Каждый банк имеет протяженность 128 байт. Однако для PIC16F83 и PIC16CR83 память данных существует только до адреса 02Fh, а для PIC16F84 и PIC16CR84 – до адреса 04Fh.

На рис. 2.4 изображена организация памяти данных.

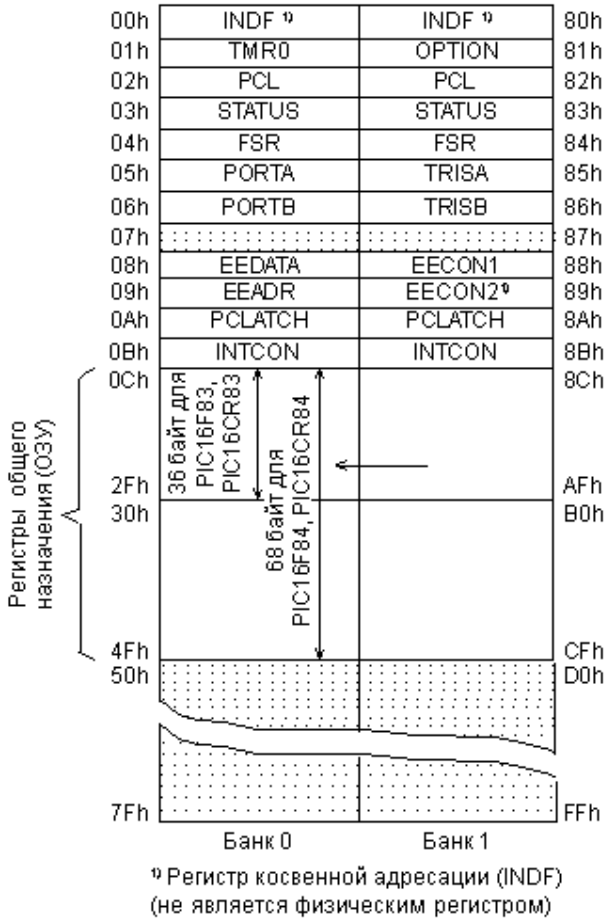

Рис. 2.4. Организация памяти данных.

Некоторые регистры специального назначения продублированы в обоих банках, а некоторые расположены в банке 1 отдельно.

Регистры с адресами 0Ch-4Fh могут использоваться как регистры общего назначения, которые представляют собой статическое ОЗУ. Адреса регистров общего назначения банка 1 отображаются на банк 0. Следовательно, когда установлен банк 1, то обращение к адресам 8Ch-CFh фактически адресует банк 0.

В регистре статуса помимо бита RP0 есть еще бит RB1, что позволяет обращаться к четырем страницам (банкам) будущих модификаций этого кристалла.

К ячейкам ОЗУ можно адресоваться прямо, используя абсолютный адрес каждого регистра, или косвенно, через регистр указатель FSR. Косвенная адресация использует текущее значение разрядов RP1:RP0 для доступа к банкам. Это относится и к EEPROM памяти данных. В обоих случаях можно адресовать до 512 регистров.

2.3. Регистры специального назначения

Регистр статуса (STATUS) содержит признаки операции (арифметические флаги) АЛУ, состояние контроллера при сбросе и биты выбора страниц для памяти данных. Назначение бит регистра приведено в табл. 2.1.

Таблица 2.1. Назначение бит регистра STATUS (адрес 03h, 83h).

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	/TO	/PD	Z	DC	C
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
Бит 7: IRP: бит выбора страницы банка данных (используется при косвенной адресации) 0 = банк 0,1 (00h – FFh) 1 = банк 2,3 (100h – 1FFh) Бит IRP не используется в МК подгруппы PIC16F8X							
Биты 6-5: RP1:RP0: биты выбора страницы банка данных (используются при прямой адресации) 00 = банк 0 (00h – 7Fh) 01 = банк 1 (80h – FFh) 10 = банк 2 (100h – 17Fh) 11 = банк 3 (180h – 1FFh) В МК подгруппы PIC16F8X используется только бит RP0							
Бит 4: /TO: бит срабатывания сторожевого таймера 1 = после включения питания, а также командами CLRWDT и SLEEP 0 = по завершении выдержки сторожевого таймера							
Бит 3: /PD: бит снижения потребляемой мощности 1 = после включения питания, а также командой CLRWDT 0 = по команде SLEEP							
Бит 2: Z: бит нулевого результата 1 = результат арифметической или логической операции нулевой 0 = результат арифметической или логической операции ненулевой							
Бит 1: DC: бит десятичного переноса/заема (для команд ADDWF и ADDLW) 1 = имеет место перенос из 4-го разряда 0 = нет переноса из 4-го разряда							
Бит 0: C: бит переноса/заема (для команд ADDWF и ADDLW) 1 = имеет место перенос из самого старшего разряда 0 = нет переноса из самого старшего разряда							
Примечание: вычитание осуществляется путем прибавления дополнительного кода второго операнда. При выполнении команд сдвига этот бит загружается из младшего или старшего разряда сдвигаемого источника.							

Здесь и далее: R — читаемый бит; W — записываемый бит; S — устанавливаемый бит; U — неиспользуемый бит (читается как "0"); -n = 0 или 1 — значение бита после сброса.

Регистр статуса доступен для любой команды так же, как любой другой регистр. Однако если регистр STATUS является регистром назначения для

команды, влияющей на биты Z, DC или C, то запись в эти три бита запрещается. Кроме того, биты /TO и /PD устанавливаются аппаратно и не могут быть записаны в статус программно. Это следует иметь в виду при выполнении команды с использованием регистра статуса. Например, команда CLRF STATUS обнулит все биты, кроме битов /TO и /PD, а затем установит бит Z=1. После выполнения этой команды регистр статуса может и не иметь нулевого значения (из-за битов /TO и /PD) STATUS=000uu1uu, где u – неизменяемое состояние. Поэтому рекомендуется для изменения регистра статуса использовать только команды битовой установки BCF, BSF, MOVWF, которые не изменяют остальные биты статуса. Воздействие всех команд на биты статуса рассматривается в разделе "Описание системы команд".

Регистр конфигурации (OPTION) является доступным по чтению и записи регистром, который содержит управляющие биты для конфигурации предварительного делителя (пределителя), внешних прерываний, таймера, а также резисторов "pull-up" на выводах PORTB. Назначение бит регистра приведено в табл. 2.2.

Таблица 2.2. Назначение бит регистра OPTION (адрес 81h).

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0

Бит 7: /RBPU: бит установки резисторов "pull-up" на выводах PORTB
0 = резисторы "pull-up" подключены
1 = резисторы "pull-up" отключены

Бит 6: INTEDG: бит выбора перехода сигнала прерывания
0 = прерывание по спаду сигнала на выводе RB0/INT
1 = прерывание по фронту сигнала на выводе RB0/INT

Бит 5: T0CS: бит выбора источника сигнала таймера TMR0
0 = внутренний тактовый сигнал (CLKOUT)
1 = переход на выводе RA4/T0CKI

Бит 4: T0SE: бит выбора перехода источника сигнала для TMR0
0 = приращение по фронту сигнала на выводе RA4/T0CKI
1 = приращение по спаду сигнала на выводе RA4/T0CKI

Бит 3: PSA: бит назначения пределителя
0 = предделитель подключен к TMR0
1 = предделитель подключен к сторожевому таймеру WDT

Биты 2-0: PS2:PS0: биты выбора коэффициента деления пределителя

Значения бит	**Скорость TMR0**	**Скорость 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**

В том случае, когда предделитель обслуживает сторожевой таймер WDT, таймеру TMR0 назначается коэффициент предварительного деления

1:1. Регистр условий прерывания (INTCON) является доступным по чтению и записи регистром, который содержит биты доступа для всех источников прерываний. Назначение бит регистра приведено в табл. 2.3.

Таблица 2.3. Назначение бит регистра INTCON (адреса 0Bh, 8Bh).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
Бит 7: GIE: бит разрешения всех прерываний 0 = запрещены все прерывания 1 = разрешены все незамаскированные прерывания							
Бит 6: EEIE: бит разрешения прерывания записи в EEPROM 0 = запрещены прерывания записи в EEPROM 1 = разрешены прерывания записи в EEPROM							
Бит 5: T0IE: бит разрешения прерывания по переполнению TMR0 0 = запрещены прерывания от TMR0 1 = разрешены прерывания от TMR0							
Бит 4: INTE: бит разрешения прерываний по входу RB0/INT 0 = запрещены прерывания по входу RB0/INT 1 = разрешены прерывания по входу RB0/INT							
Бит 3: RBIE: бит разрешения прерываний по изменению PORTB 0 = запрещены прерывания по изменению PORTB 1 = разрешены прерывания по изменению PORTB							
Бит 2: T0IF: бит запроса прерывания по переполнению TMR0 0 = прерывание по переполнению TMR0 отсутствует 1 = прерывание по переполнению TMR0 имеет место							
Бит 1: INTF: бит запроса прерывания по входу RB0/INT 0 = прерывание по входу RB0/INT отсутствует 1 = прерывание по входу RB0/INT имеет место							
Бит 0: RBTF: бит запроса прерывания по изменению PORTB 0 = ни на одном из входов RB7:RB4 состояние не изменилось 1 = хотя бы на одном из входов RB7:RB4 изменилось состояние							

Бит разрешения всех прерываний GIE сбрасывается автоматически при следующих обстоятельствах:

- по включению питания;
- по внешнему сигналу /MCLR при нормальной работе;
- по внешнему сигналу /MCLR в режиме SLEEP;
- по окончанию задержки таймера WDT при нормальной работе;
- по окончанию задержки таймера WDT в режиме SLEEP.

Прерывание INT может вывести процессор из режима SLEEP, если перед входом в этот режим бит INTE был установлен в единицу. Состояние бита GIE также определяет: будет ли процессор переходить на подпрограмму прерывания после выхода из режима SLEEP.

Сброс битов – запросов прерываний – должен осуществляться соответствующей программой обработки.

III. Особенности выполнения команд.

3.1. Выборка команд

Счетчик команд PCL и PCLATH имеет разрядность 13 бит. Младший байт счетчика (PCL) доступен для чтения и записи и находится в регистре 02h. Старший байт счетчика команд не может быть напрямую записан или считан и берется из регистра PCLATH (PC latch high), адрес которого 0Ah. Содержимое PCLATH передается в старший байт счетчика команд, когда он загружается новым значением.

В зависимости от того, загружается ли в счетчик команд новое значение во время выполнения команд CALL, GOTO, или в младший байт счетчика команд (PCL) производится запись, – старшие биты счетчика команд загружаются из PCLATH разными способами, как показано на рис. 2.5.

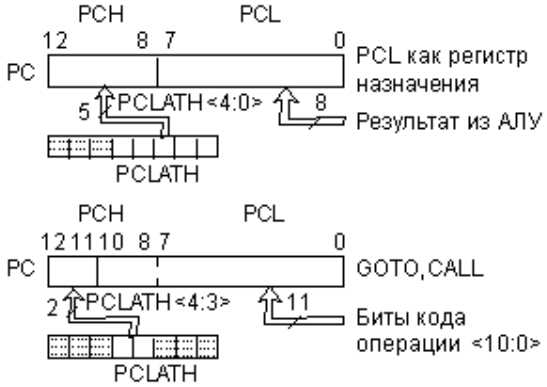

Рис. 2.5. Загрузка старших бит счетчика команд.

Команды CALL и GOTO оперируют 11-разрядным адресным диапазоном, достаточным для смещения в пределах страницы программной памяти объемом 2К слов. Для МК подгруппы PIC16F8X этого хватает. С целью обеспечения возможности расширения памяти команд для будущих моделей МК предусмотрена загрузка двух старших бит счетчика команд из регистра PCLATH<4:3>. При использовании команд CALL и GOTO пользователь должен убедиться в том, что эти страничные биты запрограммированы для выхода на нужную страницу. При выполнении команды CALL или выполнении прерывания весь 13-битный счетчик команд помещается в стек, поэтому для возвращения из подпрограммы не нужны манипуляции с разрядами PCLATH<4:3>.

Микроконтроллеры подгруппы PIC16F8X игнорируют значения бит PCLATH<4:3>, которые используются для обращения к страницам 1, 2 и 3 программной памяти. Однако применять биты PCLATH<4:3> в качестве ячеек памяти общего назначения не рекомендуется, так как это может повлиять на совместимость с будущими поколениями изделий.

Возможность выполнять арифметические операции непосредственно над счетчиком команд позволяет очень быстро и эффективно осуществлять табличные преобразования в PIC-контроллерах.

Микроконтроллеры подгруппы PIC16F8X имеют восьмиуровневый аппаратный стек шириной 13 бит (см. рис. 2.4). Область стека не принадлежит

ни к программной области, ни к области данных, а указатель стека пользователю недоступен. Текущее значение счетчика команд посылается в стек, когда выполняется команда CALL или производится обработка прерывания. При выполнении процедуры возврата из подпрограммы (команды RETLW, RETFIE или RETURN) содержимое счетчика команд восстанавливается из стека. Регистр PCLATH при операциях со стеком не изменяется.

Стек работает как циклический буфер. Следовательно, после того как стек был загружен 8 раз, девятая загрузка перепишет значение первой. Десятая загрузка перепишет вторую и т.д. Если стек был выгружен 9 раз, счетчик команд становится таким же, как после первой выгрузки.

Признаков положения стека в контроллере не предусмотрено, поэтому пользователь должен самостоятельно следить за уровнем вложения подпрограмм.

3.2. Прямая и косвенная адресации

Когда производится прямая 9-битная адресация, младшие 7 бит берутся как прямой адрес из кода операции, а два бита указателя страниц (RP1, RP0) из регистра статуса, как показано на рис. 2.6.

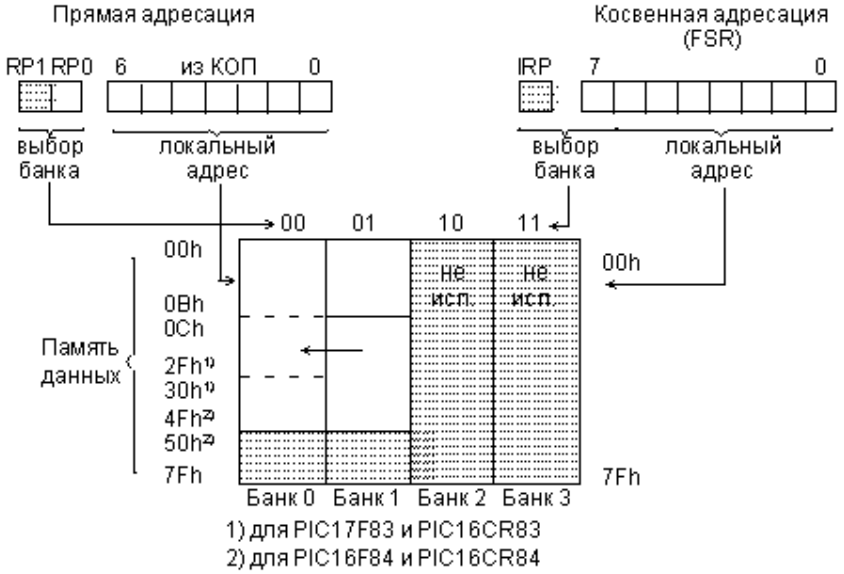

Рис. 2.6. Методы адресации данных.

Признаком косвенной адресации является обращение к регистру INDF. Любая команда, которая использует INDF (адрес 00h) в качестве регистра фактически обращается к указателю, который хранится в FSR (адрес 04h). Чтение косвенным образом самого регистра INDF даст результат 00h. Запись в регистр INDF косвенным образом будет выглядеть как NOP, но биты статуса могут быть изменены. Необходимый 9-битный адрес формируется объединением содержимого 8-битного FSR регистра и бита IRP из регистра статуса (см. рис. 2.6).

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

Микроконтроллеры подгруппы PIC16F8X:

- порты ввода/вывода
- модуль таймера и регистр таймера
- память данных в РПЗУ (EEPROM)
- организация прерываний

1.1. Порты ввода/вывода

Контроллеры подгруппы PIC16F8X имеют два порта: PORTA (5 бит) и PORTB (8 бит) с побитовой индивидуальной настройкой на ввод или на вывод.

Порт A (PORTA) представляет собой 5-битовый фиксатор, соответствующий выводам контроллера RA<4:0>. Линия RA4 имеет вход триггера Шмитта и выход с открытым стоком. Все остальные линии порта имеют ТТЛ входные уровни и КМОП выходные буферы. Адрес регистра порта A 05h. К каждой линии порта поставлен в соответствие бит направления передачи данных, который хранится в управляющем регистре TRISA, расположенном по адресу 85h. Если бит управляющего TRISA регистра имеет значение 1, то соответствующая линия будет устанавливаться на ввод. Ноль переключает линию на вывод и одновременно выводит на нее содержимое соответствующего регистра-фиксатора порта. При включении питания все линии порта по умолчанию настроены на ввод.

