Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

В. Н. Левкович, Е. Н. Каленкович, А. А. Казека

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальностям 1-39 01 01 «Радиотехника», 1-39 01 02 «Радиоэлектронные системы», 1-39 01 03 «Радиоинформатика», 1-39 01 04 «Радиоэлектронная защита информации»

Минск БГУИР 2012

УДК 004.31-022.53(076.5)
ББК 32.973.26-04я73
Л37

Р е ц е н з е н т ы:
кафедра информатики и компьютерных систем
учреждения образования «Белорусский государственный университет»
(протокол №11 от 17.05.2011 г.);

профессор кафедры программного обеспечения учреждения образования «Высший государственный колледж связи», кандидат технических наук, доцент О. Р. Ходасевич

Левкович, В. Н.

Л37 Микропроцессорные устройства. Лабораторный практикум : учеб.-метод. пособие / В. Н. Левкович, Е. Н. Каленкович, А. А. Казека. – Минск : БГУИР, 2012. – 92 с. : ил.

ISBN 978-885-488-713-5.

Практикум содержит описание аппаратно-программной установки, позволяющей выполнить шесть лабораторных работ по курсу «Микропроцессорные устройства». Тематика работ охватывает наиболее важные программно реализуемые процедуры, используемые в реальном проектировании микропроцессорных устройств. В результате выполнения работ студенты углубляют и закрепляют теоретические знания по архитектуре современных микропроцессоров, а также получают практические навыки по отладке программ и работе с реальной измерительной аппаратурой.

Для студентов специальностей 1-39 01 01 «Радиотехника», 1-39 01 02 «Радиоэлектронные системы», 1-39 01 03 «Радиоинформатика», 1-39 01 04 «Радиоэлектронная защита информации».

УДК 004.31-022.53(076.5)
ББК 32.973.26-04я73

ISBN 978-885-488-713-5

Содержание

1. ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

1.1. Структура лабораторной установки

Лабораторная установка представляет собой аппаратно-программный комплекс. Ее структура показана на рис. 1.1. Центральное место в установке занимает макетная плата с установленным однокристальным микроконтроллером PIC16F886.

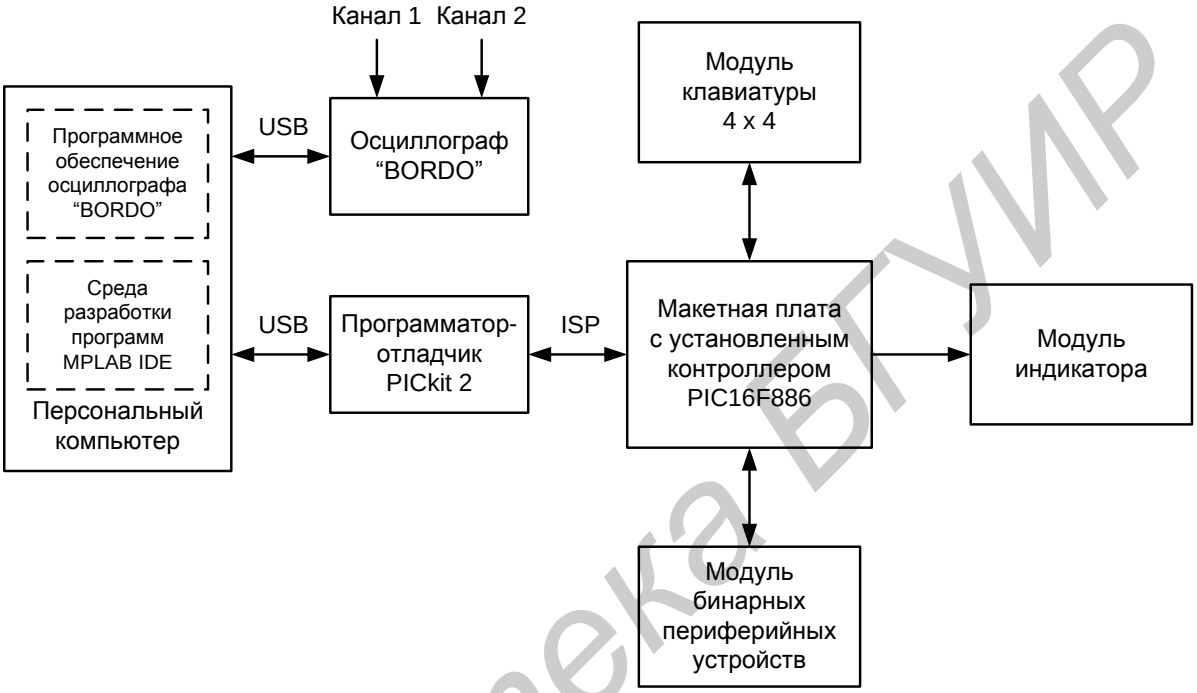


Рис. 1.1. Структура лабораторной установки

К макетной плате посредством разъемов могут подключаться сменные модули бинарных периферийных устройств, цифрового индикатора, клавиатуры. Через программатор-отладчик PICkit2 и USB-интерфейс плата соединяется с компьютером. Питание платы осуществляется напряжением +5 В от USB-интерфейса. Двухканальный осциллограф-приставка «BORDO» предназначен для визуализации и контроля временных параметров сигналов. На компьютере помимо операционной системы установлены пакеты программ, обслуживающие осциллограф-приставку и интегрированную среду MPLAB IDE разработки программ для PIC-контроллеров.

1.2. Принципиальные схемы функциональных модулей

Принципиальная электрическая схема макетной платы показана на рис. 1.2, внешний вид ее с обозначениями мест расположения отдельных узлов – на рис. 1.3. Центральное место на плате занимает микроконтроллер PIC16F886 (DD1). К элементам схемы, обеспечивающим функционирование

микроконтроллера, относятся: C1, C2, ZQ1 (кварцевый резонатор) – для тактирования работы микроконтроллера; C3 – для фильтрации помех по цепи питания; SB1, R8, R9 – для ручной подачи сигнала СБРОС (MCLR).

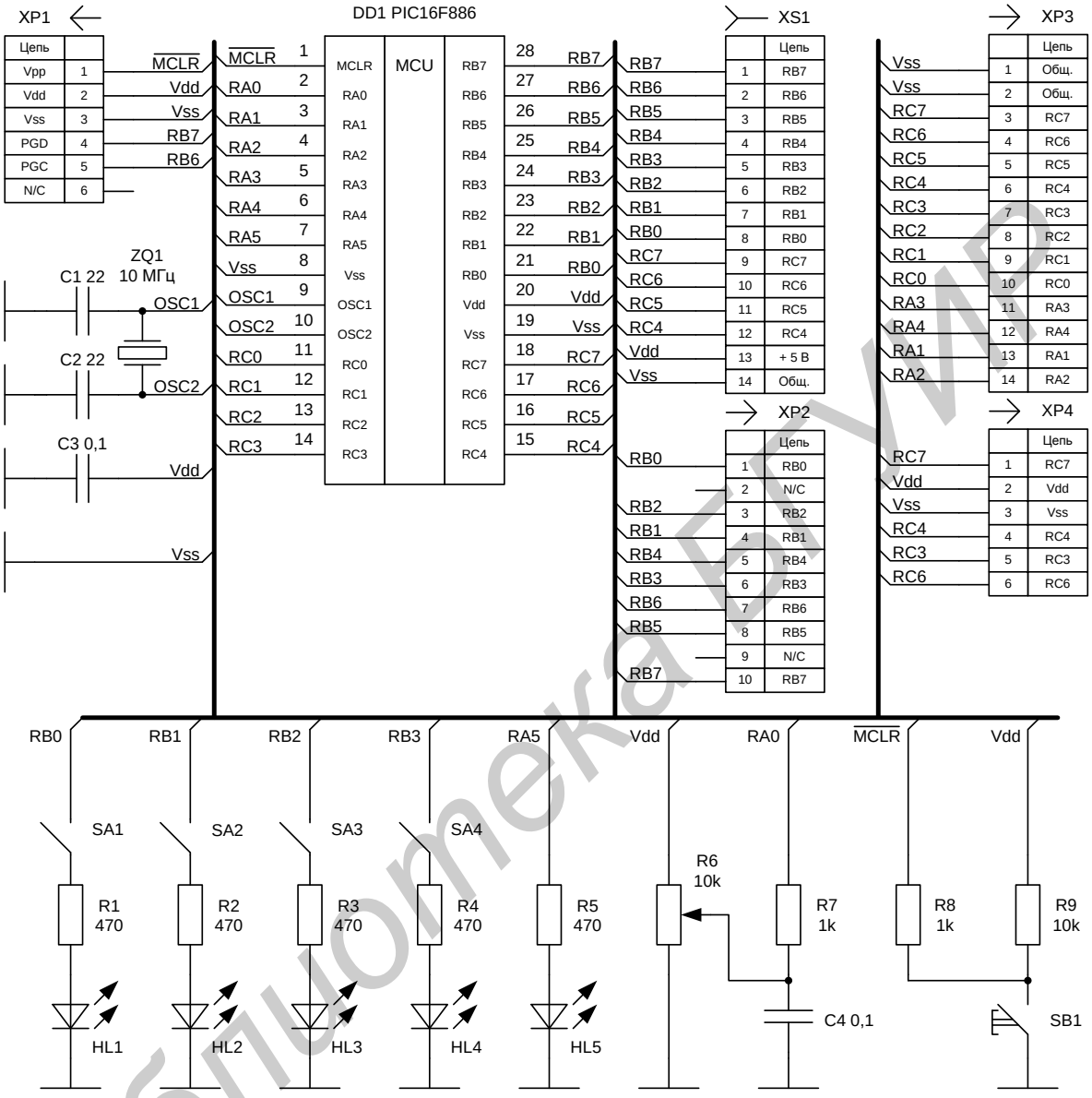


Рис. 1.2. Принципиальная схема макетной платы

Разъем XP1 служит для подключения программатора-отладчика PICkit2, а разъемы XS1, XP2, XP3, XP4 – для подключения периферийных модулей и устройств. Посредством переменного резистора R6 на порт контроллера можно подавать постоянное регулируемое напряжение. Единичные индикаторы на плате HL1 – HL5 являются простейшими периферийными устройствами и используются в качестве флагов для отображения состояния отлаживаемых программ. Резисторы R1 – R5 служат для ограничения токов, протекающих через индикаторы.