На рис. 3.1 дана схема линий RA<3:0> порта A.

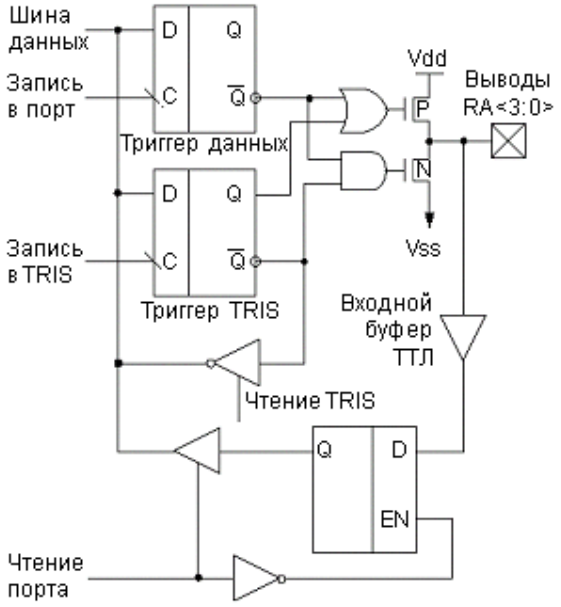

Рис. 3.1. Схема линий RA <3:0 порта A
(выводы порта имеют защитные диоды к Vdd и Vss)

Операция чтения порта А считывает состояние выводов порта, в то время как запись в него изменяет состояние триггеров порта. Все операции с портом являются операциями типа "чтение-модификация-запись". Поэтому запись в порт предполагает, что состояние выводов порта вначале считывается, затем модифицируется и записывается в триггер-фиксатор. Вывод RA4 мультиплексирован с тактовым входом таймера TMR0. Схема линии RA4 порта А приведена на рис. 3.2.

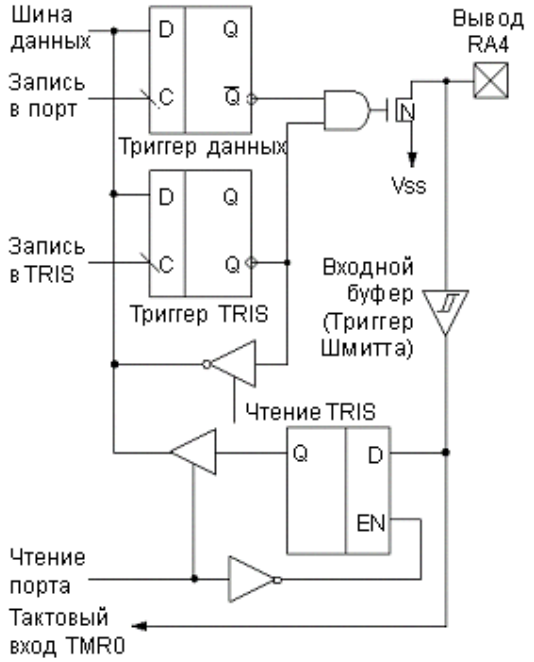

Рис. 3.2. Схема линии RA4 порта А.
(вывод порта имеет защитный диод только к Vss)

Порт В (PORTB) – это двунаправленный 8-битовый порт, соответствующий выводам RB<7:0> контроллера и расположенный по адресу 06h. Относящийся к порту В управляющий регистр TRISB расположен на первой странице регистров по адресу 86h. Если бит управляющего TRISB регистра имеет значение 1, то соответствующая линия будет устанавливаться на ввод. Ноль переключает линию на вывод и одновременно выводит на нее содержимое соответствующего регистра защелки. При включении питания все линии порта по умолчанию настроены на ввод.

У каждой ножки порта В имеется небольшая активная нагрузка (около 100мкА) на линию питания (pull-up). Она автоматически отключается, если эта ножка запрограммирована как вывод. Более того, управляющий бит /RBPU регистра OPTION<7> может отключить (при RBPU=1) все нагрузки. Сброс при включении питания также отключает все нагрузки.

Четыре линии порта В (RB<7:4>) могут вызвать прерывание при изменении значения сигнала на любой из них. Если эти линии настроены на ввод, то они опрашиваются и защелкиваются в цикле чтения Q1. Новая величина входного сигнала сравнивается со старой в каждом командном цикле. При

несовпадении значения сигнала на ножке и в фиксаторе генерируется высокий уровень. Выходы детекторов "несовпадений" RB4, RB5, RB6, RB7 объединяются по ИЛИ и генерируют прерывание RBIF (запоминаемое в регистре INTCON<0>). Любая линия, настроенная как вывод, в этом сравнении не участвует. Прерывание может вывести кристалл из режима SLEEP. В подпрограмме обработки прерывания следует сбросить запрос прерывания одним из следующих способов:

- прочитать (или записать в) порт B. Это завершит состояние сравнения;
- обнулить бит RBIF регистра INTCON<0>.

При этом необходимо иметь в виду, что условие "несовпадения" будет продолжать устанавливать признак RBIF. Только чтение порта B может устранить "несовпадение" и позволит обнулить бит RBIF.

Прерывание по несовпадению и программно устанавливаемые внутренние активные нагрузки на этих четырех линиях могут обеспечить простой интерфейс, например, с клавиатурой, с выходом из режима SLEEP по нажатию клавиш.

Схемы линий порта B приведены на рис. 3.3 и рис. 3.4.

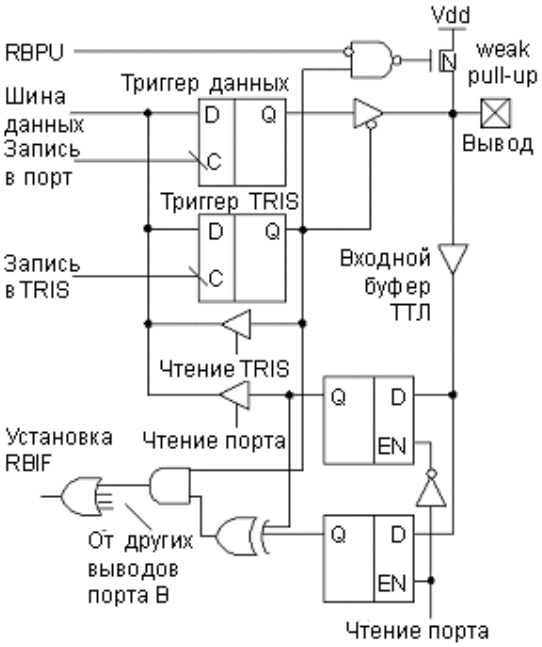

Рис. 3.3. Схема линий RB <7:4. B.

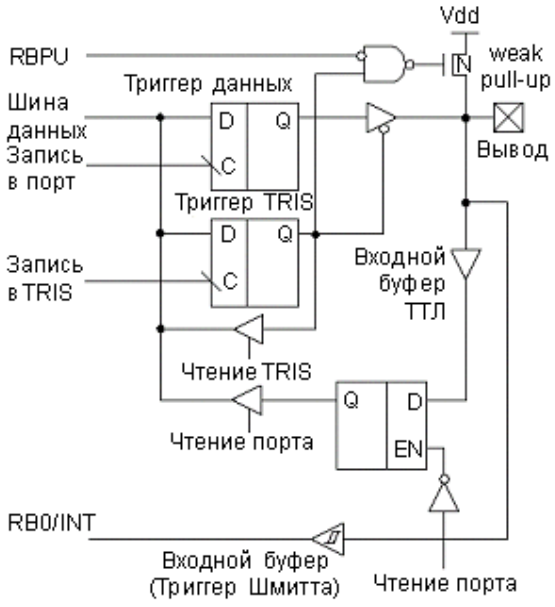

Рис. 3.4. Схема линий RB <3:0.

(выводы порта B имеют защитные диоды к Vdd и Vss).

При организации двунаправленных портов необходимо учитывать особенности организации ввода/вывода данных МК. Любая команда, которая осуществляет запись, выполняет ее внутри как "чтение-модификация-запись". Например, команды BCF и BSF считывают порт целиком, модифицируют один бит и выводят результат обратно. Здесь необходима осторожность. В частности, команда BSF PORTB, 5 (установить в единицу бит 5 порта B) сначала считывает все реальные значения сигналов, присутствующие в

данный момент на выводах порта. Затем выполняются действия над битом 5, и новое значение байта целиком записывается в выходные фиксаторы. Если другой бит регистра PORTB используется в качестве двунаправленного ввода/вывода (скажем, бит 0), и в данный момент он определен как входной, то входной сигнал на этом выводе будет считан и записан обратно в выходной триггер-фиксатор этого же вывода, стирая предыдущее состояние. До тех пор, пока эта ножка остается в режиме ввода, никаких проблем не возникает. Однако если позднее линия 0 переключится в режим вывода, ее состояние будет неопределенным.

На ножку, работающую в режиме вывода, не должны нагружаться внешние источники токов ("монтажное И", "монтажное ИЛИ"). Большие результирующие токи могут повредить кристалл.

Необходимо выдерживать определенную последовательность обращения к портам ввода/вывода. Запись в порт вывода происходит в конце командного цикла. Но при чтении данные должны быть стабильны в начале командного цикла. Необходимо быть внимательным в операциях чтения, следующих сразу за записью в тот же порт. Здесь надо учитывать инерционность установления напряжения на выводах. Может потребоваться программная задержка, чтобы напряжение на ножке (которое зависит от нагрузки) успело стабилизироваться до начала исполнения следующей команды чтения.

1.2. Модуль таймера и регистр таймера

Структура модуля таймера/счетчика TIMER0 и его взаимосвязь с регистрами TMR0 и OPTION показаны на рис. 3.5.

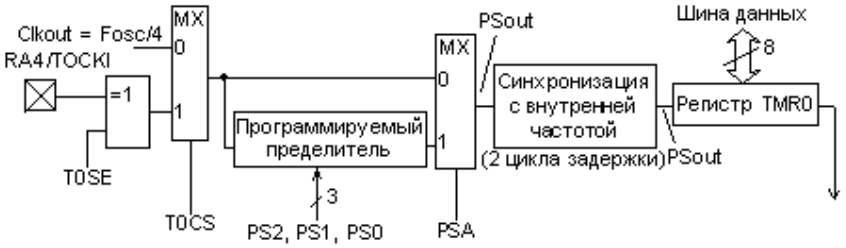

Рис. 3.5. Структурная схема таймера/счетчика TMR0.

TIMER0 является программируемым модулем и содержит следующие компоненты:

- 8-разрядный таймер/счетчик TMR0 с возможностью чтения и записи как регистр;
- 8-разрядный программно управляемый предварительный делитель (предделитель);
- мультиплексор входного сигнала для выбора внутреннего или внешнего тактового сигнала;
- схему выбора фронта внешнего тактового сигнала;
- формирователь запроса прерывания по переполнению регистра TMR0 с FFh до 00h.

Режим таймера выбирается путем сбрасывания в ноль бита T0CS регистра OPTION <5>. В режиме таймера TMR0 инкрементируется каждый командный цикл (без пределителя). После записи информации в TMR0 инкрементирование его начнется после двух командных циклов. Это происходит со всеми командами, которые производят запись или чтение-модификацию-запись TMR0 (например, MOVF TMR0, CLRF TMR0). Избежать этого можно при помощи записи в TMR0 скорректированного значения. Если TMR0 нужно проверить на равенство нулю без останова счета, следует использовать инструкцию MOVF TMR0,W.

Режим счетчика выбирается путем установки в единицу бита T0CS регистра OPTION<5>. В этом режиме регистр TMR0 будет инкрементироваться либо нарастающим, либо спадающим фронтом на выводе RA4/T0CKI от внешних событий. Направление фронта определяется управляющим битом T0SE в регистре OPTION<4>. При T0SE = 0 будет выбран нарастающий фронт.

Предделитель может использоваться или совместно с TMR0, или со сторожевым (Watchdog) таймером. Вариант подключения делителя контролируется битом PSA регистра OPTION<3>. При PSA=0 делитель будет подсоединен к TMR0. Содержимое делителя программе недоступно. Коэффициент деления пределителя программируется битами PS2…PS0 регистра OPTION<2:0>.

Прерывание по TMR0 вырабатывается тогда, когда происходит переполнение регистра таймера/счетчика при переходе от FFh к 00h. Тогда устанавливается бит запроса T0IF в регистре INTCON<2>. Данное прерывание можно замаскировать битом T0IE в регистре INTCON<5>. Бит запроса T0IF должен быть сброшен программно при обработке прерывания. Прерывание по TMR0 не может вывести процессор из режима SLEEP потому, что таймер в этом режиме не функционирует.

При PSA=1 делитель будет подсоединен к сторожевому таймеру как постделитель (делитель на выходе). Возможные варианты использования пределителя показаны на рис. 3.6.

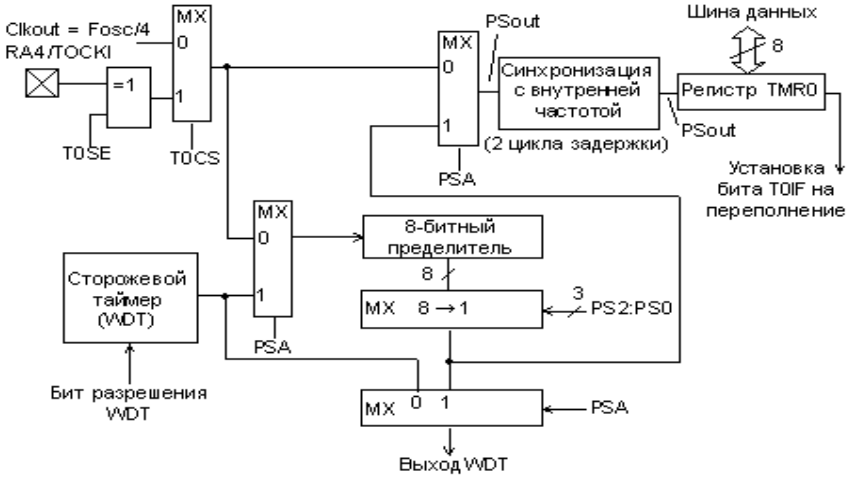

Рис. 3.6. Структура и возможные варианты использования пределителя.

1.3. Память данных в РПЗУ (EEPROM)

Микроконтроллеры подгруппы PIC6F8X имеют энергонезависимую память данных 64x8 EEPROM бит, которая допускает запись и чтение во время нормальной работы (во всем диапазоне питающих напряжений). Эта память не принадлежит области регистровой памяти ОЗУ. Доступ к ней осуществляется посредством косвенной адресации через регистры специальных функций: EEDATA <08h>, который содержит 8-битовые данные для чтения/записи и EEADR <09h>, включающий адрес ячейки, к которой идет обращение. Для управления процессом чтения/записи используются два регистра: EECON1 <88h> и EECON2 <89h>.

При записи байта автоматически стирается предыдущее значение, и записываются новые данные (стирание перед записью). Все эти операции производит встроенный автомат записи EEPROM. Содержимое ячеек этой памяти при выключении питания сохраняется.

Регистр EEADR может адресовать до 256 байт данных EEPROM. В МК подгруппы PIC6F8X используются только первые 64 байта, адресуемые шестью младшими битами EEADR<5:0>. Однако старшие два бита также декодируются. Поэтому эти два бита должны быть установлены в ‘0’, чтобы адрес попал в доступные 64 бита адресного пространства.

Назначение бит регистра EECON1 приведено в 3.1.

Таблица 3.1. Назначение бит регистра EECON1 (адреса 88h).

U	U	U	R/W-0	R/W-x	R/W-0	R/S-0	R/S-x
-	-	-	EEIF	WRERR	WREN	WR	RD
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
Биты 7:5 не используются (читаются как ‘0’)							
Бит 4: EEIF: бит запроса прерывания по записи в EEPROM 0 = операция записи не завершена или не начиналась 1 = операция записи завершена (должен быть сброшен программно)							
Бит 3: WRERR: бит признака ошибки записи в EEPROM 0 = операция записи завершена 1 = операция записи прервана преждевременно (сбросом по /MCLR или сбросом от WDT)							
Бит 2: WREN: бит разрешения записи в EEPROM 0 = запрещена запись в EEPROM 1 = разрешены циклы записи							
Бит 1: WR: бит управления записью 0 = цикл записи данных в EEPROM завершен 1 = инициирует цикл записи (сбрасывается аппаратно по завершении записи. Бит WR может быть только установлен (но не сброшен) программно)							
Бит 0: RD: бит управления чтением 0 = чтение данных EEPROM не инициировано 1 = инициирует чтение данных EEPROM (чтение занимает один цикл. Бит RD сбрасывается аппаратно. Бит RD может быть только установлен (но не сброшен) программно)							

Регистр EECON2 не является физическим регистром. Он используется исключительно при организации записи данных в EEPROM. Чтение регистра EECON2 дает нули.