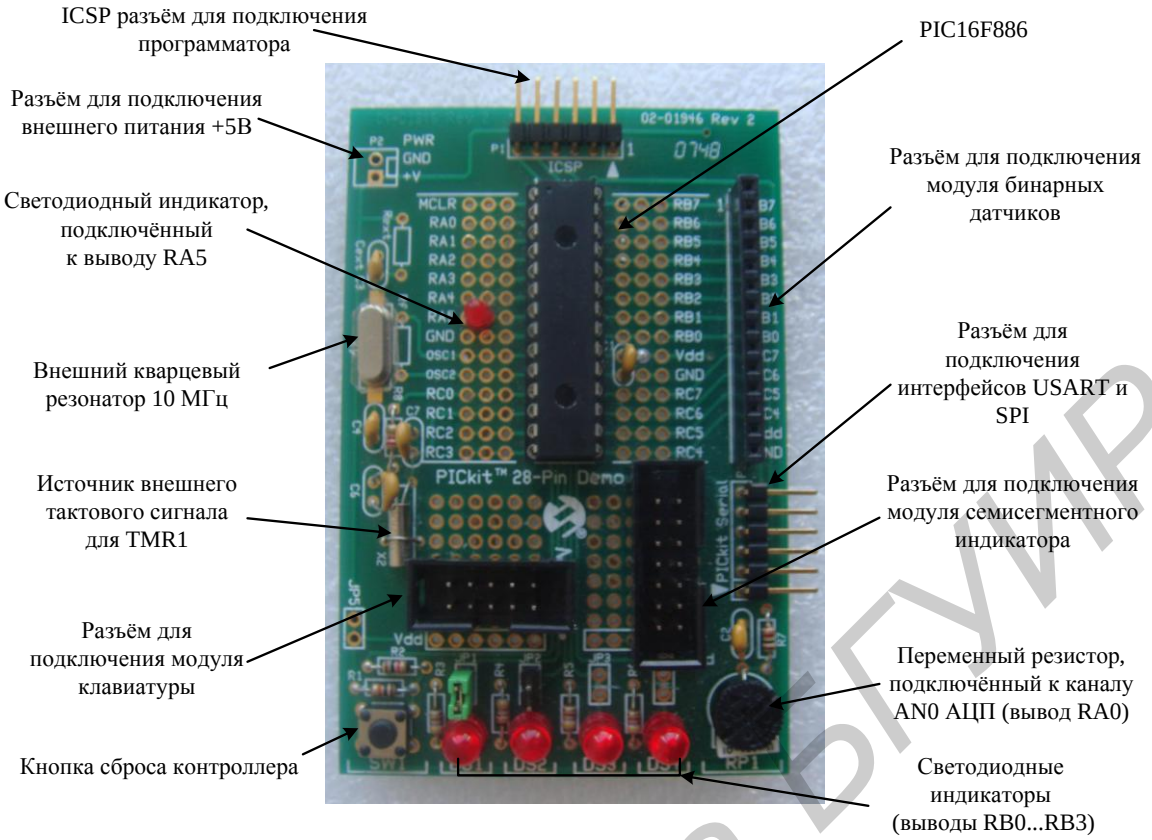


Рис. 1.3. Внешний вид макетной платы

На рис. 1.4 – 1.6 приведены принципиальные схемы сменных периферийных модулей: бинарных устройств ввода/вывода, цифрового четырехразрядного индикатора и 16-кнопочной клавиатуры соответственно.

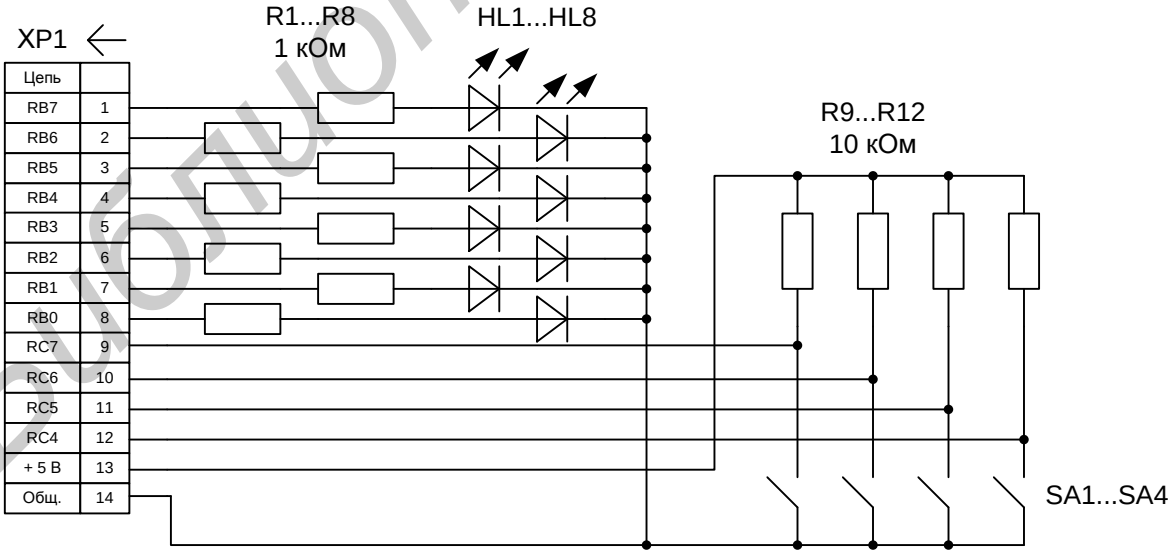


Рис. 1.4. Принципиальная схема модуля бинарных периферийных устройств

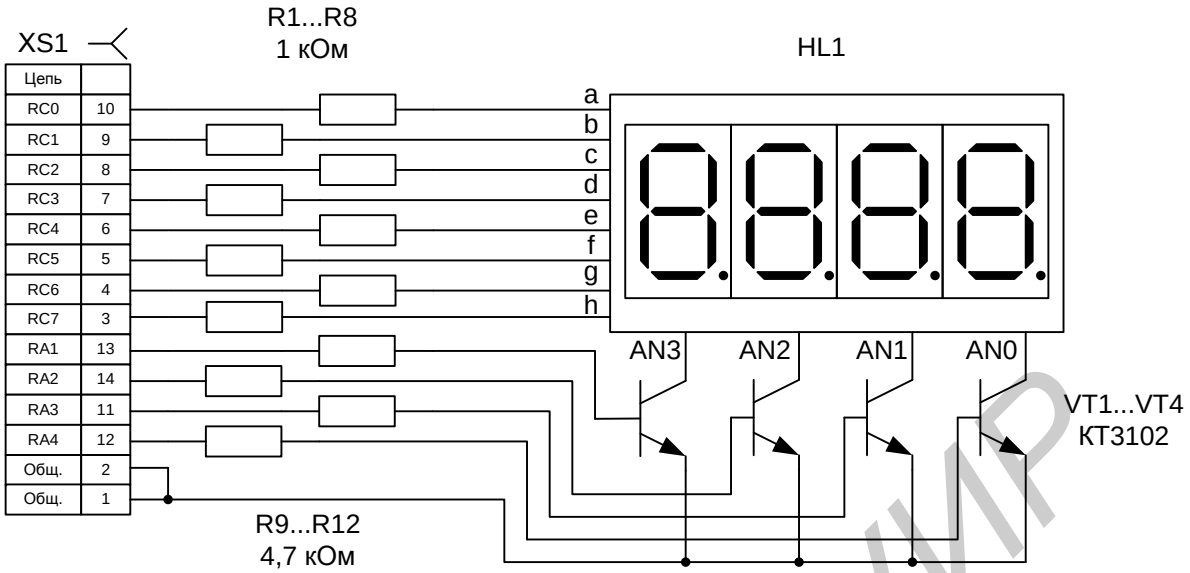


Рис. 1.5. Принципиальная схема модуля цифрового индикатора

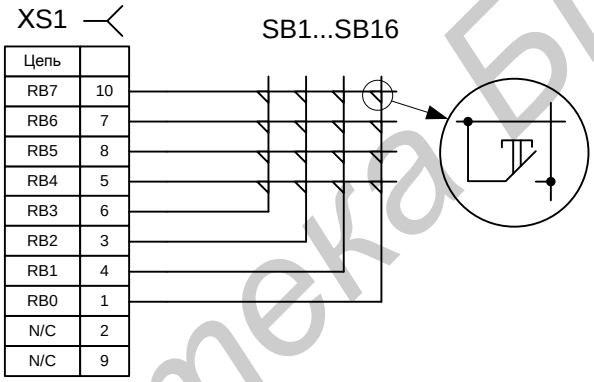


Рис. 1.6. Принципиальная схема модуля клавиатуры

1.3. Архитектура однокристального микроконтроллера PIC16F886

1.3.1. Структурная организация и общие характеристики

В настоящем пособии описание однокристального микроконтроллера PIC16F886 дано лишь в объеме, необходимом для его начального изучения и выполнения лабораторных работ.