При считывании данных из памяти EEPROM необходимо записать нужный адрес в EEADR регистр и затем установить бит RD EECON1<0> в единицу. Данные появятся в следующем командном цикле в регистре EEDATA и могут быть прочитаны. Данные в регистре EEDATA фиксируются.

При записи в память EEPROM необходимо сначала записать адрес в EEADR-регистр и данные в EEDATA-регистр. Затем следует выполнить специальную последовательность команд, производящую непосредственную запись:

```
movlw 55h
movwf EECON2
movlw AAh
movwf EECON2
bsf EECON1,WR ;установить WR бит, начать запись
```

Во время выполнения этого участка программы все прерывания должны быть запрещены, для точного выполнения временной диаграммы. Время записи – примерно 10 мс. Фактическое время записи может изменяться в зависимости от напряжения, температуры и индивидуальных свойств кристалла. В конце записи бит WR автоматически обнуляется, а флаг завершения записи EEIF, он же запрос на прерывание, устанавливается.

Для предотвращения случайных записей в память данных предусмотрен специальный бит WREN в регистре EECON1. Рекомендуется держать бит WREN выключенным, кроме тех случаев, когда нужно обновить память данных. Более того, кодовые сегменты, которые устанавливают бит WREN, и те, которые выполняют запись, следует хранить на различных адресах, чтобы избежать случайного выполнения их обоих при сбое программы.

1.4. Организация прерываний

МК подгруппы PIC16F8X имеют четыре источника прерываний:

- внешнее прерывание с вывода RB0/INT;
- прерывание от переполнения счетчика/таймера TMR0;
- прерывание от изменения сигналов на линиях порта RB<7:4>;
- прерывание по окончании записи данных в EEPROM.

Все прерывания имеют один и тот же вектор/адрес – 0004h. Однако в управляющем регистре прерываний INTCON соответствующим битом-признаком записывается, от какого именно источника поступил запрос прерывания. Исключение составляет прерывание по завершении записи в EEPROM, признак которого находится в регистре EECON1.

Бит общего разрешения/запрещения прерывания GIE (INTCON <7>) разрешает (если = 1) все индивидуально незамаскированные прерывания или запрещает их (если = 0). Каждое прерывание в отдельности может быть до-

полнительно разрешено/запрещено установкой/сбросом соответствующего бита в регистре INTCON.

Бит GIE при сбросе обнуляется. Когда начинает обрабатываться прерывание, бит GIE обнуляется, чтобы запретить дальнейшие прерывания, адрес возврата посылается в стек, а в программный счетчик загружается адрес 0004h. Время реакции на прерывание для внешних событий, таких как прерывание от ножки INT или порта B, составляет приблизительно пять циклов. Это на один цикл меньше, чем для внутренних событий, таких как прерывание по переполнению от таймера TMR0. Время реакции всегда одинаковое.

В подпрограмме обработки прерывания источник прерывания может быть определен по соответствующему биту в регистре признаков. Этот флаг-признак должен быть программно сброшен внутри подпрограммы. Признаки запросов прерываний не зависят от соответствующих маскирующих битов и бита общего маскирования GIE.

Команда возврата из прерывания RETFIE завершает прерывающую подпрограмму и устанавливает бит GIE, чтобы опять разрешить прерывания.

Логика прерываний контроллера изображена на рис. 3.7.

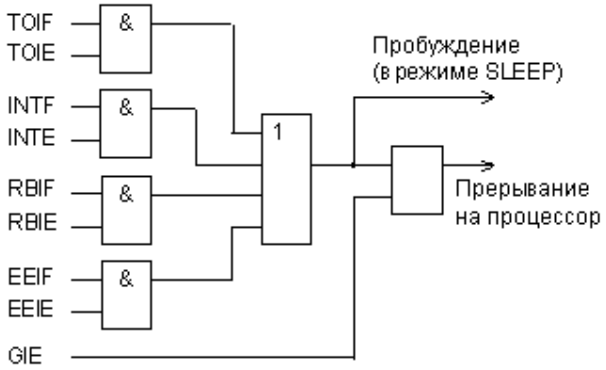

Рис. 3.7. Логика прерываний микроконтроллера.

Внешнее прерывание на ножке RB0/INT осуществляется по фронту: либо по нарастающему (если в регистре OPTION бит INTEDG=1), либо по спадающему (если INTEDG=0). Когда фронт обнаруживается на ножке INT, бит запроса INTF устанавливается в единицу (INTCON <1>). Это прерывание может быть замаскировано сбросом управляющего бита INTE в ноль (INTCON <4>). Бит запроса INTF необходимо очистить прерывающей программой перед тем, как опять разрешить это прерывание. Прерывание INT может вывести процессор из режима SLEEP, если перед входом в этот режим бит INTE был установлен в единицу. Состояние бита GIE также определяет, будет ли процессор переходить на подпрограмму прерывания после выхода из режима SLEEP.

Переполнение счетчика TMR0 (FFh->00h) устанавливает в единицу бит запроса T0IF (INTCON<2>). Это прерывание может быть разрешено/запрещено установкой/сбросом бита маски T0IE (INTCON<5>). Сброс запроса T0IF – дело программы обработки.

Любое изменение сигнала на одном из четырех входов порта RB<7:4> устанавливает в единицу бит RBIF (INTCON<0>). Это прерывание может быть разрешено/запрещено установкой/сбросом бита маски RBIE (INTCON<3>). Сброс запроса RBIF – дело программы обработки.

Признак запроса прерывания по завершении записи в EEPROM, EEIF (EECON1<4>) устанавливается в единицу по окончании автоматической записи данных в EEPROM. Это прерывание может быть замаскировано сбросом бита EEIE (INTCON<6>). Сброс запроса EEIF – дело программы обработки.

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

- специальные функции

Система команд:

- перечень и формат команд
- команды работы с байтами и битами
- команды управления и работы с константами

I. Специальные функции.

Микроконтроллеры подгруппы PIC16F8X имеют набор специальных функций, предназначенных для расширения возможностей системы, минимизации стоимости, исключения навесных компонентов, обеспечения минимального энергопотребления и защиты кода от считывания. В PIC16F8X реализованы следующие специальные функции:

- сброс;
- сторожевой таймер (WDT);
- режим пониженного энергопотребления (SLEEP);
- выбор типа генератора;
- защита кода от считывания;
- биты идентификации;
- последовательное программирование в составе схемы.

В PIC16F8X существуют различия между вариантами сбросов:

- сброс по включению питания;
- сброс по внешнему сигналу /MCLR при нормальной работе;
- сброс по внешнему сигналу /MCLR в режиме SLEEP;
- сброс по окончании задержки таймера WDT при нормальной работе;
- сброс по окончании задержки таймера WDT в режиме SLEEP.

Для реализации сброса по включению питания в МК подгруппы PIC16F8X предусмотрен встроенный детектор включения питания. Таймер установления питания (PWRT) начинает отсчет времени после того, как напряжение питания пересекает уровень около 1,2…1,8 Вольт. По истечении выдержки около 72мс считается, что напряжение достигло номинала и запускается другой таймер – таймер запуска генератора (OST), формирующий выдержку на стабилизацию кварцевого генератора. Программируемый бит конфигурации позволяет разрешать или запрещать выдержку от встроенного таймера установления питания. Выдержка запуска меняется в зависимости от экземпляров кристалла, от питания и температуры. Таймер на стабилизацию генератора отсчитывает 1024 импульса от начавшего работу генератора. Считается, что кварцевый генератор за это время выходит на режим. При использовании RC генераторов выдержка на стабилизацию не производится.

Если сигнал /MCLR удерживается в низком состоянии достаточно долго (дольше времени всех задержек), тогда после перехода /MCLR в высокое состояние программа начнет выполняться немедленно. Это необходимо в тех

случаях, когда требуется синхронно запустить в работу несколько PIC-контроллеров через общий для всех сигнал /MCLR.

Микроконтроллеры подгруппы PIC16F8X имеют встроенный сторожевой таймер WDT. Для большей надежности он работает от собственного внутреннего RC-генератора и продолжает функционировать, даже если основной генератор остановлен, как это бывает при исполнении команды SLEEP. Таймер вырабатывает сигнал сброса. Выработка таких сбросов может быть запрещена путем записи нуля в специальный бит конфигурации WDTE. Эту операцию производят на этапе прожига микросхем.

Номинальная выдержка WDT составляет 18 мс (без использования делителя). Она зависит от температуры, напряжения питания, особенностей типов микросхем. Если требуются большие задержки, то к WDT может быть подключен встроенный пределитель с коэффициентом деления до 1:128, который программируется битами PS2:PS0 в регистре OPTION. В результате могут быть реализованы выдержки до 2,3 секунд.

Команды "CLRWDT" и "SLEEP" обнуляют WDT и пределитель, если он подключен к WDT. Это запускает выдержку времени сначала и предотвращает на некоторое время выработку сигнала сброса. Если сигнал сброса от WDT все же произошел, то одновременно обнуляется бит /TO в регистре статуса. В приложениях с высоким уровнем помех содержимое регистра OPTION подвержено сбою. Поэтому регистр OPTION должен обновляться через равные промежутки времени.

Состояние регистров МК после сброса представлено в табл. 4.1.

Таблица 4.1. Состояние регистров МК после сброса.

Регистр	**Адрес**	**Сброс по включению**	**Другие виды питания сброса**
W	-	xxxx xxxx	uuuu uuuu
INDF	00h	——— ———	——— ———
TMR0	01h	xxxx xxxx	uuuu uuuu
PCL	02h	0000 0000	0000 0000
STATUS	03h	0001 1xxx	000q quuu
FSR	04h	xxxx xxxx	uuuu uuuu
PORT A	05h	—x xxxx	—u uuuu
PORT B	06h	xxxx xxxx	uuuu uuuu
TRIS A	85h	—1 1111	—1 1111
TRIS B	86h	1111 1111	1111 1111
OPTION	81h	1111 1111	1111 1111
EEDATA	08h	xxxx xxxx	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu
EECON1	88h	—0 0000	—0 q000
EECON2	89h	——— ———	——— ———
PCLATH	0Ah	—0 0000	—0 0000
INTCON	0Bh	0000 000x	0000 000u
Здесь: x — неизвестное значение; u — неизменяемый бит; "–" — неиспользуемый бит (читается как "0"); q — значение бита зависит от условий сброса.			

Некоторые из специальных регистров при сбросе не инициализируются. Они имеют случайное состояние при включении питания и не изменяются при иных видах сброса. Другая часть специальных регистров инициализиру-

ется в "состояние сброса" при всех видах сброса, кроме сброса по окончанию задержки таймера WDT в режиме SLEEP. Просто этот сброс рассматривается как временная задержка в нормальной работе. Есть еще несколько исключений. Счетчик команд всегда сбрасывается в ноль (0000h). Биты регистра статуса /TO и /PD устанавливаются или сбрасываются в зависимости от варианта сброса. Эти биты используются программой для определения природы сброса.

Режим пониженного энергопотребления SLEEP предназначен для обеспечения очень малого тока потребления в ожидании (менее 1 мкА при выключенном сторожевом таймере). Выход из режима SLEEP возможен по внешнему сигналу сброса или по окончании выдержки сторожевого таймера. Кристаллы PIC16F8X могут работать с четырьмя типами встроенных генераторов. Пользователь может запрограммировать два конфигурационных бита (FOSC1 и FOSC0) для выбора одного из четырех режимов: RC, LP, XT, HS. Здесь XT – стандартный кварцевый генератор, HS – высокочастотный кварцевый генератор, LP – низкочастотный генератор для экономичных приложений. Микроконтроллеры PIC16F8X могут тактироваться и от внешних источников.

Генератор, построенный на кварцевых или керамических резонаторах, требует периода стабилизации после включения питания. Для этого встроенный таймер запуска генератора держит устройство в состоянии сброса примерно 18 мс после того, как сигнал на /MCLR ножке кристалла достигнет уровня логической единицы.

Возможность выбора типа генератора позволяет эффективно использовать микроконтроллеры семейства в различных приложениях. Применение RC генератора позволяет уменьшить стоимость системы, а низкочастотный LP-генератор сокращает энергопотребление.

Программный код, который записан в кристалл, может быть защищен от считывания при помощи установки бита защиты (CP) в слове конфигурации в ноль. Содержимое программы не может быть прочитано так, чтобы с ним можно было работать. Кроме того, при установленном бите защиты невозможно изменять программу. То же относится и к содержимому памяти данных EEPROM.

Если установлена защита, то бит CP можно стереть только вместе с содержимым кристалла. Сначала будет стерта EEPROM программная память и память данных, и в последнюю очередь – бит защиты кода CP. При считывании защищенного кристалла чтение любого адреса памяти даст результат вида 0000000XXXXXXX(двоичный код), где X – это 0 или 1.

Память данных EEPROM невозможно проверить после установки бита защиты.

Для выбора различных режимов работы используются биты конфигурации. Микроконтроллеры подгруппы PIC16F8X имеют 5 или 6 бит конфигурации, которые хранятся в EEPROM и устанавливаются на этапе программирования кристалла. Эти биты могут быть запрограммированы (читается как "0") или оставлены незапрограммированными (читается "1") для выбора

подходящего варианта конфигурации устройства. Они расположены в EEPROM-памяти по адресу 2007h. Пользователю следует помнить, что этот адрес находится ниже области кодов и недоступен программе.

Назначение бит конфигурации МК PIC16CR83 и PIC16CR84 приведено в табл. 4.2.

Таблица 4.2. Назначение бит конфигурации МК PIC16CR83 и PIC16CR84.

R-u	R/P-u	R-u	R-u	R-u	R-u	R-u
CP	DP	CP	/PWRTE	WDTE	FOSC1	FOSC0
Бит 13:8	Бит 7	Бит 6:4	Бит 3	Бит 2	Бит 1	Бит 0
Биты 13:8 CP: бит защиты памяти программ 0 = память программ защищена 1 = защита отсутствует						
Бит 7 DP: бит защиты памяти данных 0 = память данных защищена 1 = защита отсутствует						
Биты 6:4 CP: бит защиты памяти программ 0 = память программ защищена 1 = защита отсутствует						
Бит 3 /PWRTE: бит использования таймера по включению питания 0 = таймер используется (есть задержка) 1 = таймер не используется						
Бит 2: WDTE: бит использования сторожевого таймера 0 = WDT не используется 1 = WDT используется						
Биты 1:0 FOSC1:FOSC0: бит выбора типа генератора 11 = генератор RC 10 = генератор HS 01 = генератор XT 00 = генератор LP						
Здесь: P — программируемый бит; – n = значение по сбросу после включения питания.						

Назначение бит конфигурации МК PIC16F83 и PIC16F84 приведено в табл. 4.3.

Таблица 4.3. Назначение бит конфигурации МК PIC15F83 и PIC16F84.

R-u	R-u	R-u	R-u	R-u
CP	/PWRTE	WDTE	FOSC1	FOSC0
Бит 13:4	Бит 3	Бит 2	Бит 1	Бит 0
Биты 13:4 CP: бит защиты памяти программ 0 = память программ защищена 1 = защита отсутствует				
Бит 3 /PWRTE: бит использования таймера по включению питания 0 = таймер используется (есть задержка) 1 = таймер не используется				
Бит 2: WDTE: бит использования сторожевого таймера 0 = WDT не используется 1 = WDT используется				
Биты 1:0 FOSC1:FOSC0: бит выбора типа генератора 11 = генератор RC 10 = генератор HS 01 = генератор XT 00 = генератор LP				

Четыре слова памяти, расположенные по адресам 2000h-2003h, предназначены для хранения идентификационного кода (ID) пользователя, контрольной суммы или другой информации. Как и слово конфигурации, они могут быть прочитаны или записаны только с помощью программатора. Доступа из программы к ним нет.

Микроконтроллеры подгруппы PIC16F8X могут быть запрограммированы последовательным способом в составе устройства. Для этого используется всего пять линий: две для данных и тактового сигнала и три для земли, напряжения питания и программирующего напряжения. Разработчик может создать и смакетировать устройство с незапрограммированным прибором, а перед использованием ввести в него программу.