Структурная схема PIC16F886 показана на рис. 1.7, назначение выводов отражено в табл. 1.1.

Ядро микроконтроллера PIC16F886 разработано в соответствии с модифицированной гарвардской RISC-архитектурой и изготавливается по высокоскоростной КМОП-технологии. Оно имеет внутреннюю энергонезависимую электрически перепрограммируемую память программ (FLASH) емкостью 8192 14-разрядных слов, восьмибитовую длину машинного слова и 368-байтовую внутреннюю память данных.

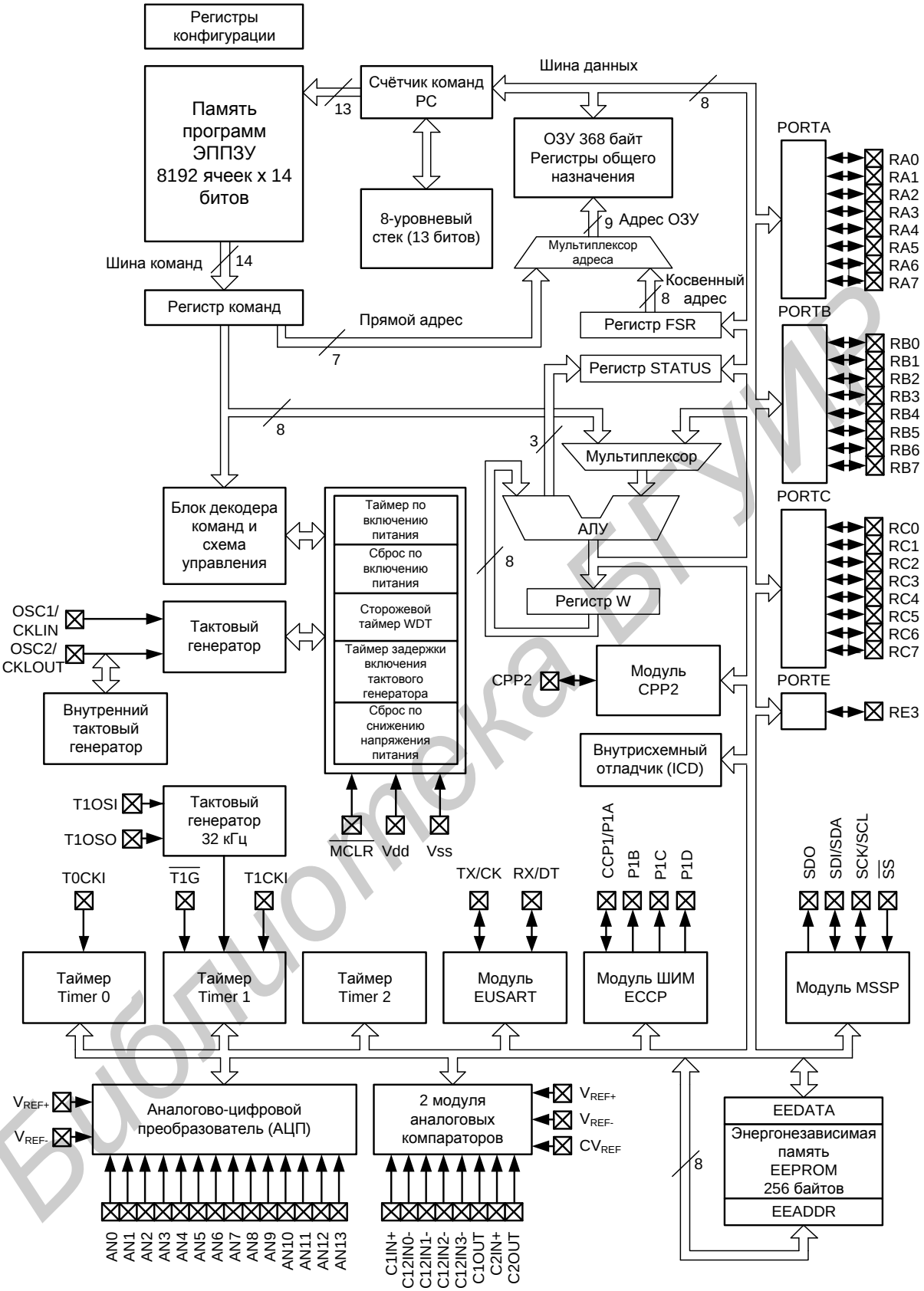


Рис. 1.7. Структурная схема PIC16F886

Таблица 1.1

Название	Функция	Тип входа	Тип выхода	Описание
PORTA				
RA0	RA0	ТТЛ	КМОП	Цифровой вход/выход
	AN0	Ан	–	Канал 0 АЦП
RA1	RA1	ТТЛ	КМОП	Цифровой вход/выход
	AN1	Ан	–	Канал 1 АЦП
RA2	RA2	ТТЛ	КМОП	Цифровой вход/выход
	AN2	Ан	–	Канал 2 АЦП
	V_{REF-}	Ан	–	Вход отрицательного опорного напряжения АЦП
RA3	RA3	ТТЛ	КМОП	Цифровой вход/выход
	AN3	Ан	–	Канал 3 АЦП
	V_{REF+}	Ан	–	Вход положительного опорного напряжения АЦП
RA4	RA4	ТТЛ	КМОП	Цифровой вход/выход
	T0CKI	ST	–	Внешний тактовый вход Timer 0
RA5	RA5	ТТЛ	КМОП	Цифровой вход/выход
	AN4	Ан	–	Канал 4 АЦП
RA6	RA6	ТТЛ	КМОП	Цифровой вход/выход
	OSC2	XTAL	–	Вход для кварцевого резонатора
	CLKOUT	–	КМОП	Выход тактовой частоты $F_{OSC}/4$
RA7	RA7	ТТЛ	КМОП	Цифровой вход/выход
	OSC1	XTAL	–	Вход для кварцевого резонатора
	CLKIN	ST	–	Вход внешней синхронизации/RC тактовый генератор
PORTB				
RB0	RB0	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN12	Ан	–	Канал 12 АЦП
	INT	ST	–	Вход внешнего прерывания
RB1	RB1	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN10	Ан	–	Канал 10 АЦП
RB2	RB2	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN8	Ан	–	Канал 8 АЦП
RB3	RB3	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN9	Ан	–	Канал 9 АЦП
RB4	RB4	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN11	Ан	–	Канал 11 АЦП
RB5	RB5	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	AN13	Ан	–	Канал 13 АЦП
RB6	RB6	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	ICSPCLK	ST	–	Вход тактовой частоты при программировании
RB7	RB7	ТТЛ	КМОП	Цифровой вход/выход. Вход внешнего прерывания
	ICSPDAT	ST	КМОП	Вход данных при программировании
PORTC				
RC0	RC0	ST	КМОП	Цифровой вход/выход
RC1	RC1	ST	КМОП	Цифровой вход/выход
RC2	RC2	ST	КМОП	Цифровой вход/выход
RC3	RC3	ST	КМОП	Цифровой вход/выход
RC4	RC4	ST	КМОП	Цифровой вход/выход
RC5	RC5	ST	КМОП	Цифровой вход/выход
RC6	RC6	ST	КМОП	Цифровой вход/выход
RC7	RC7	ST	КМОП	Цифровой вход/выход
PORTE				
RE3	RE3	ТТЛ	–	Цифровой вход
	nMCLR	ST	–	Вход сброса с внешним подтягивающим резистором
	Vpp	HV	–	Вход напряжения программирования
Vss	Vss	Питание	–	Общий вывод напряжения питания
Vdd	Vdd	Питание	–	Положительный вывод напряжения питания

Обозначения входов: ST – с триггером Шмитта, Ан – аналоговый.

Система команд включает 35 инструкций. Все команды имеют длину в одно слово шириной 14 бит и исполняются за один машинный цикл (200 нс при максимальной тактовой частоте 20 МГц), кроме команд перехода, которые выполняются за два цикла (400 нс).

PIC16F886 имеет прерывание, срабатывающее от 13 источников, и восьмиуровневый аппаратный стек.

Периферия включает в себя три таймера, два аналоговых компаратора, источник опорного напряжения, модуль широтно-импульсной модуляции, последовательный синхронно-асинхронный приемопередатчик EUSART, энергонезависимую память данных EEPROM, 24 линии двунаправленного ввода/вывода и др. В настоящей работе из периферии мы ограничимся рассмотрением лишь цифровых портов ввода/вывода и восьмибитного таймера/счетчика TMR0 с восьмибитным программируемым предварительным делителем.

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

Микроконтроллер способен работать в широком диапазоне тактовых частот – от 0 до 20 МГц, широком диапазоне питающих напряжений – от 2 до 5,5 В, в широком температурном диапазоне – от –40 до +125 °С. Он отличается также низким энергопотреблением: менее 220 мкА при напряжении питания 2 В и тактовой частоте 4 МГц; 11 мкА при питании 2 В и тактовой частоте 32 кГц.

Разработки на базе PIC-контроллеров поддерживаются ассемблером, программным симулятором, внутрисхемным эмулятором, программатором. Эти инструменты объединяются в интегрированной среде разработки программ для PIC-контроллеров MPLAB IDE.

Архитектура микроконтроллера основана на концепции раздельных шин и областей памяти для данных и для программ (гарвардская архитектура). Это увеличивает скорость обмена по сравнению с традиционной принстонской архитектурой, в которой команды и данные передаются по одной и той же шине. Разделение шин команд и данных позволяет увеличить разрядность команды по сравнению с разрядностью данных. Шина данных и память данных (ОЗУ) имеют ширину 8 битов, а программная шина и программная память (ПЗУ) – ширину 14 битов. Такая концепция обеспечивает простую, но мощную систему однословных команд, разработанную так, что битовые, байтовые и регистровые операции работают с высокой скоростью и с пере-

крытием по времени выборок команд и циклов выполнения. 14-битовая ширина программной памяти обеспечивает выборку 14-битовой команды за один цикл. Двухступенчатый конвейер обеспечивает одновременную выборку следующей и исполнение текущей команд. Все команды выполняются за один машинный цикл, исключая команды переходов, которые выполняются за два цикла. В PIC16F886 память программы расположена внутри кристалла. Исполняемая программа может находиться только во внутреннем ПЗУ.

Регистры PIC16F886 разделяются на две функциональные группы: специальные и общего назначения. Специальные регистры используются для управления режимами работы функциональных блоков контроллера, регистры общего назначения (память данных или ОЗУ) – для хранения переменных.

Микроконтроллер PIC16F886 использует прямую и косвенную адресацию всех регистров и ячеек памяти. Все специальные регистры и счетчик команд также адресуются как память данных. Ортогональная (симметричная) система команд позволяет выполнять любую операцию с любым регистром, используя любой из названных выше методов адресации. Это облегчает программирование для PIC16F886 и значительно уменьшает время, необходимое для обучения работе с микроконтроллером.

В микроконтроллере PIC16F886 имеется восьмиразрядное арифметико-логическое устройство (АЛУ) и рабочий регистр (аккумулятор) W. АЛУ выполняет сложение, вычитание, сдвиг, битовые и логические операции. В командах, обрабатывающих два операнда, один из операндов содержится в рабочем регистре W. Второй операнд может быть константой или содержимым любого регистра ОЗУ. В командах с одним операндом операнд может быть содержимым рабочего регистра или любого регистра ОЗУ. Для выполнения всех операций АЛУ используется рабочий регистр W, который не может быть прямо адресован. Результат операции в АЛУ помещается либо в рабочий регистр W, либо в регистр ОЗУ.

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

Входная тактовая частота, поступающая с вывода OSC1/CLKIN, внутри делится на четыре, и из нее формируются четыре циклические неперекрывающиеся тактовые последовательности Q1, Q2, Q3 и Q4. Счетчик команд увеличивается в такте Q1, команда считывается из памяти программы и защелкивается в регистре команд в такте Q4. Команда декодируется и выпол-

няется в течение последующего цикла в тактах Q1...Q4. Временные диаграммы тактирования и выполнения команд изображены на рис. 1.8.

Цикл выполнения команды состоит из четырех тактов: Q1...Q4. Выборка команды и ее выполнение совмещены по времени таким образом, что выборка команды занимает один цикл, а выполнение – следующий цикл. Эффективное время выполнения команды составляет один цикл. Если команда изменяет счетчик команд (например команда CALL), то для выполнения этой команды потребуется два цикла. Цикл выборки начинается с увеличения счетчика команд в такте Q1. В цикле выполнения команды выбранная команда защелкивается в регистр команд в такте Q1. В течение тактов Q2, Q3 и Q4 происходит декодирование и выполнение команды. В такте Q2 считывается память данных (чтение операнда), а запись происходит в такте Q4.

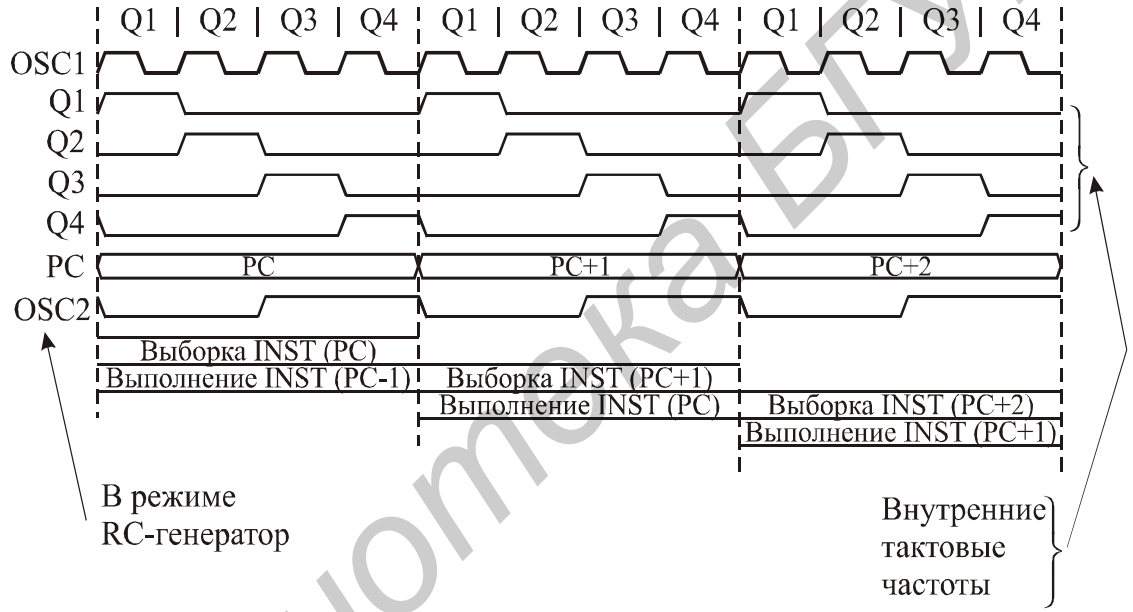


Рис. 1.8. Временные диаграммы тактирования и выполнения команд

1.3.2. Организация памятей программы и данных

Внутренняя память в микроконтроллере PIC16F886 состоит из двух частей: памяти программы и памяти данных. Память программы и память данных имеют раздельные шины, поэтому доступ к ним может происходить одновременно. Память данных делится на регистры общего назначения (ОЗУ) и регистры специального назначения.

Микроконтроллер PIC16F886 имеет 13-разрядный счетчик команд, способный адресовать до 8Кх14 слов памяти программы. При сбросе процессор запускается с адреса 0000h, вектор прерывания расположен по адресу 0004h.

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

На рис. 1.9 показаны две ситуации загрузки значения в счетчик команд PC. В примере сверху запись в счетчик команд PC происходит при записи значения в регистр PCL (PCLATH <4:0> —> PCH). В примере снизу запись значения в счетчик команд PC происходит при выполнении команды CALL или GOTO (PCLATH <4:3> —>PCH).

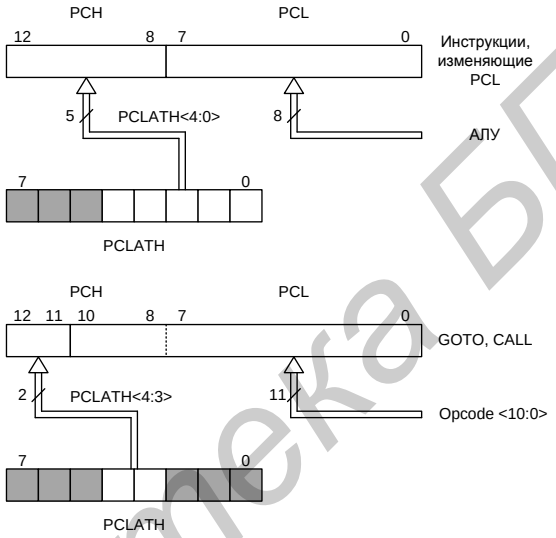


Рис. 1.9. Запись значения в счетчик команд PC

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

PIC16F886 имеет восьмиуровневый 13-разрядный аппаратный стек. Стек не имеет отображения на память программ и память данных, нельзя записать или прочитать данные из стека. Значение счетчика команд заносится в вершину стека при выполнении инструкций перехода на подпрограмму (CALL) или обработки прерываний. Чтение из стека и запись в счетчик команд PC происходит при выполнении инструкций возвращения из подпрограммы или обработки прерываний (RETURN, RETLW, RETFIE), при этом значение регистра PCLATH не изменяется.

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

В микроконтроллере не имеется никаких указателей о переполнении стека.

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

Регистры общего назначения используются для хранения переменных по усмотрению пользователя.

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

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

В табл. 1.2 приведена организация памяти данных PIC16F886.