II. Система команд.

2.1. Перечень и формат команд

Микроконтроллеры подгруппы PIC16F8X имеют простую и эффективную систему команд, состоящую всего из 35 команд.

Каждая команда МК подгруппы PIC16F8X представляет собой 14-битовое слово, разделенное на код операции (OPCODE), и поле для одного и более операндов, которые могут участвовать или не участвовать в этой команде. Система команд PIC16F8X является ортогональной и включает в себя команды работы с байтами, команды работы с битами и операции с константами и команды управления. В табл. 4.4 приведены описания полей команд.

Таблица 4.4. Описания полей команд МК семейства PIC16CXXX.

Поле	**Описание**
f	Адрес регистра
w	Рабочий регистр
b	Номер бита в 8-разрядном регистре
k	Константа
x	Не используется. Ассемблер формирует код с x=0
d	Регистр назначения: d=0 – результат в регистре w d=1 – результат в регистре f По умолчанию d=1
label	Имя метки
TOS	Вершина стека
PC	Счетчик команд
PCLATH	Регистр PCLATH
GIE	Бит разрешения всех прерываний
WDT	Сторожевой таймер
/TO	Тайм-аут
/PD	Выключение питания
dest	Регистр назначения: рабочий регистр w или регистр, заданный в команде
[ ]	Необязательные параметры
( )	Содержание
	Присвоение
< >	Поле номера бита
	Из набора

Для команд работы с байтами f обозначает регистр, с которым производится действие; d – бит, определяющий, куда положить результат. Если d =0, то результат будет помещен в регистр w, при d=1 результат будет помещен в регистр "f", упомянутый в команде.

Для команд работы с битами b обозначает номер бита, участвующего в команде, а f – это регистр, в котором данный бит расположен.

Для команд передачи управления и операций с константами, k обозначает восьми- или одиннадцатибитную константу.

Почти все команды выполняются в течение одного командного цикла. В двух случаях исполнение команды занимает два командных цикла:

- проверка условия и переход;
- изменение программного счетчика как результат выполнения команды.

Один командный цикл состоит из четырех периодов генератора. Таким образом, для генератора с частотой 4 МГц время исполнения командного цикла будет 1 мкс.

Основные форматы команд МК изображены на рис. 4.1.

Система команд МК подгруппы PIC16F8X приведена в табл. 4.5.

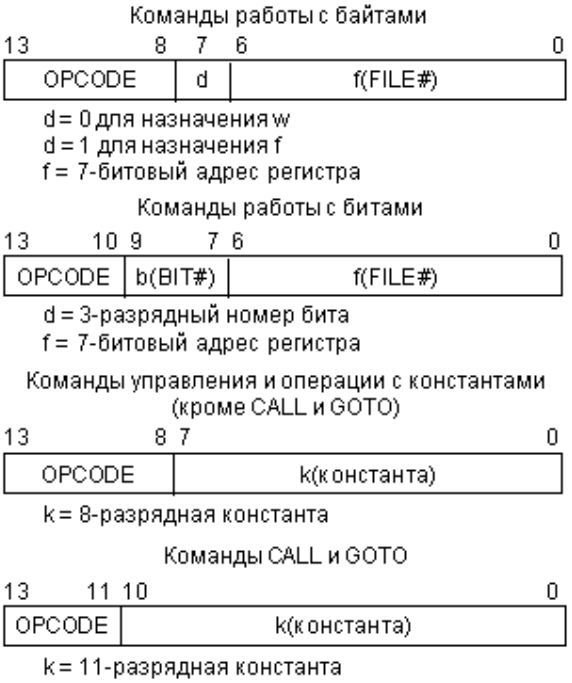

Рис. 4.1. Основные форматы команд.

Таблица 4.5. Система команд МК подгруппы PIC16F8X.

Мнемоника	**Описание команды**	**Циклы**	**Биты состояния**	**Прим.**
ADDWF f, d	Сложение W с f	1	C ,DC ,Z	1, 2
ANDWF f, d	Логическое И W и f	1	Z	1, 2
CLRF f	Сброс регистра f	1	Z	2
CLRW	Сброс регистра W	1	Z	
COMF f, d	Инверсия регистра f	1	Z	1, 2
DECF f, d	Декремент регистра f	1	Z	1, 2
DECFSZ f, d	Декремент f, пропустить коман-	1(2)		1, 2, 3

	ду, если 0			
INCF f, d	Инкремент регистра f	1	Z	1, 2
INCFSZ f, d	Инкремент f, пропустить команду, если 0	1(2)		1, 2, 3
IORWF f, d	Логическое ИЛИ W и f	1	Z	1, 2
MOVF f, d	Пересылка регистра f	1	Z	1, 2
MOVWF f	Пересылка W в f	1		
NOP -	Холостая команда	1		
RLF f, d	Сдвиг f влево через перенос	1	C	1, 2
RRF f, d	Сдвиг f вправо через перенос	1	C	1, 2
SUBWF f, d	Вычитание W из f	1	C,DC,Z	1, 2
SWAPF f, d	Обмен местами тетрад в f	1		1, 2
XORWF f, d	Исключающее ИЛИ W и f	1	Z	1, 2
BCF f, b	Сброс бита в регистре f	1		1, 2
BSF f, b	Установка бита в регистре f	1		1, 2
BTFSC f, b	Пропустить команду, если бит в f равен нулю	1(2)		3
BTFSS f, b	Пропустить команду, если бит в f равен единице	1(2)		3
ADDLW k	Сложение константы и W	1	C, DC, Z	
ANDLW k	Логическое И константы и W	1	Z	
CALL k	Вызов подпрограммы	2		
CLRWDT -	Сброс сторожевого таймера WDT	1	/TO, /P	
GOTO k	Переход по адресу	2		
IORLW k	Логическое ИЛИ константы и W	1	Z	
MOVLW k	Пересылка константы в W	1		
RETFIE -	Возврат из прерывания	2		
RETLW k	Возврат из подпрограммы с загрузкой константы в W	2		
RETURN -	Возврат из подпрограммы	2		
SLEEP -	Переход в режим SLEEP	1	/TO, /P	
SUBLW k	Вычитание W из константы	1	C, DC, Z	
XORLW k	Исключающее ИЛИ константы и W	1	Z	

Примечания к таблице:

1. Если модифицируется регистр ввода/вывода (например, MOVF PORTB,1), то используется значение, считываемое с выводов. Например, если в выходной защелке порта, включенного на ввод, находится "1", а внешнее устройство формирует на этом выводе "0", то в разряде данных будет записан "0".
2. Если операндом команды является содержимое регистра TMRO (и, если допустимо, d=1), то предварительный делитель, если он подключен к TMRO, будет сброшен.
3. Если в результате выполнения команды изменяется счетчик команд или выполняется переход по проверке условия, то команда выполняется за два цикла. Второй цикл выполняется как NOP.

2.2. Команды работы с байтами

Команды работы с байтами используются в PIC МК для пересылки данных между регистрами и выполнения математических операций над их

содержимым. Несмотря на относительно небольшой набор команд, они позволяют реализовать целый ряд операций. Это связано, в частности, с возможностью указать в команде адрес размещения результата операции.

Преимуществом системы команд является также возможность использования различных способов обращения к регистрам. Адрес регистра может быть указан непосредственно в команде соответствующим 7-битовым полем f. При этом доступ возможен только к данным, расположенным в пределах текущего банка данных. Адресация данных может осуществляться и с помощью индексного регистра FSR, путем обращения к регистру косвенной адресации INDF, расположенному по нулевому адресу.

Пересылка данных выполняется с помощью двух команд: MOVF и MOVWF, назначение которых существенно различается. Команда MOVF используется для установки бита нулевого результата в зависимости от содержимого определенного регистра и может применяться для его загрузки в регистр w. Команда MOVWF используется для записи содержимого рабочего регистра w в указанный регистр МК. Если в качестве этого регистра указывается INDF, то адрес регистра назначения выбирается из регистра FSR. При выполнении данной команды биты состояния не изменяются.

Специальные команды CLRF f и CLRW применяются для очистки регистров МК. Команда CLRF f записывает ноль в указанный регистр, а команда CLRW – в рабочий регистр. При этом необходимо помнить, что они также устанавливают соответствующее значение бита нуля.

Наиболее часто используемой арифметической операцией является сложение, которое выполняется командой ADDWF f,d. Эта операция может изменять все биты состояния. Бит нуля устанавливается в 1, если при выполнении логической операции "И" над полученным результатом и числом 0x0FF (255) получается ноль. Бит переноса устанавливается в 1, если результат превышает число 0x0FF. Бит десятичного переноса устанавливается в 1, если сумма четырех младших битов результата превышает 0x0F (15).

При использовании операции вычитания SUBWF f, d следует иметь в виду, что в PIC МК она выполняет операцию сложения с отрицательным числом. То есть вместо операции d = f – w в действительности выполняется d = f + (-w). Отрицательное значение содержимого w вычисляется по формуле Negw = (Posw ^ 0x0FF) + 1.

Команды логических операций ANDWF f, d, IORWF f, d и XORWF f, d позволяют выполнять основные логические операции над соответствующими битами содержимого указанного регистра и регистра w. Бит нуля в регистре STATUS устанавливается в 1 или сбрасывается в 0 в зависимости от значения полученного результата. Команду XORWF f, d удобно использовать для проверки содержимого некоторого регистра. Для этого необходимо загрузить заданное число в регистр w и выполнить операцию XORWF f, d над содержимым проверяемого регистра и w. Если содержимое регистра равно содержимому w, то результат операции будет равен нулю, и бит нуля установится в 1. Команда COMF f, d используется для инвертирования значений всех битов в регистре источника. Следует отметить, что эта команда не делает число от-

рицательным, то есть не переводит его в дополнительный код. Отрицательное число Neg может быть получено из положительного Pos следующим образом: Neg = (Pos ^ 0x0FF) + 1.

Команда SWAPF f, d меняет местами тетрады в регистре. Как и в остальных командах данной группы, результат выполнения может быть записан как в регистре w, так и в регистре-источнике. Данная команда не меняет значения какого либо из битов состояния, что может использоваться для восстановления содержимого контекстных регистров перед возвратом из прерывания. Команду SWAPF f, d можно применять, в частности, для хранения двух цифр в одном регистре, переставляя их в зависимости от того, какую из них вы хотите использовать. С помощью команды SWAPF f, d удобно разделить байт на две тетрады для их последующего отображения на дисплее.

Основной функцией команд циклического сдвига RLF f, d и RRF f, d является сдвиг содержимого регистра влево или вправо на один бит с записью на место младшего значащего бита значения бита переноса или, соответственно, установления бита переноса в соответствии со значением старшего значащего бита. Команды циклического сдвига могут использоваться для умножения и деления на число 2 в степени n. Они также служат для реализации последовательного ввода или вывода данных и позиционирования байта для того, чтобы можно было тестировать значение отдельных битов.

Команды инкремента INCF f, d и декремента DECF f, d используются для изменения содержимого регистра на 1. После выполнения команд инкремента и декремента может измениться только бит нуля. Изменения бита переноса, если результат превысит значение 0x0FF при инкременте или окажется меньше 0 при декременте, не происходит.

Для реализации условных переходов в программе существуют команды инкремента и декремента с пропуском команды при нулевом результате: INCFSZ f, d и DECFSZ f, d. С точки зрения обработки данных они работают аналогично командам INCF f, d и DECF f, d. Основное отличие от этих команд заключается в том, что при нулевом результате выполнения команды INCFSZ f, d или DECFSZ f, d пропускается следующая за ней команда. Это означает, что команды INCFSZ f, d и DECFSZ f, d могут использоваться для организации программных циклов. Другая особенность этих команд состоит в том, что они не влияют на содержимое битов состояния регистра STATUS.

Команда NOP означает отсутствие операции. Традиционно она используется для двух целей. Первая – обеспечение синхронизации программы с временными характеристиками различных устройств системы. Вторым возможным вариантом является использование команды NOP для удаления части программного кода. Вследствие того, что код команды NOP состоит из одних нулей, его легко ввести в память программ вместо любой другой команды, не прибегая к стиранию и репрограммированию всей памяти программ.

2.3. Команды работы с битами

Отличительной особенностью данной группы команд является то, что они оперируют с однобитными операндами, в качестве которых используются отдельные биты регистров МК.

Установка и сброс отдельных битов производится командами BSF f, b и BCF f, b. Любой доступный для записи бит в регистровой памяти может быть модифицирован таким способом. При этом гарантируется, что ни один из остальных битов регистра не будет изменен.

Однако при работе с портами ввода/вывода последнее утверждение не всегда справедливо. Связано это с тем, что значение числа, считываемого из регистра порта, зависит от конфигурации его выводов в качестве входов или выходов данных.

В системе команд, рассматриваемых PIC МК, отсутствуют команды условного перехода. Вместо них имеются такие, которые позволяют пропустить выполнение следующей команды. В частности, рассмотренные выше команды INCFSZ f, d и DECFSZ f, d удобны для организации циклов в программе.

Для управления процессом выполнения программы используются команды работы с битами BTFSC f, b и BTFSS f, b, позволяющие пропустить выполнение следующей команды программы в зависимости от состояния определенного бита в заданном регистре.

Если в качестве заданного регистра используется регистр STATUS, то можно организовать управление переходами программы в зависимости от состояния битов признаков результата операции, как предусмотрено в микропроцессорах стандартной архитектуры.

2.4. Команды управления и работы с константами

Команды работы с константами используют при выполнении операции явно заданные операнды, которые являются частью команды.

Команда MOVLW k используется для записи константы k в рабочий регистр w. Содержимое регистра STATUS при этом не изменяется.

Команда ADDLW k прибавляет непосредственно заданную величину к содержимому регистра w. Эта команда изменяет значения битов нуля, переноса и десятичного переноса таким же образом, как и команда ADDWF f, d.

Команда SUBLW k вычитает содержимое регистра w из заданного значения константы k. В отличие от SUBWF f, d, результат выполнения команды SUBLW k можно представить в следующем виде: w = k + (w ^ 0x0FF) + 1. С помощью этой команды удобно изменять знак содержимого регистра w, используя ее следующим образом: SUBLW 0.

Команды логических операций ANDLW k, IORLW k и XORLW k выполняют побитно соответствующие операции над содержимым регистра w и непосредственно заданной константой k. Эти команды, как и команды работы с байтами, устанавливают только бит нуля в регистре STATUS в соответствии с результатом операции. Полученный результат сохраняется в регистре w.

С помощью команды IORLW 0 удобно определять равенство нулю содержимого регистра w. В зависимости от результата этой операции бит нуля будет установлен в 1 или сброшен в 0.

Команда RETLW k используется для возврата из подпрограммы с установкой начальных условий в регистр w, а также для реализации табличных

преобразований, что будет описано ниже. Перед возвращением из подпрограммы эта команда осуществляет загрузку непосредственно заданной величины в рабочий регистр w.

Команды GOTO k, CALL k, RETURN и RETFIE используются для управления программой.

Команды GOTO k и CALL k могут явно задавать адрес перехода в пределах определенной страницы, размер которой зависит от типа МК: 256/512 адресов для младших моделей, 2К адресов для PIC МК среднего уровня (включая PIC16F8X) и 8К адресов для старших моделей МК. Если адрес перехода выходит за пределы страницы, то регистр PCLATH должен содержать правильную информацию о новой странице.

Команда CALL k выполняется практически так же, как и GOTO k, за исключением того, что указатель на следующую страницу сохраняется в стеке счетчика команд.

Для PIC МК средней группы существует три различных способа возврата из подпрограммы, определяемые командами RETLW k, RETURN и RETFIE. При каждом из этих способов значение адреса извлекается из вершины стека и загружается в счетчик команд. Эти адреса используются для возврата из подпрограмм или прерываний.

Обычное использование команды RETURN приводит к восстановлению адреса команды, следующей за командой вызова подпрограммы. При этом содержимое каких-либо регистров не изменяется, как и значения отдельных битов.

Команда RETFIE используется для возврата из прерывания. Она реализуется аналогично команде RETURN за исключением того, что при ее выполнении устанавливается в 1 бит GIE в регистре управления прерываниями INTCON. Это позволяет после выполнения данной команды немедленно перейти к обработке прерываний, ожидающих своей очереди. В противном случае перед окончанием обработки потребовалась бы проверка наличия запросов других прерываний, и, в случае их поступления, переход к их обработке.

Существует всего две команды, служащие для непосредственного управления функционированием МК. Первая из них – CLRWDT – используется для сброса сторожевого таймера. Вторая – SLEEP – обеспечивает сохранение текущего состояния МК в режиме ожидания, пока не произойдет какое-либо внешнее событие, которое позволит PIC МК продолжить выполнение программы.