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

В табл. 1.3 показаны структуры регистров специальных функций и начальные состояния их разрядов после поступления сигнала СБРОС.

1.3.3. Система команд

Каждая команда микроконтроллера PIC16F886 состоит из одного 14-разрядного слова, содержащего код операции (OPCODE), определяющий тип команды, и один или несколько операндов, указывающих операцию команды. Описание полей команд дается в табл. 1.4. Полный список команд приведен в табл. 1.5. Команды разделены на следующие группы: байт-ориентированные команды, бит-ориентированные команды, команды управления и операций с константами.

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

Таблица 1.2

Банк 0		Банк 1		Банк 2		Банк 3	
INDF(1)	00h	INDF(1)	80h	INDF(1)	100h	INDF(1)	180h
TMR0	01h	OPTION_REG	81h	OPTION_REG	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Резерв	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Резерв	18Fh
T1CON	10h	OSCTUNE	90h	Регистры общего назначения 16 байтов	110h	Регистры общего назначения 16 байтов	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h		116h		196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
Регистры общего назначения 96 байт	20h – 6Fh	Регистры общего назначения 80 байт	A0h – EFh	Регистры общего назначения 80 байтов	120h – 16Fh	Регистры общего назначения 80 байтов	1A0h – 1EFh
	70h – 7Fh	Регистры быстрого доступа 70h – 7Fh	F0h – FFh	Регистры быстрого доступа F0h – FFh	170h – 17Fh	Регистры быстрого доступа F0h – FFh	1F0h – 1FFh

☐ Не реализован, читается как “0”.

1: Нефизический регистр.
2: Только для PIC16F887.

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

Таблица 1.3

Адрес	Название	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	Состояние после сброса
Банк 0										
00h	INDF	Регистр данных при косвенной адресации								xxxx xxxx
01h	TMR0	Регистр модуля таймера Timer 0								xxxx xxxx
02h	PCL	Младший байт счётчика команд (PC)								0000 0000
03h	STATUS	IRP	RP1	RP0	nTO	nPD	Z	DC	C	0001 1xxx
04h	FSR	Регистр адреса при косвенной адресации								xxxx xxxx
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx xxxx
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx
09h	PORTE	–	–	–	–	RE3	–	–	–	– – – – x – – –
0Ah	PCLATH	–	–	–	Старшие 5 битов счётчика команд					– – – 0 0000
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
1Eh	ADRESH	Старший байт результата преобразования АЦП								xxxx xxxx
1Fh	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO	ADON	0000 0000
Банк 1										
80h	INDF	Регистр данных при косвенной адресации								xxxx xxxx
81h	OPTION_REG	nRBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
82h	PCL	Младший байт счётчика команд (PC)								0000 0000
83h	STATUS	IRP	RP1	RP0	nTO	nPD	Z	DC	C	0001 1xxx
84h	FSR	Регистр адреса при косвенной адресации								xxxx xxxx
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111
87h	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111
89h	TRISE	–	–	–	–	TRISE3	–	–	–	– – – – 1 – – –
8Ah	PCLATH	–	–	–	Старшие 5 битов счетчика команд					– – – 0 0000
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
95h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	1111 1111
96h	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	0000 0000
9Eh	ADRESL	Младший байт результата преобразования АЦП								xxxx xxxx
9Fh	ADCON1	ADFM	–	VCFG1	VCFG0	–	–	–	–	0 – 00 – – – –
Банк 2										
100h	INDF	Регистр данных при косвенной адресации								xxxx xxxx
101h	TMR0	Регистр модуля таймера Timer 0								xxxx xxxx
102h	PCL	Младший байт счётчика команд (PC)								0000 0000
103h	STATUS	IRP	RP1	RP0	nTO	nPD	Z	DC	C	0001 1xxx
104h	FSR	Регистр адреса при косвенной адресации								xxxx xxxx
10Ah	PCLATH	–	–	–	Старшие 5 битов счетчика команд					– – – 0 0000
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
Банк 3										
180h	INDF	Регистр данных при косвенной адресации								xxxx xxxx
181h	OPTION_REG	nRBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
182h	PCL	Младший байт счётчика команд (PC)								0000 0000
183h	STATUS	IRP	RP1	RP0	nTO	nPD	Z	DC	C	0001 1xxx
184h	FSR	Регистр адреса при косвенной адресации								xxxx xxxx
188h	ANSEL	–	–	–	ANS4	ANS3	ANS2	ANS1	ANS0	– – – 1 1111
189h	ANSELH	–	–	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	– – 11 1111
18Ah	PCLATH	–	–	–	Старшие 5 битов счетчика команд					– – – 0 0000
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x

Таблица 1.4

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

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

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

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

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

Таблица 1.5

Мнемоника команды	Описание	Кол-во циклов	Изм. флаги	Прим.
Байт-ориентированные команды				
ADDWF f,d	Сложение W и f	1	C, DC, Z	1,2
ANDWF f,d	Побитное 'И' W и f	1	Z	1,2
CLRF f	Очистить f	1	Z	2
CLRW	Очистить W	1	Z	
COMF f,d	Инвертировать f	1	Z	1,2
DECF f,d	Вычесть 1 из f	1	Z	1,2
DECFSZ f,d	Вычесть 1 из f и пропустить следующую команду, если результат 0	1(2)		1,2,3
INCF f,d	Прибавить 1 к f	1	Z	1,2
INCFSZ f,d	Прибавить 1 к f и пропустить следующую команду, если результат 0	1(2)		1,2,3
IORWF f,d	Побитное 'ИЛИ' W и f	1	Z	1,2
MOVF f,d	Переслать f	1	Z	1,2
MOVWF f	Переслать W в f	1		
NOP	Нет операции	1		
RLF f,d	Циклический сдвиг f влево через перенос	1	C	1,2
RRF f,d	Циклический сдвиг f вправо через перенос	1	C	1,2
SUBWF f,d	Вычесть W из f	1	C, DC, Z	1,2
SWAPF f,d	Поменять местами полубайты в регистре f	1		1,2
XORWF f,d	Побитное 'исключающее ИЛИ' W и f	1	Z	1,2
Бит-ориентированные команды				
BCF f,b	Очистить бит b в регистре f	1		1,2
BSF f,b	Установить бит b в регистре f	1		1,2
BTFSC f,b	Проверить бит b в регистре f, пропустить следующую команду, если результат 0	1(2)		3
BTFSS f,b	Проверить бит b в регистре f, пропустить следующую команду, если результат 1	1(2)		3
Команды управления и операций с константами				
ADDLW k	Сложить константу с W	1	C, DC, Z	
ANDLW k	Побитное 'И' константы и W	1	Z	
CALL k	Вызов подпрограммы	2		
CLRWDT	Очистить WDT	1	-TO.-PD	
GOTO k	Безусловный переход	2		
IORLW k	Побитное 'ИЛИ' константы и W	1	Z	
MOVLW k	Переслать константу в W	1		
RETFIE	Возврат из подпрограммы с разрешением прерываний	2	GIE	
RETLW k	Возврат из подпрограммы с загрузкой константы k в W	2		
RETURN	Возврат из подпрограммы	2		
SLEEP	Перейти в режим SLEEP	1	-TO, -PD	
SUBLW k	Вычесть W из константы k	1	C, DC, Z	
XORLW k	Побитное 'исключающее ИЛИ' константы и W	1	Z	

Примечания:

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

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

1.3.4. Регистр состояния STATUS

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

Структура регистра отражена в табл. 1.6. В регистре STATUS содержатся флаги состояния АЛУ, флаги причины сброса микроконтроллера и биты управления банками памяти данных.

Регистр STATUS может быть адресован любой командой, как и любой другой регистр памяти данных. Если обращение к регистру STATUS выполняется командой, которая воздействует на флаги Z, DC и C, то изменение этих трех битов командой заблокировано. Эти биты сбрасываются или устанавливаются согласно логике ядра микроконтроллера. Команды изменения регистра STATUS также не воздействуют на биты -TO и -PD. Поэтому результат выполнения команды с регистром STATUS может отличаться от ожидаемого. Например, команда CLRF STATUS сбросит три старших бита и установит бит Z (состояние регистра STATUS после выполнения команды *000uu1uu*, где *u* – неизменяемый бит).

Таблица 1.6

Номер бита	Имя бита	Доступ и состояние после сброса	Назначение
1	2	3	4
7	IRP	R/W-0	Выбор банка при косвенной адресации: 1 банк 2, 3(100h – 1FFh) 0 банк 0, 1 (000h – 0FFh)
6, 5	RP1, RP0	R/W-0	Выбор банка при непосредственной адресации: 11 банк 3 (180h – 1FFh) 10 банк 2 (100h – 17Fh) 01 банк 1 (080h – 0FFh) 00 банк 0 (000h – 07Fh)
4	-TO	R-1	Флаг переполнения сторожевого таймера: 1 после POR или выполнения команд CLRWDT, SLEEP, 0 после переполнения WDT
3	-PD	R-1	Флаг включения питания: 1 после POR или выполнения команды CLRWDT, 0 после выполнения команды SLEEP

Окончание табл. 1.6

1	2	3	4
2	Z	R/W-x	Флаг нулевого результата: 1 нулевой результат выполнения арифметической или логической операции, 0 ненулевой результат выполнения арифметической или логической операции
1	DC	R/W-x	Флаг десятичного переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL), заем имеет инверсное значение: 1 был перенос/заем из младшего полубайта, 0 не было переноса/заема из младшего полубайта
0	C	R/W-x	Флаг переноса/заема (для команд ADDWF, ADDWL, SUBWF, SUBWL): 1 был перенос/заем из старшего бита, 0 не было переноса/заема из старшего бита Заем имеет инверсное значение Вычитание выполняется путем прибавления дополнительного кода второго операнда При выполнении команд сдвига (RRF, RLF) бит C загружается старшим или младшим битом сдвигаемого регистра

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

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

1.3.5. Регистр OPTION_REG

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

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

Регистр OPTION_REG доступен по адресам 81h и 181h.

Структура регистра отражена в табл. 1.7.

Таблица 1.7

<table>
<tr><th>Номер бита</th><th>Имя бита</th><th>Доступ и состояние после сброса</th><th>Назначение</th></tr>
<tr><td>7</td><td>-RBPU</td><td>R/W-1</td><td>Включение подтягивающих резисторов на входах PORTB:
1 подтягивающие резисторы отключены,
0 подтягивающие резисторы включены</td></tr>
<tr><td>6</td><td>INTEDG</td><td>R/W-1</td><td>Выбор активного фронта сигнала на входе внешнего прерывания INT:
1 прерывания по фронту сигнала,
0 прерывания по срезу сигнала</td></tr>
<tr><td>5</td><td>T0CS</td><td>R/W-1</td><td>Выбор тактового сигнала для TMR0:
1 внешний тактовый сигнал с вывода RA4/T0CKI,
0 внутренний тактовый сигнал CLKOUT</td></tr>
<tr><td>4</td><td>T0SE</td><td>R/W-1</td><td>Выбор фронта приращения TMR0 при внешнем тактовом сигнале:
1 приращение по срезу сигнала (с высокого к низкому уровню) на выводе RA4/TOCKI,
0 приращение по фронту сигнала (с низкого к высокому уровню) на выводе RA4/TOCKI</td></tr>
<tr><td>3</td><td>PSA</td><td>R/W-1</td><td>Выбор включения предделителя:
1 предделитель включен перед WDT,
0 предделитель включен перед TMR0</td></tr>
<tr><td>2
1
0</td><td>PS2
PS1
PS0</td><td>R/W-1</td><td>Установка коэффициента деления предделителя:
<table>
<tr><th>PS2, PS1, PS0</th><th>для TMR0</th><th>для WDT</th></tr>
<tr><td>000</td><td>1:2</td><td>1:1</td></tr>
<tr><td>001</td><td>1:4</td><td>1:2</td></tr>
<tr><td>010</td><td>1:8</td><td>1:4</td></tr>
<tr><td>011</td><td>1:16</td><td>1:8</td></tr>
<tr><td>100</td><td>1:32</td><td>1:16</td></tr>
<tr><td>101</td><td>1:64</td><td>1:32</td></tr>
<tr><td>110</td><td>1:128</td><td>1:64</td></tr>
<tr><td>111</td><td>1:256</td><td>1:128</td></tr>
</table></td></tr>
</table>

1.3.6. Регистр INTCON

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

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

Структура регистра отражена в табл. 1.8.

Таблица 1.8

Номер бита	Имя бита	Доступ и состояние после сброса	Назначение
7	GIE	R/W-0	Глобальное разрешение прерываний: 1 разрешены все немаскированные прерывания, 0 все прерывания запрещены
6	PEIE	R/W-0	Разрешение прерываний от периферийных модулей: 1 разрешены все немаскированные прерывания периферийных модулей, 0 прерывания от периферийных модулей запрещены
5	T0IE	R/W-0	Разрешение прерывания по переполнению TMR0: 1 прерывание разрешено, 0 прерывание запрещено
4	INTE	R/W-0	Разрешение внешнего прерывания INT: 1 прерывание разрешено, 0 прерывание запрещено
3	RBIE	R/W-0	Разрешение прерывания по изменению сигнала на входах RB7:RB4 PORTB: 1 прерывание разрешено, 0 прерывание запрещено
2	T0IF	R/W-0	Флаг прерывания по переполнению TMR0: 1 произошло переполнение TMR0 (сбрасывается программно), 0 переполнения TMR0 не было
1	INTF	R/W-0	Флаг внешнего прерывания INT: 1 выполнено условие внешнего прерывания на выводе RB0/INT (сбрасывается программно), 0 внешнего прерывания не было
0	RBIF	R/W-x	Флаг прерывания по изменению уровня сигнала на входах RB7:RB4 PORTB: 1 зафиксировано изменение уровня сигнала на одном из входов (сбрасывается программно), 0 не было изменения уровня сигнала

1.3.7. Косвенная адресация данных

Для выполнения косвенной адресации необходимо обратиться к физически не реализованному регистру INDF. Обращение к регистру INDF фактически вызовет действие с регистром, адрес которого указан в FSR. Косвенное чтение регистра INDF (FSR = 0) даст результат 00h. Косвенная запись в

регистр INDF вызывает только воздействия на флаги АЛУ в регистре STATUS. Девятый бит косвенного адреса IRP сохраняется в регистре STATUS<7>. Пример девятиразрядной косвенной адресации показан на рис. 1.10.

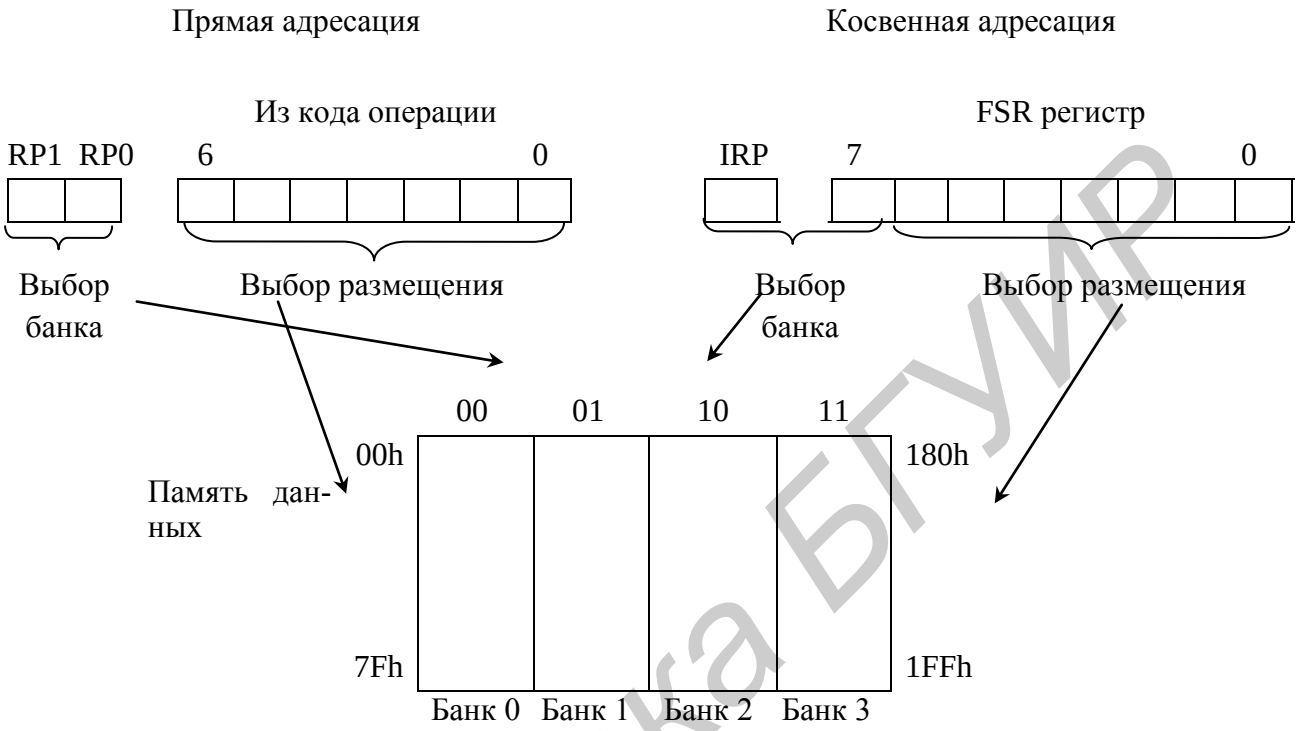


Рис. 1.10. Схема прямой и косвенной адресаций в PIC16F886

В примере ниже показано использование косвенной адресации для очистки памяти данных в диапазоне адресов 20h – 2Fh.

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

1.3.8. Прерывания

PIC16F886 имеет 13 источников прерываний. Регистр INTCON содержит флаги отдельных прерываний, биты разрешения этих прерываний и бит глобального разрешения прерываний.

Если бит GIE (INTCON<7>) установлен в '1', то разрешены все немаскированные прерывания. Если GIE = 0, то все прерывания запрещены. Каждое

прерывание в отдельности может быть разрешено или запрещено установкой/сбросом соответствующего бита в регистрах INTCON и PIE1. При сбросе микроконтроллера бит GIE сбрасывается в '0'.

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

В регистре INTCON находятся флаги следующих прерываний: внешнего сигнала INT, изменения уровня сигнала на входах RB7:RB0, переполнения TMR0.