Команда CLRWDT сбрасывает в 0 содержимое сторожевого таймера WDT и пределителя (если он используется для установки интервала времени срабатывания WDT), запуская сначала отсчет времени сторожевого таймера. Целью введения команды CLRWDT является предотвращение перезапуска МК при нормальном выполнении программы.

Команда SLEEP служит для двух целей. Первой из них является отключение МК после того, как он закончит выполнение программы. Такое использование МК предполагает, что он необходим только для решения опре-

деленной задачи, например, инициализации других устройств в системе, а затем его функционирование не требуется.

Второй целью использования команды SLEEP является реализация в МК режима ожидания какого-либо события. Существует три события, способные вывести МК из режима ожидания. Первым из них является подача сигнала запуска на вход сброса МК, что приведет к перезапуску процессора и началу выполнения программы с нулевого адреса. Второй способ – поступление сигнала "пробуждения" МК от сторожевого таймера. Третьим способом "пробуждения" является прерывание от какого-либо внешнего источника. При любом способе "пробуждения" использование команды SLEEP позволяет избежать необходимости организации циклов ожидания, а также снизить потребляемую системой мощность.

При этом необходимо иметь в виду, что выход МК из режима ожидания занимает, по меньшей мере, 1024 такта. Поэтому команду SLEEP нельзя использовать в тех случаях, когда требуется быстрая реакция на внешнее событие.

Лекция №5 «Особенности программирования и отладки, разработка программного кода для микроконтроллеров PIC и AVR»

- особенности программирования и отладки

Разработка программного кода:

- ассемблер MPASM

I. Особенности программирования и отладки.

Анализ архитектуры микроконтроллеров PIC с точки зрения их программирования и отладки систем позволяет сделать следующие выводы:

- RISC-система команд обеспечивает высокую скорость выполнения инструкций, но вызывает затруднения и снижение производительности при программировании нетривиальных алгоритмов. Поскольку все инструкции в системе команд являются одноадресными, загрузка константы в любой из регистров требует двух инструкций. Вначале нужно загрузить константу в рабочий регистр w, а затем переслать его содержимое в нужную ячейку памяти данных:
- movlw k
- movwf f

Аналогично, все бинарные арифметико-логические операции приходится выполнять с привлечением рабочего регистра w;

- высокое быстродействие достигается в значительной степени за счет применения конвейера команд. Инструкции ветвления, изменяющие счетчик команд (безусловный переход, вычисляемый переход), не используют инструкцию из очереди, поэтому выполняются за два машинных цикла и снижают темп выполнения программы. Кроме того, сам анализ условий в архитектуре PIC требует выполнения "лишних" команд;
- наличие одного вектора прерываний, отсутствие развитого механизма обработки запросов по приоритетам и вложенных прерываний затрудняют решение сколько-нибудь сложных задач управления. При приходе запроса от любого из источников выполняется переход на процедуру обработки по единственному вектору. В процедуре приходится по битам признаков определять источник, причем условия ветвления, как указывалось выше, анализируются сложно, и все это увеличивает время реакции. После обработки прерывания нужно самостоятельно очистить бит запроса. Из-за отсутствия вложенных прерываний возможно длительное ожидание обработки запросом от источника с более высоким приоритетом;
- аппаратный стек глубиной 8 слов не имеет признака переполнения и ограничивает вложенность процедур. За тем, чтобы он не переполнялся, программист должен следить самостоятельно;
- память данных состоит из банков, для определения текущего банка используются биты регистров STATUS (для PIC16) или BSR (для PIC17).

На этапе трансляции принадлежность указанного регистра текущему активному банку проверить невозможно, для этого требуется моделирование хода выполнения программы;

- память программ разбита на страницы размером 2К слов. Для перехода на нужный адрес по командам CALL и GOTO должны быть правильно установлены биты выбора текущей страницы в регистре PCLATH. На этапе трансляции невозможно проверить корректность передачи управления во время выполнения, для этого также требуется моделирование выполнения программы;
- ограниченность ресурсов МК серии PIC делает проблематичным их программирование на языках высокого уровня.

Указанные особенности архитектуры микроконтроллеров PIC компенсируются чрезвычайно низкой ценой, поэтому такие изделия (особенно семейства PIC16) весьма популярны. В настоящее время их используют даже вместо логических ИС средней степени интеграции. Но реализовать все преимущества этих МК можно только при наличии средств программирования и отладки, адекватных по цене и функциональным возможностям решаемым задачам. Важнейшие требования к инструментальным средствам для МК, ориентированным на выполнение функций ввода-вывода, можно сформулировать следующим образом:

- основным назначением этих средств является поддержка программирования на языке ассемблер и перенос программы на плату системы управления;
- мощные драйверы портов ввода/вывода, состояние которых однозначно описывается значениями в регистрах управления, упрощают функцию замещения электрофизических параметров прототипной БИС, поэтому такие порты можно имитировать с помощью БИС программируемой логики;
- стоимость инструментальных средств должна соответствовать невысокой стоимости одноплатного контроллера.

II. Разработка программного кода.

Разработка программного обеспечения является центральным моментом общего процесса проектирования. Центр тяжести функциональных свойств современных цифровых систем находится именно в программных средствах.

Основным инструментом для профессиональной разработки программ является ассемблер, предполагающий детализацию на уровне команд МК. Только ассемблер позволяет максимально использовать ресурсы кристалла.
Для микроконтроллеров PIC выпущено большое количество различных средств разработки. В данной главе речь пойдет о средствах, предоставляемых фирмой Microchip, которые весьма эффективны и широко используются на практике.

2.1. Ассемблер MPASM

Ассемблер MPASM представляет собой интегрированную программную среду для разработки программных кодов PIC микроконтроллеров всех семейств. Выпускается фирмой Microchip в двух вариантах: для работы под DOS и для работы под Windows 95/98/NT. Ассемблер MPASM может использоваться как самостоятельно, так и в составе интегрированной среды разработки MPLAB. Он включает несколько программ: собственно MPASM, MPLINK и MPLIB, причем каждая из них обладает собственным интерфейсом.

Программа MPASM может использоваться для двух целей:

- генерации исполняемого (абсолютного) кода, предназначенного для записи в МК с помощью программатора;
- генерации перемещаемого объектного кода, который затем будет связан с другими ассемблированными или компилированными модулями.

Исполняемый код является для MPASM выходным кодом по умолчанию. При этом все переменные источника должны быть явно описаны в тексте программы или в файле, подключаемом с помощью директивы INCLUDE <filename>. Если при ассемблировании не выявляется ошибок, то генерируется выходной .hex-файл, который может быть загружен в МК с помощью программатора.

При использовании ассемблера MPASM в режиме генерации перемещаемого объектного кода формируются объектные модули, которые могут быть впоследствии объединены с другими модулями при помощи компоновщика MPLINK. Программа-компоновщик MPLINK преобразует перемещаемые объектные коды в исполняемый бинарный код, привязанный к абсолютным адресам МК. Библиотечная утилита MPLIB позволяет для удобства работы сгруппировать перемещаемые объекты в один файл или библиотеку. Эти библиотеки могут быть связаны компоновщиком MPLINK в файл выходного объектного кода ассемблера MPASM.

Программы MPASM и MPLINK доступны через оболочку MPASM, тогда как MPLIB доступна только со своей командной строки.

Исходным файлом для ассемблера MPASM по умолчанию является файл с расширением .ASM. Текст исходного файла должен соответствовать требованиям синтаксиса, приведенным далее.

Ассемблер MPASM может быть вызван командной строкой

MPASM [/<Option>[/<Option>...]] <file_name>

где /<Option> означает выбор режима работы ассемблера в командной строке; <file_name> – имя файла на ассемблирование.

Режимы работы ассемблера, выбранные по умолчанию, приведены в 5.1.

Здесь и далее используются следующие соглашения по использованию символов:

- [ ] – для аргументов по выбору;
- < > – для выделения специальных ключей <TAB>, <ESC> или дополнительного выбора;

- | – для взаимоисключающих аргументов (выбор ИЛИ);
- строчные символы – для обозначения типа данных.

Таблица 5.1. Режимы работы ассемблера по умолчанию.

Выбор	**Значение по умолчанию**	**Описание**
?	N/A	Вызвать помощь
a	INHX8M	Генерировать абсолютный .COD и hex выход непосредственно из ассемблера:
c	On	Выбрать/запретить случай чувствительности
e	On	Выбрать/запретить файл ошибок
h	N/A	Отобразить панель помощи MPASM
l	On	Выбрать/запретить файл листинга, генерированный из макроассемблера.
m	On	Вызвать/запретить макрорасширение
o	N/A	Установить путь для объектных файлов /o<path>\object.file
p	None	Установить тип процессора: /p<processor_type>;
q	Off	Разрешить/Запретить скрытый режим (запретить вывод на экран)
r	Hex	Определяет тип числа по умолчанию: /r<radix>
w	0	Определяет уровень диагностических сообщений в файле листинга /w<level>, где <level> может быть: 0 – сообщать все, 1 – сообщать о предупреждениях и ошибках, 2 – сообщать только об ошибках.
x	Off	Разрешить/запретить перекрестные ссылки в файле листинга.

Выбор по умолчанию, приведенный в табл. 5.1, может быть изменен командной строкой:

- /<option> разрешает выбор;
- /<option>+ разрешает выбор;
- /<option> – запрещает выбор.

Исходный ассемблерный файл создается с использованием любого ASCII текстового редактора. Каждая линия исходного файла может содержать до четырех типов информации:

- метки (labels)
- мнемоника (mnemonics)
- операнды (operands)
- комментарий (comments)

Порядок и положение каждого типа имеет значение. Метка должна начинаться в колонке номер один. Мнемоника может начинаться в колонке два или далее. Операнды идут за мнемоникой. Комментарий может следовать за операндом, мнемоникой или меткой или может начинаться в любом столбце, если в качестве первого не пустого символа используется * или ;.
Максимальная длина строки 255 символов.

Один или несколько пробелов должны отделять метку и мнемонику или мнемонику и операнд(ы). Операнды могут отделяться запятой.

Например:

```
List  p=16C54, r=HEX
      ORG 0x1FF      ;Вектор сброса
      GOTO START     ;Возврат на начало
      ORG 0x000      ;Адрес начала исполнения
                     ;программы
START
      MOVLW 0x0A     ;Выполнение программы
                     ;PIC МК
      MOVLW 0x0B     ;Выполнять всегда
      GOTO START
      END
```

Метки

В поле метки размещается символическое имя ячейки памяти, в которой хранится отмеченный операнд. Все метки должны начинаться в колонке 1. За ними может следовать двоеточие (:), пробел, табуляция или конец строки. Комментарий может также начинаться в колонке 1, если используется одно из обозначений комментария.

Метка может начинаться с символа или нижнего тире (_) и содержать буквенные символы, числа, нижние тире и знак вопроса. Длина метки может быть до 32 символов.

Мнемоники

Мнемоники представляют собой мнемонические обозначения команды, которые непосредственно транслируются в машинный код. Мнемоники ассемблерных инструкций, директивы ассемблера и макровызовы должны начинаться, по крайней мере, в колонке 2. Если есть метка на той же линии, она должна быть отделена от этой метки двоеточием или одним или более пробелами или табуляцией.

Операнды

В этом поле определяются операнды (или операнд), участвующие в операции. Операнды должны быть отделены от мнемоники одним или более пробелами или табуляцией. Операнды отделяются друг от друга запятыми. Если операция требует фиксированного номера (числа) или операндов, то все на линии после операндов игнорируется. Комментарии разрешаются в конце линии. Если мнемоники позволяют использовать различное число операндов, конец списка операндов определяется концом строки или комментарием.

Выражения используются в поле операнда и могут содержать константы, символы или любые комбинации констант и символов, разделенных арифметическими операторами. Перед каждой константой или символом может стоять + или –, что указывает на положительное или отрицательное выражение.

В ассемблере MPASM используются следующие форматы выражений:

- текстовая строка;
- числовые константы и Radix;

- арифметические операторы и приоритеты;
- High / Low операторы.

Текстовая строка – это последовательность любых допустимых ASCII символов (в десятичном диапазоне от 0 до 127), заключенная в двойные кавычки. Строка может иметь любую длину в пределах 132 колонок. При отсутствии ограничения строки она считается до конца линии. Если строка используется как буквенный операнд, она должна иметь длину в один символ, иначе будет ошибка.

Числовая константа представляет собой число, выраженное в некоторой системе счисления. Перед константой может стоять + или –. Промежуточные величины в константах рассматриваются как 32-разрядные целые без знака.

MPASM поддерживает следующие системы счисления (представления значений или Radix): шестнадцатеричную, десятичную, восьмеричную, двоичную и символьную. По умолчанию принимается шестнадцатеричная система. Табл. 5.2. представляет различные системы счисления.

Таблица 5.2. Системы счисления (Radix).

Тип	**Синтаксис**	**Пример**
Десятичная	D'<цифры>' или .<цифры>	D'100' или .100
16-ричная	H'<цифры>' или 0x<цифры>	H'9f' или 0x9f
Восьмеричная	O'<цифры>'	O'777'
Двоичная	B'<цифры>'	B'00111001'
Символьная	'<символ>' или A'<символ>'	"C" или A'C'

Операторы – это арифметические символы, подобные + и –, которые используются при формировании выражений. Каждый оператор имеет свой приоритет. В общем случае приоритет устанавливается слева направо, а выражения в скобках оцениваются первыми. В табл. 5.3. приведены обозначения, описания и примеры применения основных операторов MPASM.

Операторы high, low и upper используются для получения одного байта из многобайтного значения, соответствующего метке. Применяются для управления расчетом точек динамического перехода при чтении таблиц и записи программ.

Операторы инкремента и декремента могут применяться к переменной только в качестве единственного оператора в строке. Они не могут быть встроенным фрагментом более сложного выражения.

Таблица 5.3. Основные арифметические операторы MPASM

Оператор	Описание	Пример
$	Текущий счетчик команд	goto $ + 3
(	левая скобка	1 + (d * 4)
)	правая скобка	(lenght + 1) * 255
!	операция «НЕ» (логическая инверсия)	if ! (a - b)
~	дополнение	flags = ~ flags
-	инверсия (двоичное дополнение)	– 1 * lenght
High	выделить старший байт слова	movlw high llasid
Low	выделить младший байт слова	movlw low (llasid + .251)
upper	выделить наибольший байт слова	movlw upper (llasid + .251)
*	Умножение	a = c * b
/	Деление	a = b / c
%	Модуль	lenght = totall % 16
+	Сложение	Tot_len = lenght * 8 + 1
-	Вычитание	Entry_Son = (Tot – 1) / 8
<<	сдвиг влево	Val = flags << 1
>>	сдвиг вправо	Val = flags >> 1
>=	больше либо равно	if ent >= num
>	больше	if ent > num
<	меньше	if ent < num
<=	меньше либо равно	if ent <= num
==	равно	if ent == num
!=	не равно	if ent != num
&	поразрядное «И»	flags = flags & err_bit
^	поразрядное «ИСКЛЮЧАЮЩЕЕ ИЛИ»	flags = flags ^ err_bit
|	поразрядное «ИЛИ»	flags = flags | err_bit
&&	логическое «И»	if (len == 512)&&(b == c)
||	логическое «ИЛИ»	if (len == 512) || (b == c)
=	установить равным...	entry_index = 0
++	увеличить на 1 (инкремент)	i ++
—	уменьшить на 1 (декремент)	i —

Комментарии

Поле комментария может использоваться программистом для текстового или символьного пояснения логической организации программы. Поле комментария полностью игнорируется ассемблером, поэтому в нем можно применять любые символы. Комментарии, которые используются в строке сами по себе, должны начинаться с символа комментария (* или ;). Комментарии в конце строки должны быть отделены от остатка строки одним или более пробелами или табуляцией.

Расширения файлов, используемые MPASM и утилитами

Существует ряд расширений файлов, применяемых по умолчанию MPASM и связанными утилитами. Назначения таких расширений приведены в табл. 5.4.

Таблица 5.4. Используемые по умолчанию назначения расширений файлов.