В регистре INTCON находится бит разрешения прерываний от периферийных модулей. Упрощенная структурная схема логики прерываний (без прерываний от периферийных модулей) показана на рис. 1.11.

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

Для внешних источников прерываний (сигнал INT, изменения уровня сигнала на входах RB7:RB0) время перехода на подпрограмму обработки прерываний будет составлять 3 – 4 машинных цикла. Точное время перехода зависит от конкретного случая, оно одинаково для одного и двух цикловых команд. Флаги прерываний устанавливаются независимо от состояния соответствующих битов маски и бита GIE.

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

Внешнее прерывание с входа RB0/INT происходит:

– по переднему фронту сигнала, если бит INTEDG (OPTION_REG<6>) установлен в '1';

– по заднему фронту сигнала, если бит INTEDG сброшен в '0'.

Когда активный фронт сигнала появляется на входе RB0/INT, бит INTF (INTCON<1>) устанавливается в '1'. Прерывание может быть запрещено сбросом бита INTE (INTCON<4>) в '0'. Флаг прерывания INTF должен быть сброшен программно в подпрограмме обработки прерываний. Прерывание INT может вывести микроконтроллер из режима SLEEP, если бит INTE = 1 до перехода в режим SLEEP. Состояние бита GIE определяет, переходить ли на подпрограмму обработки прерываний после выхода из режима SLEEP.

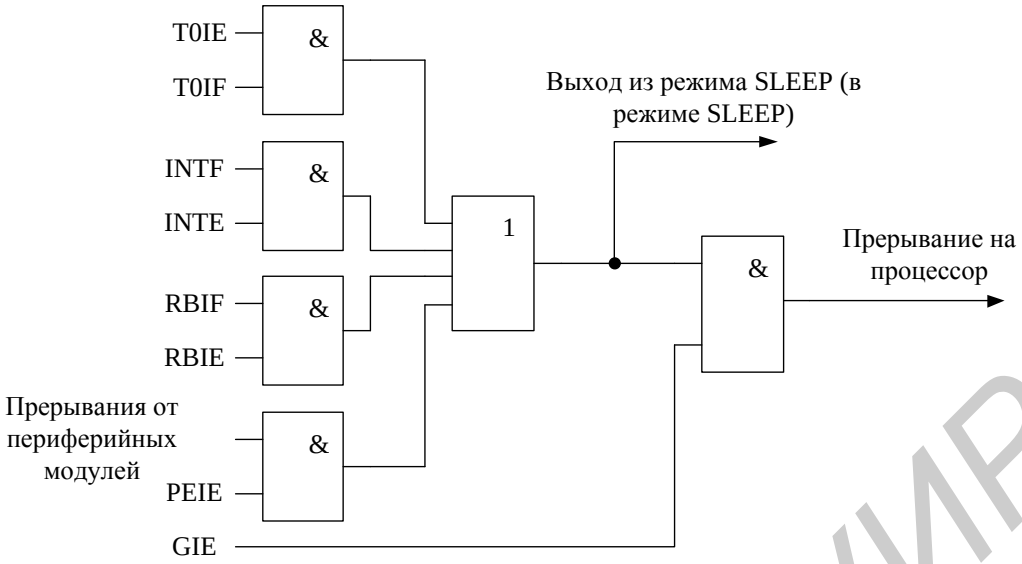


Рис. 1.11. Логика прерываний

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

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

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

Так как старшие 16 байтов каждого банка микроконтроллера PIC16F886 доступны во всех банках, то регистры STATUS_TEMP, PCLATH_TEMP и W_TEMP могут быть размещены в этой области. Ниже показан пример текста программы сохранения контекста.

Пример: Сохранение и восстановление регистров STATUS, W и PCLATH

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

```
MOVWF    PCLATH          ; Восстановить регистр PCLATH
SWAPF    STATUS_TEMP,W   ; Прочитать регистр STATUS_TEMP
                         ; в W, восстанавливая банк памяти программ
MOVWF    STATUS          ; Переписать W в регистр STATUS
SWAPF    W_TEMP,F        ; Поменять местами полубайты в W_TEMP
SWAPF    W_TEMP,W        ; Поменять местами полубайты в W_TEMP и
                         ; записать в W
```

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

1.3.9. Порты ввода/вывода

Микроконтроллеры PIC16F886 имеют четыре порта ввода/вывода: PORTA, PORTB, PORTC и PORTE. Некоторые выводы портов мультиплексированы с периферийными модулями микроконтроллера. Когда периферийный модуль включен, вывод не может использоваться как универсальный канал ввода/вывода.

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

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

Описание выводов регистра PORTA приведено в табл. 1.1.

Пример: Программы инициализации (настройки) PORTA.

```
CLRF     PORTA      ;Очистка (сброс в «0») выходных защелок PORTA
BANKSEL  ANSEL      ; Выбрать банк памяти с регистром ANSEL
CLRF     ANSEL      ; Очистить регистр ANSEL, т. е. задать
                    ; цифровой режим PORTA
BSF      STATUS,RP0 ; Выбор банка 1
MOVLW    0x 0F      ; Значение константы для выбора режимов
                    ; работы разрядов
MOVWF    TRISA      ; Задать RA<3:0> как входы и RA<7:4>  как выходы.
BCF      STATUS,RP0 ; Выбор банка 0
```

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

Имеется возможность прерывания по изменению состояния всех разрядов PORTB (выводы RB7…RB0). Прерывание может возникнуть только от тех разрядов PORTB<7:0>, которые запрограммированы как входы; выходы не включаются в процедуру сравнения. Текущее состояние разрядов PORTB<7:0>, запрограммированных как входы, сравнивается с состоянием, защелкнутым в регистр PORTB при последнем считывании. При несовпадении возникает прерывание по изменению состояния. Это прерывание может вывести микроконтроллер из режима пониженного энергопотребления SLEEP. Для сброса прерывания в подпрограмме обработки необходимо выполнить следующие действия:

1) считать PORTB (это сбросит условие несовпадения);

2) сбросить флаг RBIF.

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

Пример: Инициализация (настройка) PORTB.

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

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

Например, команды BCF и BSF считывают содержимое порта целиком, модифицируют один бит и выводят результат обратно. При использовании этих команд с портами, в которых некоторые разряды запрограммированы как входы, а некоторые как выходы, необходима осторожность. Например, команда BSF для пятого бита регистра PORTB сначала считывает все восемь бит, затем выполняется установка пятого бита и новое значение байта целиком записывается в выходную защелку. Если другой бит регистра PORTB

используется в качестве двунаправленного входа/выхода (например бит 0) и в данный момент он определен как вход, входной сигнал на этом выводе будет считан и записан обратно в выходную защелку этого же вывода, затирая ее предыдущее состояние. До тех пор пока этот вывод остается в режиме входа, никаких проблем не возникает.

Однако если позднее линия 0 переключится в режим выхода, ее состояние будет неопределенным. Команда считывания порта считывает состояние вывода, а не выходных регистров. Например, если разряд порта запрограммирован как выход и установлен в '1', но внешняя схема поддерживает низкий уровень на выводе, порт будет считываться как '0'. На вывод, работающий в режиме выхода, не должны подключаться внешние нагрузки по схеме "монтажное И" либо "монтажное ИЛИ". Возникающие при этом большие токи могут повредить кристалл.

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

Пример: Выполнение операции «чтение – модификация – запись» с портом ввода/вывода.

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

1.3.10. Модуль таймера TMR0

Модуль таймера TMR0 имеет следующие особенности:

– восьмиразрядный таймер/счетчик, доступен по чтению и записи;

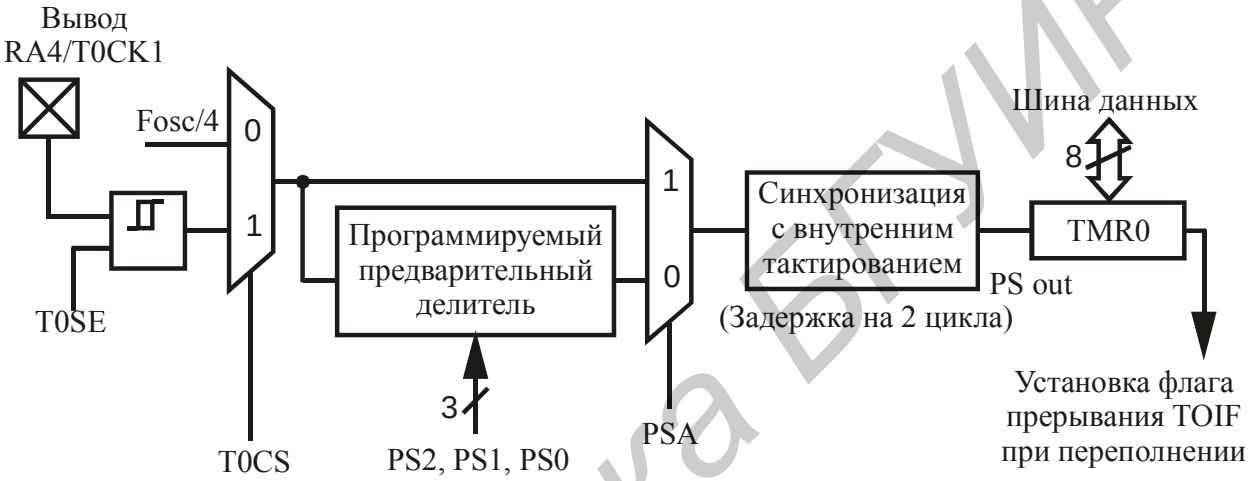
– восьмиразрядный программируемый предварительный делитель;

– внутреннее или внешнее тактирование;

– прерывание по переполнению счетчика (переход от 0FFh к 00h);

– выбор фронта тактирующего импульса при внешнем тактировании.