Расширение	**Назначение**
.ASM	Входной файл ассемблера для MPASM <source_name>.ASM
.OBJ	Выходной файл перемещаемого объектного кода из MPASM <source_name>.OBJ
.LST	Выходной файл листинга, генерируемый ассемблером MPASM или MPLINK: <source_name>.LST
.ERR	Выходной файл ошибок из MPASM: <source_name>.ERR
.MAP	Выходной файл распределения памяти из MPASM: <source_name>.MAP
.HEX	Выходной файл объектного кода в шестнадцатеричном представлении из MPASM: <source_name>.HEX
.HXL/.HXH	Выходной файл объектного кода в шестнадцатеричном представлении с раздельным представлением младших и старших байт: <source_name>.HXL, <source_name>.HXH
.LIB	Библиотечный файл, созданный MPLIB и привязанный компоновщиком MPLINK: <source_name>.LIB
.LNK	Выходной файл компоновщика: <source_name>.LNK
.COD	Выходной символьный файл или файл отладчика. Формируются MPASM или MPLINK: <source_name>.COD

Листинг представляет собой текстовый файл в формате ASCII, который содержит машинные коды, сгенерированные в соответствии с каждой ассемблерной командой, директивой ассемблера или макрокомандой исходного файла. Файл листинга содержит: имя продукта и версии, дату и время, номер страницы вверху каждой страницы.

В состав листинга входят также таблица символов и карта использования памяти. В таблице символов перечисляются все символы, которые есть в программе, и где они определены. Карта использования памяти дает графическое представление о расходовании памяти МК.

Директивы языка

Директивы языка – это ассемблерные команды, которые встречаются в исходном коде, но не транслируются прямо в исполняемые коды. Они используются ассемблером при трактовке мнемоники входного файла, размещении данных и формировании файла листинга.

Существует четыре основных типа директив в MPASM:

- директивы данных;
- директивы листинга;
- управляющие директивы;
- макро-директивы.

Директивы данных управляют распределением памяти и обеспечивают доступ к символическим обозначениям данных.

Директивы листинга управляют листингом файла MPASM и форматом. Они определяют спецификацию заголовков, генерацию страниц и другие функции управления листингом.

Директивы управления позволяют произвести секционирование обычного ассемблерного кода.

Макро-директивы управляют исполнением и распределением данных в пределах определений макротела.

Ниже приводится описание некоторых директив ассемблера MPASM.

CODE – начало секции объектного кода

Синтаксис:

```
[<label>] code [ROM address>]
```

Используется при генерации объектных модулей. Объявляет начало секции программного кода. Если <label> не указана, секция будет названа code. Стартовый адрес устанавливается равным указанному значению или нулю, если адрес не был указан.

Пример:

```
RESET code H'01FF'
     goto START
```

#DEFINE – определить метку замены текста

Синтаксис:

```
#define <name> [<string>]
```

Директива задает строку <string>, замещающую метку <name> всякий раз, когда та будет встречаться в исходном тексте.

Символы, которые определены директивой #DEFINE, не могут быть просмотрены симулятором. Используйте вместо этой директивы EQU.

Пример:

```
#define length 20
#define control 0x19,7
#define position (X,Y,Z) (y-(2 * Z +X)).
test_label dw position(1, length, 512)
bsf control     ; установить в 1 бит 7 в f19
```

END – конец программного блока

Синтаксис:

```
end
```

Определяет конец программы. После остановки программы таблица символов сбрасывается в файл листинга.

Пример:

```
start
;исполняемый код
;
end     ; конец программы
```

EQU – определить ассемблерную константу

Синтаксис:

```
<label> equ <expr>
```

Здесь <expr> – это правильное MPASM выражение. Значение выражения присваивается метке <label>.

Пример:

```
four equ 4   ; присваивает численное значение
             ; метке four
```

INCLUDE – включить дополнительный файл источника

Синтаксис:

```
include <<include_file>>
include "<include_file>"
```

Определяемый файл считывается как источник кода. По окончании включаемого файла будет продолжаться ассемблирование исходника. Допускается до шести уровней вложенности. <include_file> может быть заключен в кавычки или угловые скобки. Если указан полный путь к файлу, то поиск будет происходить только по этому пути. В противном случае порядок поиска следующий: текущий рабочий каталог, каталог, в котором находится исходник, каталог MPASM.

Пример:

```
include "c:\sys\sysdefs.inc" ; system defs
include <addmain.asm> ; register defs
```

LIST – установить параметры листинга

Синтаксис:

```
list [<list_option>, , <list_option>]
```

Директива <list> разрешает вывод листинга, если он до этого был запрещен. Кроме того, один из параметров листинга может быть изменен для управления процессом ассемблирования в соответствии с табл. 5.5..

Таблица 5.5. Параметры, используемые директивой list.

Параметр	**Значение по умолчанию**	**Описание**
C=nnn	80	Количество символов в строке
n=nnn	59	Количество строк на странице
t=ON|OFF	OFF	Укорачивать строки листинга
p=<type>	None	Установить тип процессора: PIC16C54, PIC16C84, PIC16F84, PIC17C42 и др.
r=<radix>	HEX	Установить систему счисления по умолчанию: hex, dec, oct.
w=<level>	0	Установить уровень сообщений диагностики в файле листинга: 0 – выводить все сообщения; 1 – выводить предупреждения и ошибки; 2 – выводить только ошибки.
x=ON|OFF	OFF	Включить или выключить макрорасширения.

NOLIST – выключить выход листинга

Синтаксис:

```
NOLIST
```

ORG – установить начальный адрес программы

Синтаксис:

```
<label> org <expr>
```

Устанавливает начальный адрес программы для последующего кода в соответствии с адресом в <expr>. MPASM выводит перемещаемый объектный код, а MPLINK разместит код по определенному адресу. Если метка <label> определена, то ей будет присвоена величина <expr>. По умолчанию начальный адрес имеет нулевое значение. Директива может не использоваться, если создается объектный модуль.

Пример:

```
int_1 org 0x20; Переход по вектору 20
int_2 org int_1+0x10; Переход по вектору 30
```

PROCESSOR – установить тип процессора

Синтаксис:

```
processor <processor_type>
```

Устанавливает тип используемого процессора <processor_type>: [16C54 | 16C55 | 16C56 | 16C57 | 16C71 | 16C84 | 16F84 | 17C42]. Общие процессорные семейства могут быть выбраны как:[16C5X | 16CXX | 17CXX]

Для поддержания совместимости с новыми изделиями выбирается максимум доступной памяти.

SET – определить ассемблерную переменную

Синтаксис:

```
<label> set <expr>
```

Директива SET функционально эквивалентна директиве EQU, за исключением того, что величина, определяемая SET, может быть изменена директивой SET.

Пример:

```
area set 0
width set 0x12
length set 0x14
area set length * width
length set length + 1
```

TITLE – Определить программный заголовок

Синтаксис:

```
title "<title_text>"
```

Эта директива устанавливает текст, который используется в верхней линии страницы листинга. <title_text> - это печатная ASCII последовательность, заключенная в двойные скобки. Она может быть до 60 символов длиной.

Пример:

```
title "operational code, rev 5.0"
```

Лекция №6 «Разработка программного обеспечения для микроконтроллеров PIC и AVR»

- компоновщик MPLINK
- менеджер библиотек MPLIB
- симулятор MPSIM

1. Компоновщик MPLINK

Абсолютный (неперемещаемый) код программы генерируется непосредственно при ассемблировании и располагается в программной памяти в порядке следования операторов программы. Операторы перехода на метку сразу же заменяются соответствующим кодом перехода на адрес метки.

При генерации перемещаемого кода каждая секция программного кода должна предваряться директивой CODE. Окончательное размещение программных кодов, расстановку физических адресов переходов выполняет компоновщик MPLINK.

Компоновщик MPLINK выполняет следующие задачи:

- распределяет коды и данные, т.е. определяет, в какой части программной памяти будут размещены коды и в какую область ОЗУ будут помещены переменные;
- распределяет адреса, т.е. присваивает ссылкам на внешние объекты в объектном файле конкретные физические адреса;
- генерирует исполняемый код, т.е. выдает файл в формате .hex, который может быть записан в память МК;
- отслеживает конфликты адресов, т.е. гарантирует, что программа или данные не будут размещаться в пространстве адресов, которое уже занято;
- предоставляет символьную информацию для отладки.

Для более подробного изучения работы компоновщика следует обратиться к специальной литературе.

2. Менеджер библиотек MPLIB

Менеджер библиотек позволяет создавать и модифицировать файлы библиотек. Библиотечный файл является коллекцией объектных модулей, которые размещены в одном файле. MPLIB использует объектные модули с именем типа «filename.o» формата COFF (Common Object File Format).

Использование библиотечных файлов упрощает компоновку программы, делает ее более структурированной и облегчает ее модификацию.

3. Симулятор MPSIM

Симулятор MPSIM представляет собой симулятор событий, предназначенный для отладки программного обеспечения PIC-контроллеров. MPSIM моделирует все функции контроллера, включая все режимы сброса, функции таймера/счетчика, работу сторожевого таймера, режимы SLEEP и Power-down, работу портов ввода/вывода.

MPSIM запускается из командной строки DOS, конфигурируется пользователем и непосредственно применяет выходные данные ассемблера MPASM.

Перед использованием симулятора необходимо отассемблировать исходный файл <file_name>.asm и получить файл объектного кода в формате INHX8M, создаваемый MPASM по умолчанию:
MPASM <file_name>.asm <RETURN>
Чтобы запустить симулятор, необходимо набрать в командной строке
MPSIM<RETURN>.

Вид экрана, получаемого при запуске MPSIM, показан на рис.6.1. Экран разделен на три части, или окна. В верхнем окне показано текущее состояние моделирования, включая моделируемую программу, тип МК, число выполненных командных циклов и затраченное на них время. Среднее окно используется для вывода содержимого регистров пользователя. Набор регистров и формат выводимых на экран данных определяются файлом MPSIM.INI, который далее будет описан подробнее. Нижнее окно содержит приглашение на ввод команд, а также текущие операции и результат их выполнения.

При запуске симулятор MPSIM начинает искать командный файл MPSIM.INI. Этот текстовой файл создается пользователем и используется для задания всех задействованных в программе параметров.

```
User4 RADIX=X MPSIM 5.20 16c84 TIME=0.0u 0 ?=Help
W: 00 F1: 00 F2: 1FF F3: 0001111 IOA: 0F F5: 0F

%P84                ;Choose Microcontroller number = 84
%SR X               ;Set Input/Output radix to
heXadecimal
%ZR                 ;Set all registers to 0
%ZT                 ;Zero elapsed time counter to 0
%RE                 ;Reset elapsed time and step
count
%V W,X,2            ;register W
%AD  F1,X,2         ;register TMR0
%AD  F2,X,3         ;register PCL
%AD  F3,B,8         ;register STATUS
%AD  IOA,X,2        ;Port "A" TRIS register
%AD  F5,X,2         ;Port "A" register
%RS                 ;Reset
%SC 1               ;Set the clock 1MHz
%LO user4
Hex code loaded
Listing file        loaded
Symbol table        loaded
218960 bytes        memory free
%_
```

Рис. 6.1. Вид рабочего окна симулятора MPSIM.

Один из примеров файла MPSIM.INI приведен ниже.

Листинг 1. Пример файла MPSIM.INI

```
;*******************************************************
;*листинг исходной программы
;*******************************************************
;
LIST P=16C84, R=HEX     ;директива, определяющая тип
                        ;процессора и систему счисления
                        ;по умолчанию
;*******************************************************
;*описание используемых переменных и назначения адресов
;*ячеек для хранения переменных пользователя
;*******************************************************
;
;  INTCON    EQU         0x0B
;  OPTION    EQU         0x81
;  TMR0      EQU         0x01
;  INTF      EQU         1
;  T0IF      EQU         5
   PCL       EQU         0x02
   STATUS    EQU         0x03
   RP0       EQU         5
   PORTA     EQU         0x05
   PORTB     EQU         0x06
   TRISA     EQU         0x05
   TRISB     EQU         0x06
   W         EQU         0
   F         EQU         1
   TEMPA     EQU         0x0C
   TEMPB     EQU         0x0D
   COUNT1    EQU         0x0E
   COUNT2    EQU         0x0F
   COUNT3    EQU         0x10
;*******************************************************
;*определение меток замены текста
;*******************************************************
;
   #DEFINE   Z           STATUS,2  ;бит нулевого результата
   #DEFINE   BA1         PORTA,0   ;динамик BA1
   #DEFINE   VD2         PORTA,1   ;светодиод VD2
   #DEFINE   SA1         PORTA,2   ;тумблер SA1
   #DEFINE   SA2         PORTA,3   ;тумблер SA2
   #DEFINE   SB1         PORTA,4   ;кнопка SB1
   #DEFINE   SB2         PORTB,0   ;кнопка SB2
   #DEFINE   HL1_A       PORTB,1   ;индикатор-сегмент A
   #DEFINE   HL1_B       PORTB,2   ;индикатор-сегмент B
   #DEFINE   HL1_C       PORTB,3   ;индикатор-сегмент C
   #DEFINE   HL1_D       PORTB,4   ;индикатор-сегмент D
   #DEFINE   HL1_E       PORTB,5   ;индикатор-сегмент D
   #DEFINE   HL1_F       PORTB,6   ;индикатор-сегмент E
   #DEFINE   HL1_G       PORTB,7   ;индикатор-сегмент F
;*******************************************************
;
;*исполняемая программа
;*******************************************************
;
   ORG    0x000          ;установка начального адреса по
                         ;сбросу
   GOTO   BEGIN          ;переход на начало программы
```

```
   ORG    0x005        ;установка начального адреса
                       ;размещения программы
BEGIN
   CALL   INIT_PORTS   ;вызов подпрограммы
                       ;инициализации портов МК
;*****************************************************
;*программа пользователя
;
;*****************************************************
;
INIT_PORTS             ;подпрограмма инициализации
                       ;портов
   MOVLW    0xFF       ;установка линий портов
   MOVWF    PORTA      ;A и B в единичное
   MOVWF    PORTB      ;состояние
   BSF      STATUS,RP0 ;переход на банк 1
   MOVLW    0x1C       ;настройка линий RA0 и
   MOVWF    TRISA      ;RA1 порта A на вывод –
                       ;остальных – на ввод
   MOVLW    0x01       ;настройка линии RB0
   MOVWF    TRISB      ;порта B на ввод -
                       ;остальных – на вывод
   BCF      STATUS,RP0 ;возврат в банк 0
   RETURN              ;возврат из подпрограммы
;
   END                 ;конец программы
```

В представленном файле указаны: тип микроконтроллера, система счисления данных по умолчанию, регистры, содержимое которых выводится на экран, способ представления данных, рабочие параметры. Любая команда, которая исполняется MPSIM, может быть задана в файле MPSIM.INI, который определяет начальное состояние программы. При работе MPSIM создает файл MPSIM.JRN, в котором сохраняются все сведения о нажатии клавиш в процессе работы.

В файле MPSIM.INI допускается вводить комментарии, которые даются после знака «;», но не допускается использование пустых строк.

Основные команды, применяемые в симуляторе MPSIM, приведены в табл 6.1. Когда эти команды вводятся в сеансе работы с MPSIM, они заносятся в файл MPSIM.JRN, который используется при создании расширенного файла MPSIM.INI. Данный файл можно задействовать для выявления ошибок и обеспечения нормального выполнения программы после исправления кода.

Для моделирования внешних тестовых событий (воздействий) на моделируемый МК используются файлы стимуляции с расширением .STI. Эти файлы используются MPSIM для того, чтобы обеспечить подачу однократных и повторяющихся входных сигналов в процессе выполнения программы. При этом можно наблюдать на экране, как МК реагирует на сигналы.

Таблица 6.1. Основные команды симулятора MPSIM.

Команда	Параметр	Комментарии
AB	-	Прерывание текущей сессии
AD	Reg[, Radix[, Digits]]	Вывод содержимого регистра на экран в указанном формате и заданной системе счисления X, B или D
B	[addr]	Установка точки останова по текущему или указанному адресу
C	[#break]	Продолжение выполнения программы с пропуском указанного количества следующих точек останова
DB	-	Вывод на экран всех активных точек останова
DI	[addr1[,addr2]]	Вывод на экран фрагмента памяти программ
DR	-	Вывод содержимого всех регистров
DW	[E|D]	Разрешение/запрещение функционирования сторожевого таймера
E	[addr]	Выполнение программы с текущего или указанного адреса
F	Reg	Вывод на экран содержимого регистра и возможность его редактирования пользователем
GE	filename	Получение и выполнение командного файла. Это способ загрузки командного файла .INI
GO	-	Запуск МК и начало выполнения программы
IP	[time|step]	Ввод входных воздействий в соответствии со значением параметра step в файле Stimulus
LO	filename	Загрузка в MPSIM файлов .HEX и .COD
M	addr	Вывод на экран содержимого памяти программ, начиная с адреса «addr» и возможность его редактирования. Ввод «Q» завершает команду.
P	device	Выбор типа моделируемого МК
Q	-	Выход из MPSIM и запись команд в файл .JRN
RE	-	Сброс времени выполнения и счетчика циклов
RS	-	Сброс моделируемого МК
SE	pin|port	Вывод на экран состояния указанного вывода или порта и возможность его изменения
SR	O|X|D	Установка системы счисления по умолчанию
SS	[addr]	Пошаговое исполнение, начиная с указанного адреса. При отсутствии адреса – исполнение идет с текущего места
ST	filename	Загрузка файла стимуляции
W		Отображение состояния регистра W с возможностью его модификации
ZM	addr1,addr2	Очистка памяти программ с адреса addr1 по addr2
ZR	-	Сброс всех регистров МК
ZT	-	Сброс таймера/счетчика МК

В качестве примера ниже приведен файл для тестирования программы, выполняющей опрос состояния линии 1 порта A.

```
! test1.STI
STEP  RA1
   1   1 !Установка на входе RA1 состояния "1"
 200   0 !Поступление на вход RA1 сигнала "0"
1000   1 !Переход сигнала на входе RA1 в "1"
1200   0 !Повторная подача нулевого сигнала
```

Файл воздействия состоит из множества состояний, для которых задается параметр STEP, определяющий число циклов, в течение которых поддерживается указанное состояние. Он позволяет одновременно подавать сигналы на различные выводы МК. В файле воздействия можно указать любой вывод МК, в том числе и вывод сброса (_MCLR). Для обозначения комментариев используется знак !

Лекция №7 «Макет микропроцессорной системы и программирование простейших задач для микроконтроллеров PIC и AVR»

- Описание макета микроконтроллера
- Примеры завершенных программ

I. Описание макета контроллера.

1. Описание макета

Для того чтобы написать первые программы и проверить их функционирование, желательно описать относительно несложный макет, содержащий самые распространенные периферийные устройства. Схема подобного макета приведена на рис. 7.1.

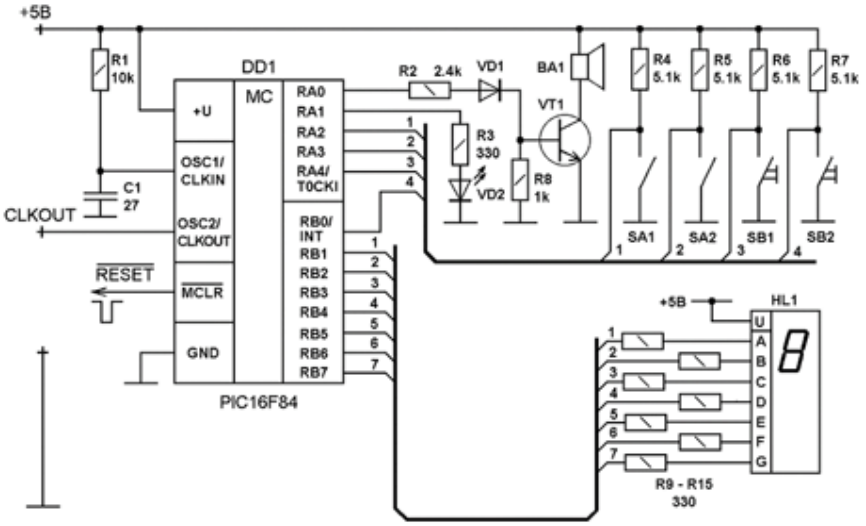

Рис. 7.1. Схема макета.

Макет питается от источника стабилизированного напряжения +5В. Тактовая частота МК задается RC-цепью и составляет около 2 МГц. К линии RA0 порта А подключен биполярный транзистор в ключевом режиме, нагруженный на динамик BA1. Звучание динамика обеспечивается подачей на выход RA0 изменяющегося сигнала в звуковом диапазоне. К линии RA1 порта А подключен светодиод VD2, светящийся при высоком напряжении на выходе. Тумблеры SA1 и SA2, а также кнопки SB1 и SB2 подключены, соответственно, к линиям RA2 и RA3 порта А, а также к линии RA4 порта А и линии RB0 порта В. Исходное состояние кнопок – разомкнутое, что обеспечивает подачу на соответствующие входы МК высокого уровня сигнала.

Линии RB1 – RB7 порта В обслуживают семисегментный индикатор HL1 с общим анодом. Поэтому свечение сегмента индикатора обеспечивается при низком уровне сигнала на соответствующем выходе порта В. Макет также содержит средства программирования и связи с компьютером, которые на схеме не показаны.

2. Инициализация микроконтроллера макета

Прежде чем переходить к созданию простейших пользовательских программ, необходимо описать используемые в дальнейшем переменные и настроить МК на работу с выбранным макетом. С этой целью мы напишем и подробно рассмотрим листинг исходной программы init.asm, в состав которой будут включаться все остальные программы пользователя (*см. листинг 1, лекции 6*)

Рассмотрим работу этой программы. Вначале она указывает ассемблеру тип используемого МК и систему счисления по умолчанию. Идущие далее ассемблерные директивы EQU определяют ассемблерные константы, используемые в этой и последующих программах. Они позволяют использовать в тексте программы более удобные мнемонические метки, привязанные к структуре конкретного МК, вместо корректных, но более сложных ассемблерных выражений. Указатели TEMPA, TEMPB, COUNT1 и COUNT2 назначают адреса ячеек памяти для хранения промежуточных данных (текущих состояний, переменных циклов и т.п.).

Ассемблерные директивы #define задают строку, замещающую соответствующую метку, каждый раз, когда та будет встречаться в исходном тексте. В нашем случае эти директивы позволяют использовать символические имена, привязанные к схеме макета, вместо физических адресов соответствующих разрядов портов и регистров. При этом необходимо иметь в виду, что символы, которые определены директивой #DEFINE, не могут быть просмотрены симулятором. Поэтому для просмотра необходимо использовать физические адреса портов и регистров.

Директива ORG 0x00 устанавливает стартовый адрес программного кода равным 0, т.е. соответствующим начальному состоянию счетчика команд МК после сброса. Команда GOTO BEGIN вместе с ассемблерной директивой ORG 0x005 и меткой BEGIN обеспечивают переход на адрес памяти программ 0x005, начиная с которого и размещается основная часть программы. Это необходимо для того, чтобы обойти адрес 0x004, используемый в качестве вектора прерывания, и тем самым зарезервировать его для возможных будущих применений.

Затем с помощью команды CALL INIT_PORTS производится вызов подпрограммы инициализации портов. Вначале подпрограмма инициализации устанавливает в высокое (единичное) состояние выходные триггеры данных. Эта операция рекомендуется разработчиком МК для того, чтобы исключить неопределенность в состояниях регистров портов. Затем командой BSF STATUS,RP0 производится переключение на банк 1 памяти данных, где расположены регистры управления направлением передачи информации TRISA и TRISB. С помощью команд MOVLW 0x1C и MOVWF TRISA линии RA0 и RA1 порта A настраиваются на вывод, а остальные – на ввод. Команды MOVLW 0x01 и MOVWF TRISB настраивают линию RB0 порта B на ввод, а остальные – на вывод. С помощью команды BCF STATUS,RP0 производится возврат в банк 0, где располагаются необходимые для работы программы регистры и порты.

Поскольку в процессе работы с макетом перенастройка портов не производится, и введенных переменных достаточно для работы всех рассматриваемых учебных задач, они будут далее рассматриваться включенными по умолчанию в состав исходной программы init.asm. При написании учебных задач будет по возможности использоваться метод структурного программирования, при котором прикладная программа строится из некоторого набора программных модулей, каждый из которых реализует определенную процедуру обработки данных. При этом каждый из программных модулей имеет только одну точку входа и одну точку выхода. Введенные однажды программные модули могут использоваться под своим именем в других прикладных программах.

II. Примеры завершенных программ.

Начнем программирование задач с написания программы, которая считывает состояние кнопки SB1 и выводит его на светодиодный индикатор VD2 так, что не нажатому состоянию кнопки (высокому уровню сигнала на входе RA4) соответствует светящееся состояние светодиода, и наоборот.

Листинг 2.

```
;основная программа
LOOP
   CLRWDT              ;сброс сторожевого таймера
   CALL     GET_RA     ;вызов подпрограммы GET_RA
   CALL     SB1_VD2    ;вызов подпрограммы SB1_VD2
   GOTO     LOOP       ;переход к метке LOOP для
                       ;повторения процесса
;
GET_RA                 ;подпрограмма чтения состояния
                       ;порта A
   MOVF     PORTA,W    ;чтение состояния порта A в W
   MOVWF    TEMPA      ;пересылка W в TEMPA
   RETURN              ;возврат из подпрограммы
;
SB1_VD2                ;подпрограмма вывода на светодиод
                       ;VD2 состояния кнопки SB1 (разряда 4
                       ;регистра TEMPA)
   BTFSS    TEMPA,4    ;пропустить команду, если
                       ;TEMPA,4=1 (кнопка не нажата)
   GOTO     P0         ;перейти на P0
   BSF      VD2        ;зажечь светодиод VD2
P0
   BTFSC    TEMPA,4    ;пропустить команду, если
                       ;TEMPA,4=0 (кнопка нажата)
   GOTO     P1         ;перейти на P1
   BCF      VD2        ;погасить светодиод
P1
   RETURN
;
```

Основная программа содержит замкнутый цикл LOOP – GOTO LOOP, необходимый для периодического повторения цикла контроля состояния кнопки и вывода его на индикатор. Команда CLRWDT исключает влияние возможного сброса по переполнению сторожевого таймера на работу программы. Две следующие команды осуществляют вызов подпрограмм GET_RA и SB1_VD2. Первая из них (GET_RA) вначале считывает текущее состояние порта A, которое помещается в рабочий регистр W. Поскольку рабочий регистр может потребоваться при исполнении других команд, его состояние записывается в регистр TEMPA, используемый здесь для временного хранения состояния порта A. Таким образом, после возврата из подпрограммы GET_RA в разряде 4 регистра TEMPA содержится информация о состоянии кнопки SB1: «1» – не нажата, «0» – нажата.

Подпрограмма SB1_VD2 анализирует состояние разряда 4 регистра TEMPA и, в зависимости от него, зажигает или гасит светодиод. В системе команд МК PIC16F84 нет команд условного перехода, поэтому для организации проверки того или иного условия используются команды, позволяющие пропустить выполнение следующей команды программы, в зависимости от состояния определенного бита в заданном регистре (BTFSS и BTFSC). В частности, команда BTFSS TEMP,4 пропускает исполнение команды GOTO P0, если TEMP,4 = 1 (кнопка не нажата). Тем самым реализуется команда BSF VD2, которая зажигает светодиод VD2. Затем анализируется условие TEMP,4 = 0 (кнопка нажата) и, если оно имеет место, светодиод гасится.

Возможна более простая реализация заданного алгоритма, поскольку нажатое состояние кнопки исключает не нажатое (и наоборот), но представленный вариант более нагляден.

Рассмотрим более сложный вариант программы, предусматривающий зажигание светодиода VD2 только при следующем состоянии тумблеров и кнопок макета: SA1 = 1, SA2 = 1, SB1 = 1 и SB2 = 0.

Листинг 3.

```
;основная программа
   LOOP
   CLRWDT                ;сброс сторожевого таймера
   CALL      GET_RA      ;вызов подпрограммы GET_RA
   CALL      GET_RB      ;вызов подпрограммы GET_RB
   CALL      ZAG_1110    ;вызов подпрограммы ZAG_1110
   GOTO      LOOP        ;переход к метке LOOP для
                         ;повторения процесса
;
GET_RB                   ;подпрограмма чтения состояния
                         ;порта B
   MOVF      PORTB,W     ;чтение состояния порта B в W
   MOVWF     TEMPB       ;пересылка W в TEMPB
   RETURN
;
ZAG_1110                 ;зажигает светодиод VD2 только при
                         ;следующем состоянии тумблеров и
                         ;кнопок макета:
                         ;SA1 = SA2 = SB1 = 1 и SB2 = 0
```

```
   BTFSS     TEMPA,2     ;пропустить команду, если
   GOTO      P0          ;TEMPA,2=1
   BTFSS     TEMPA,3     ;пропустить команду, если
   GOTO      P0          ;TEMPA,3=1
   BTFSS     TEMPA,4     ;пропустить команду, если
   GOTO      P0          ;TEMPA,4=1
   BTFSC     TEMPB,0     ;пропустить команду, если
   GOTO      P0          ;TEMPB,0=0
   BSF       VD2         ;зажечь светодиод VD2
   GOTO      P1
P0
   BCF       VD2         ;погасить светодиод VD2
P1
   RETURN
;
   INCLUDE   GET_RA.ASM
;
```

Подпрограммы GET_RA и GET_RB помещают в регистры TEMPA и TEMPB текущие состояния портов A и B, соответственно. Подпрограмма ZAG_1110 анализирует состояния разрядов 2,3 и 4 регистра TEMPA и разряда 0 регистра TEMPB, и при условии TEMPA,2,3,4 = 1,1,1 и TEMPB,0 = 0, зажигает светодиод VD2. При невыполнении хотя бы одного из этих условий светодиод гасится.

Использование директивы INCLUDE GET_PORTA.ASM позволяет включать уже отлаженные модули подпрограмм в текущую программу. Для того чтобы этой возможностью можно было воспользоваться, необходимо сохранять отлаженные модули в виде отдельных ассемблерных файлов.

Попробуем теперь использовать семисегментный индикатор для контроля состояния тумблеров макета. Вначале напишем программу, которая выводит на индикатор HL семисегментное изображение любого двоичного числа от 0b до 1111b в шестнадцатеричном представлении.

Листинг 4.

```
;основная программа
   LOOP
   CLRWDT                ;сброс сторожевого таймера
   MOVLW     0x0A        ;пересылка константы 0A в W
   CALL      SEV_SEG     ;вызов подпрограммы SEVEN_SEG
   MOVWF     PORTB       ;пересылка W в PORTB
   GOTO      LOOP        ;переход к метке LOOP для
                         ;повторения процесса
;
SEV_SEG                  ;подпрограмма обслуживания
                         ;семисегментного индикатора
   ANDLW     0x0F        ;маскирование 4-х младших разрядов
                         ;W и обнуление 4-х старших
   ADDWF     PCL,F       ;сложение W с PCL и пересылка
                         ;результата в PCL
   RETLW     0x80        ;возврат из подпрограммы с 80 в W
   RETLW     0xF2        ;возврат из подпрограммы с F2 в W
```

```
   RETLW     0x48        ;возврат из подпрограммы с 48 в W
   RETLW     0x60        ;возврат из подпрограммы с 60 в W
   RETLW     0x32        ;возврат из подпрограммы с 32 в W
   RETLW     0x25        ;возврат из подпрограммы с 25 в W
   RETLW     0x04        ;возврат из подпрограммы с 04 в W
   RETLW     0xF0        ;возврат из подпрограммы с F0 в W
   RETLW     0x00        ;возврат из подпрограммы с 00 в W
   RETLW     0x20        ;возврат из подпрограммы с 20 в W
   RETLW     0x10        ;возврат из подпрограммы с 10 в W
   RETLW     0x06        ;возврат из подпрограммы с 06 в W
   RETLW     0x8C        ;возврат из подпрограммы с 8C в W
   RETLW     0x42        ;возврат из подпрограммы с 42 в W
   RETLW     0x0C        ;возврат из подпрограммы с 0C в W
   RETLW     0x1C        ;возврат из подпрограммы с 1C в W
;
```

Программа начинает свою работу с пересылки константы 0x0A в рабочий регистр W. Затем производится вызов подпрограммы обслуживания семисегментного индикатора SEV_SEG. Работа подпрограммы SEV_SEG начинается с маскирования 4-х младших разрядов W и обнуления 4-х старших. Тем самым из анализа исключаются старшие разряды передаваемого из рабочего регистра W числа. Затем маскированное содержимое регистра W добавляется к текущему состоянию младшего байта счетчика команд PCL, и результат помещается в PCL. Таким образом, производится дополнительное смещение счетчика команд на величину, которая была передана в рабочем регистре. Например, если было W=0, то содержимое счетчика команд не изменится, и будет выполнена следующая команда RETLW 0x80, которая вызовет возврат из подпрограммы с записью 0x80 = B'1000000' в регистр W. Если, как было в при веденной программе, W=0A, то к содержимому PCL будет добавлено число 0x0A, и произойдет дополнительное смещение на 10 шагов. В результате будет выполнена команда RETLW 0x10, которая вызовет возврат из подпрограммы с записью 0x10 = B'0001000' в регистр W.

После возврата из подпрограммы производится пересылка W в PORTB и отображение его состояния на семисегментном индикаторе HL. В частности, если W = 0, то при выводе 1000000b на порт B семисегментный индикатор покажет 0, а при W = A покажет A.Таким образом, может быть отображено любое 4-разрядное двоичное число.

Метод прямого управления счетчиком команд, использованный в подпрограмме SEV_SEG, может применяться для реализации табличной конвертации чисел. При этом необходимо иметь в виду, что данный метод не позволяет конвертировать более 256 значений в одной таблице. Кроме того, программа табличной конвертации должна целиком располагаться внутри 256-байтного блока во избежание переполнения младшего байта счетчика команд.

Используя подпрограмму SEV_SEG, напишем теперь программу, которая читает состояния тумблеров SA1 и SA2 и выводит на индикатор соответствующее число.

Листинг 5.

```
;основная программа
   LOOP
   CLRWDT                ;сброс сторожевого таймера
   CALL      GET_RA      ;вызов подпрограммы GET_RA
   RRF       TEMPA,F     ;сдвиг вправо на один разряд
                         ;через перенос
   RRF       TEMPA,W     ;сдвиг вправо на один разряд
                         ;через перенос
   ANDLW     0x03        ;маска на два младших разряда
   CALL      SEV_SEG     ;вызов подпрограммы SEVEN_SEG
   MOVWF     PORTB       ;пересылка W в PORTB
   GOTO      LOOP        ;переход к метке LOOP для
                         ;повторения процесса
;
   INCLUDE   GET_RA.ASM
   INCLUDE   SEV_SEG.ASM
;
```

Подпрограмма GET_RA помещает в регистр TEMPA текущее состояние порта A. Таким образом, в разрядах 2 и 3 регистра TEMPA хранится текущее состояние тумблеров SA1 и SA2. Для того чтобы биты состояния тумблеров заняли позиции 0 и 1 регистра TEMPA, производится два сдвига вправо через перенос, причем результат второго сдвига помещается в регистр W. Затем накладывается маска на два младших разряда рабочего регистра и производится вызов подпрограммы SEV_SEG. После выхода из подпрограммы результат подается на порт B и отображается на индикаторе.

Рассмотрим теперь программы, работающие в реальном масштабе времени, т.е. выдающие сигналы определенной длительности и частоты следования, либо учитывающие временные параметры входных сигналов. Основным элементом таких программ является подпрограмма формирования временной задержки. Рассмотрим один из возможных вариантов такой подпрограммы с использованием программных методов формирования задержки, т.е. без применения встроенного таймера.

Листинг 6.

```
;основная программа
   MOVLW     0xL         ;пересылка константы H'L' в W
   CALL      DELAY       ;вызов подпрограммы DELAY
;
   DELAY                 ;подпрограмма формирования
                         ;задержки времени
   MOVWF     COUNT1      ;загрузка W в регистр COUNT1
LOOPD
   DECFSZ    COUNT1,F    ;декремент COUNT1
   GOTO      LOOPD       ;повторение цикла H'L' раз
   RETURN                ;возврат из подпрограммы
;
```

Основная программа производит вызов подпрограммы DELAY с некоторой константой L в рабочем регистре W, определяющей число внутренних циклов подпрограммы. Подпрограмма DELAY начинает свою работу с загрузки содержимого рабочего регистра в регистр пользователя COUNT1. Команда DECFSZ COUNT1,F уменьшает на единицу содержимое регистра COUNT1 и проверяет его на равенство нулю. Нулевое состояние регистра COUNT1 приводит к выходу из цикла и возврату из подпрограммы. Для исполнения каждого внутреннего цикла требуется три машинных цикла МК (1 цикл на исполнение команды DECFSZ при ненулевом результате и 2 цикла на каждую команду GOTO). Выход из подпрограммы DELAY потребует 4-х циклов (2 цикла на исполнение команды DECFSZ при нулевом результате и 2 цикла на RETURN). Если добавить к этому еще 4 цикла, необходимых для загрузки константы в рабочий регистр , вызова подпрограммы и загрузки регистра пользователя COUNT1, то общее время исполнения подпрограммы DELAY (задержка) составит

$$T_D = 4 + 3*(L - 1) + 4 = 5 + 3*L \text{ циклов,}$$

где L – константа, переданная через рабочий регистр в подпрограмму DELAY.

При тактовой частоте fosc = 2МГц время цикла равно $t_ц$ = 2 мкс, поэтому при загрузке L = H'00' = .0 максимальный формируемый интервал времени составит 1,55 мс. Такой результат связан с тем, что команда DECFSZ сначала декрементирует содержимое регистра (H'00' – 1 = H'FF'), а затем уже анализирует результат.

Минимальный формируемый интервал времени составит при тех же условиях 5 циклов или 10 мкс. Для получения такого интервала необходимо перед вызовом подпрограммы DELAY загрузить в рабочий регистр число 0x01.

Для расширения верхней границы формируемых временных интервалов, а также с целью повышения удобства работы с подпрограммой, можно добавить в цикл LOOPD одну или несколько дополнительных команд, в качестве которых чаще всего используется команда NOP. Для примера рассмотрим подпрограмму формирования задержки времени DELAY_C

Листинг 7.

```
;
   DELAY_C                      ;подпрограмма формирования
                                ;задержки времени (вариант C)
   MOVWF      MOVWF COUNT1      ;загрузка W в регистр COUNT1
LOOPD
   NOP                          ;пустая команда
   DECFSZ     COUNT1,F          ;декремент COUNT1
   GOTO       LOOPD             ;повторение цикла H'L' раз
   RETURN                       ;возврат из подпрограммы
;
```

Общее время исполнения подпрограммы DELAY_C, включая ее вызов, составит

$$T_D = 4 + 4*(L - 1) + 4 = 4 + 4*L \text{ циклов.}$$

При тактовой частоте fosc = 2МГц и загрузке константы L = H'F9' = .249 формируемый интервал времени составит ровно 2 мс. Уменьшение константы на единицу уменьшает формируемый временной интервал на 8 мкс. В частности, при L = .124 образуется задержка в 1 мс.

Для формирования больших задержек времени, лежащих в диапазоне долей и единиц секунд, такой подход неудобен. В этом случае используются вложенные циклы, как показано в следующем примере.

Листинг 8.

```
;основная программа
   MOVLW     0xL          ;пересылка константы H'L' в W
   CALL      DELAY_D      ;вызов подпрограммы DELAY_D
;
DELAY_D                   ;подпрограмма формирования
                          ;большой задержки времени (вариант D)
   MOVWF     COUNT2       ;загрузка W в регистр COUNT2
   CLRF      COUNT1       ;сброс содержимого регистра COUNT1
LOOPD
   DECFSZ    COUNT1,F     ;декремент COUNT1
   GOTO      LOOPD        ;повторение цикла 256 раз
   CLRWDT                 ;сброс сторожевого таймера
   DECFSZ    COUNT2,F     ;декремент COUNT2
   GOTO      LOOPD        ;повторение цикла H'L' раз
   RETURN                 ;возврат из подпрограммы
;
```

Время исполнения внутреннего цикла подпрограммы DELAY_D составляет 3*256 + 4 машинных циклов МК, поэтому общая задержка составит $T_D = 5 + (3*256 + 4)*L$ циклов.

При тактовой частоте fosc = 2МГц время цикла равно tц = 2 мкс, поэтому при загрузке L = H'00' = .0 максимальный формируемый интервал времени составит около 0,4 с.

Поскольку формируемый интервал времени достаточно велик, во внешний цикл включена команда сброса сторожевого таймера.

Интервал времени 0,4 с не совсем удобен для получения задержек времени, кратных секунде, поэтому рассмотрим еще один вариант подпрограммы формирования больших задержек времени с дополнительной командой NOP во внутреннем цикле.

Листинг 9.

```
;
DELAY_E                   ;подпрограмма формирования
                          ;большой задержки времени (вариант E)
   MOVWF     COUNT2       ;загрузка W в регистр COUNT2
   CLRF      COUNT1       ;сброс содержимого регистра COUNT1
LOOPD
   NOP                    ;пустая команда
   DECFSZ    COUNT1,F     ;декремент COUNT1
   GOTO      LOOPD        ;повторение цикла 256 раз
   CLRWDT                 ;сброс сторожевого таймера
```

```
   DECFSZ     COUNT2,F    ;декремент COUNT2
   GOTO       LOOPD       ;повторение цикла H'L' раз
   RETURN                 ;возврат из подпрограммы
;
```

Время исполнения внутреннего цикла подпрограммы DELAY_E составляет 4*256 + 4 машинных циклов МК, поэтому общая задержка составит $T_D = 5 + (4*256 + 4)*L$ циклов.

При тактовой частоте fosc = 2МГц и при загрузке L = H'F3' = .243 формируемый интервал времени составит около 0,5 с при погрешности не более 0,2%. Если необходима более высокая точность, можно вставить необходимое количество пустых операций во внешний цикл формирования задержки.

Рассмотрим далее несколько программ с использованием подпрограмм формирования задержки времени. Начнем с написания программы, которая подает звуковой сигнал на динамик BA1 при нажатии на кнопку SB1. Динамик будет звучать только в том случае, если на выход RA0 будет подан периодически изменяющийся сигнал. Для того чтобы звук был хорошо слышен, его частота должна находиться вблизи максимума слышимости человеческого уха. Выберем частоту звучания равной 1 КГц, что соответствует периоду следования импульсов сигнала 1 мс.

Листинг 10.

```
;основная программа
LOOP
   CLRWDT                 ;сброс сторожевого таймера
   CALL       GET_RA      ;вызов подпрограммы GET_PORTA
   CALL       SB1_BA1     ;вызов подпрограммы SB1_BA1
   GOTO       LOOP        ;переход к метке LOOP для
                          ;повторения процесса
;
SB1_BA1                   ;подпрограмма подачи звука на
                          ;динамик BA1 при нажатии на кнопку
                          ;SB1
   BTFSC      TEMPA,4     ;пропустить команду, если
                          ;TEMPA,4=0 (кнопка нажата)
   GOTO       B0          ;перейти на B0
   BSF        BA1         ;подача высокого уровня на RA0
   MOVLW      0x3E        ;пересылка константы
                          ;H'3E' = .62 в W
   CALL       DELAY_C     ;вызов подпрограммы DELAY_C
   BCF        BA1         ;подача низкого уровня на RA0
   MOVLW      0x3E        ;пересылка константы
                          ;H'3E' = .62 в W
   CALL       DELAY_C     ;вызов подпрограммы DELAY_C
B0
   RETURN
;
   INCLUDE    GET_RA.ASM
   INCLUDE    DELAY_C.ASM
;
```

Как и раньше, подпрограмма GET_RA считывает текущее состояние порта A, которое затем передается в регистр TEMPA. Подпрограмма SB1_BA1 анализирует состояние разряда 4 регистра TEMPA и, в зависимости от результата, озвучивает динамик BA1 или нет. Необходимая выдержка линии RA0 в единичном и нулевом состояниях обеспечивается подпрограммой DELAY_C с параметром L = H'3E' = .62. Это соответствует времени задержки около 0,5 мс, что и дает в результате необходимую частоту следования сигнала 1 Кгц.

Рассмотрим далее программу, которая заставляет мигать светодиод VD2 при нажатии на кнопку SB1. Для того чтобы мигания были хорошо видны, выберем их частоту равной 1 Гц.

Листинг 11.

```
;основная программа
LOOP
   CLRWDT                  ;сброс сторожевого таймера
   CALL      GET_RA        ;вызов подпрограммы GET_RA
   CALL      SB1_VD2M      ;вызов подпрограммы
                           ;SB1_VD2M
   GOTO      LOOP          ;переход к метке LOOP для
                           ;повторения процесса
;
SB1_VD2M                   ;подпрограмма мигания светодиода
                           ;VD2 при нажатии на кнопку SB1
   BTFSC     TEMPA,4       ;пропустить команду, если
                           ;TEMPA,4=0 (кнопка нажата)
   GOTO      V0            ;перейти на V0
   BSF       VD2           ;зажечь светодиод VD2
   MOVLW     0xF3          ;пересылка константы
                           ;H'F3' = .243 в W
   CALL      DELAY_E       ;вызов подпрограммы DELAY_E
   BCF       VD2           ;погасить светодиод
   MOVLW     0xF3          ;пересылка константы
                           ;H'F3' = .243 в W
   CALL      DELAY_E       ;вызов подпрограммы DELAY_E
V0
   BTFSS     TEMPA,4       ;пропустить команду, если
                           ;TEMPA,4=1 (кнопка не нажата)
   GOTO      V1            ;перейти на V1
   BCF       VD2           ;погасить светодиод
V1
   RETURN
;
   INCLUDE   GET_RA.ASM
   INCLUDE   DELAY_E.ASM
;
```

Программа работает почти так же, как и предыдущая. Первое отличие заключается в том, что светодиод принудительно гасится при не нажатой кнопке. Второе отличие заключается в величине интервала времени, который составляет здесь 0,5 с и формируется подпрограммой DELAY_E.

Подпрограммы формирования задержки времени могут быть также полезны при работе с такими внешними источниками сигналов, как тумблеры, кнопки, переключатели и т.п. Дело в том, что все механические коммутаторы имеют одно негативное свойство, известное как «дребезг» контактов, которое обусловлено механическими колебаниями контактов при их замыкании и размыкании. Длительность колебаний составляет обычно несколько миллисекунд, в течение которых на вход МК может поступать пачка импульсов вместо идеального перепада.

Аппаратные способы борьбы с «дребезгом» контактов основаны на использовании RS-триггеров, одновибраторов или триггеров Шмитта. В устройствах на основе МК подавление «дребезга» контактов обычно осуществляется программными способами, которые основаны на повторном считывании состояния линии порта через определенное время.

В качестве примера рассмотрим «бездребезговый» вариант подпрограммы чтения состояния порта А.

Листинг 12.

```
;
GET_RAD                 ;подпрограмма чтения состояния
                        ;порта A в регистр TEMPA
                        ;с подавлением "дребезжания"
DD
   MOVF       PORTA,W   ;чтение состояния порта A в W
   ANDLW      0x1C      ;наложение маски b'00011100'
                        ;на неиспользуемые биты W
   MOVWF      TEMPA     ;пересылка W в TEMPA
   CLRWDT               ;сброс сторожевого таймера WDT
   MOVLW      0x0A      ;пересылка константы
                        ;H'0A' = .10 в W
   CALL       DELAY_E   ;вызов подпрограммы DELAY_E
   MOVF       PORTA,W   ;чтение состояния порта A в W
   ANDLW      0x1C      ;наложение на W маски b'00011100'
   SUBWF      TEMPA,W   ;вычитание W из TEMPA
   BTFSS      Z         ;пропустить команду, если результат
                        ;нулевой
   GOTO       DD        ;перейти на метку DD
   RETURN
;
   INCLUDE    DELAY_E.ASM
;
```

Суть работы подпрограммы заключается в повторном чтении состояния порта А спустя некоторое время после предыдущего и сравнении его с прежним значением. Константа H'0A' = .10, пересылаемая в регистр W перед вызовом подпрограммы DELAY_E, обеспечивает значение задержки времени около 20 мс - этого, как правило, достаточно для завершения переходных процессов при переключении механических коммутаторов. Маскирование неиспользуемых разрядов порта повышает надежность работы подпрограм-

мы. Сброс сторожевого таймера перед вызовом подпрограммы задержки нужен для исключения сброса МК между двумя процедурами опроса порта А.

Рассмотрим теперь работу программы, которая использует некоторые из разработанных ранее подпрограмм. Пусть целью работы программы является подсчет числа нажатий на кнопку SB1 с выводом результата на семисегментный индикатор в шестнадцатиричном коде.

Листинг 13.

```
;основная программа
   CLRF      COUNT3       ;сброс счетчика нажатий
LOOP
   CLRWDT                 ;сброс сторожевого таймера
   CALL      GET_RAD      ;вызов подпрограммы GET_RAD
   BTFSC     TEMPA,4      ;проверка нажатия SB1
   GOTO      LOOP         ;если не нажата - возврат
                          ;на метку LOOP
   INCF      COUNT3,F     ;инкремент счетчика
   MOVF      COUNT3,W     ;пересылка содержимого
                          ;счетчика в рабочий регистр
   CALL      SEV_SEG      ;вызов подпрограммы SEVEN_SEG
   MOVWF     PORTB        ;пересылка W в PORTB
TEST
   CALL      GET_RAD      ;вызов подпрограммы GET_RAD
   BTFSS     TEMPA,4      ;проверка нажатия SB1
   GOTO      TEST         ;если еще нажата - возврат
                          ;на метку TEST
   GOTO      LOOP         ;возврат на метку LOOP
;
   INCLUDE   GET_RAD.ASM
   INCLUDE   SEV_SEG.ASM
;
```