Упрощенная структурная схема модуля таймера приведена на рис. 1.12.



Примечания:

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

Рис. 1.12. Структурная схема таймера TMR0

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

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

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

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

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

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

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

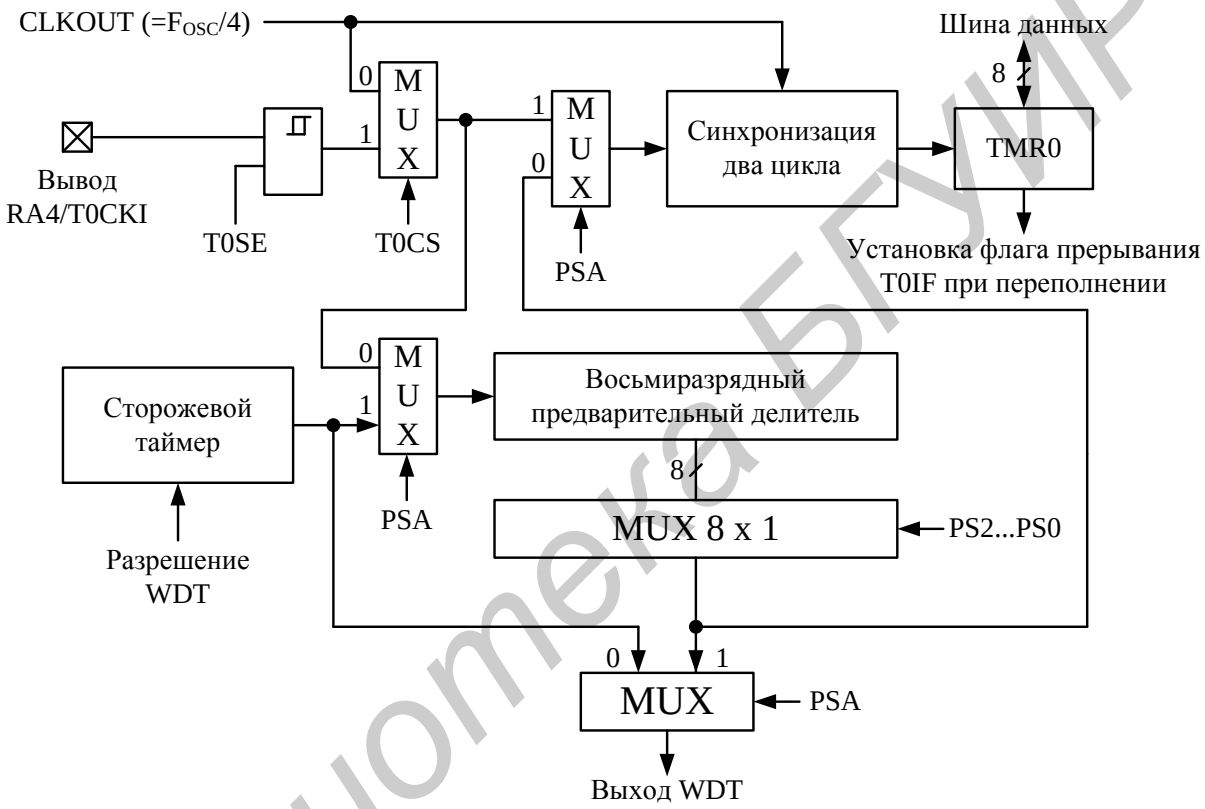
Встроенный восьмиразрядный счетчик может использоваться как предварительный делитель для TMR0 или как дополнительный делитель для сторожевого таймера WDT. Необходимо учесть, что делитель может быть использован либо с TMR0, либо со сторожевым таймером WDT, но не одновременно.

Схема использования предварительного делителя отражена на рис 1.13.

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

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

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



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

Рис. 1.13. Схема использования предварительного делителя

1.4. Основы программирования на Ассемблере

1.4.1. Составление схем алгоритмов

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

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

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

Требование массовости предполагает, что предписание должно обеспечивать выполнение не одной конкретной процедуры, а быть пригодным для реализации класса однородных процедур. Бессмысленно писать программу для получения суммы двух констант, но имеет смысл программа, определяющая сумму двух переменных, которые могут принимать множество значений в некотором диапазоне.

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

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

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

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

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

Первые рассмотренные выше четыре этапа относятся к тому, что принято называть «искусством разработки». Здесь успех в основном определяет-

ся опытом разработчика, его знанием объекта разработки, поэтому дать конкретные рекомендации не представляется возможным. Однако можно детальнее остановиться на составлении схем алгоритмов.

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

Схема алгоритма составляется из отдельных операторов. Различают шесть типов операторов, каждый из которых имеет один или несколько входов или один или несколько выходов (рис. 1.14). Стрелками обозначают направление хода действий.

Оператор в форме прямоугольника (рис. 1.14, *а*) символизирует выполнение каких-либо операций по обработке данных; текст внутри прямоугольника является кратким описанием этого процесса обработки. Например, если в схеме алгоритма содержится оператор «Очистка аккумулятора», то это означает, что на данном этапе работы машины аккумулятор должен быть обнулен. В таком случае для выполнения этой операции микроконтроллеру требуется одна машинная команда. Однако могут быть заданы сложные действия, для которых требуется целый набор команд.

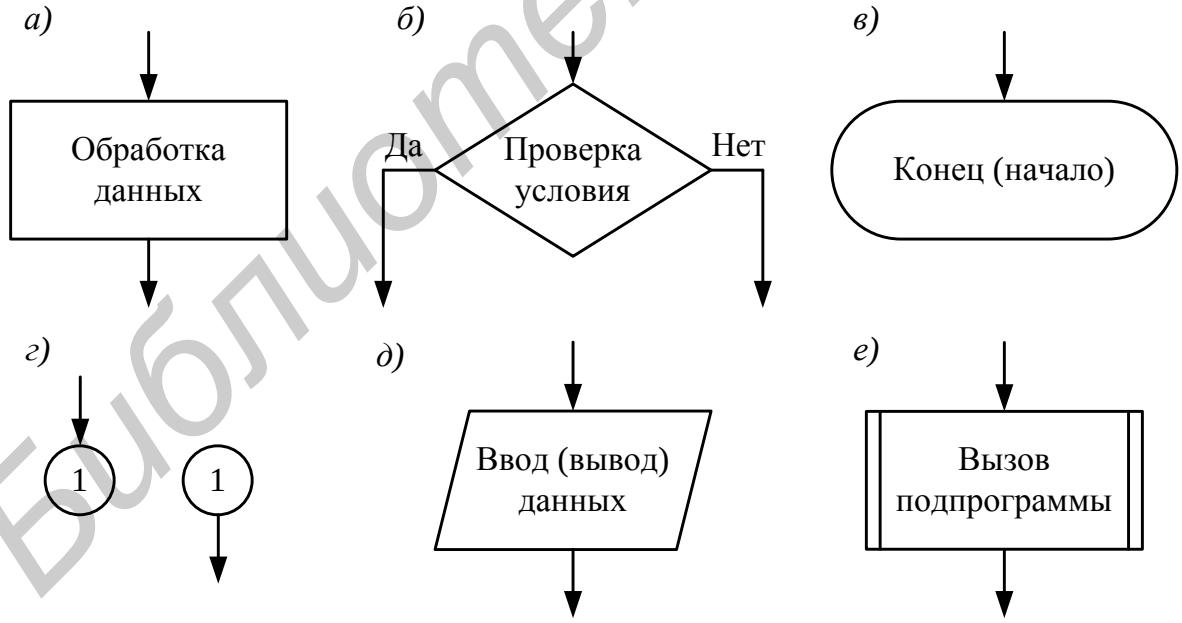


Рис. 1.14. Операторы обработки данных (*а*), проверки условия (*б*), начала или конца алгоритма (*в*), соединения (*г*), вода или вывода данных (*д*), вызова подпрограммы (*е*)

Оператор, имеющий форму ромба (рис. 1.14, *б*), используется для символического обозначения проверки выполнения какого-либо условия с целью принятия решения о направлении последующего хода вычислений. Внутри ромба описывается условие, подлежащее проверке в той точке схемы алгоритма, где размещается данный оператор. Возможные результаты проверки указываются на линиях, выходящих из ромба. Для проведения подобной проверки требуется использование одной или нескольких команд микроконтроллера.

Оператор овальной формы используется для символического обозначения начала или конца алгоритма (рис. 1.14, *в*). Текст внутри овала, как правило, состоит из одного слова – «Начало», «Конец», «Выход», «Возврат» или имени подпрограммы.

В тех случаях, когда необходимо «разорвать» линию потока вычислений, идущую от одного оператора к другому, применяются так называемые соединители в виде окружности с указанной внутри нее цифрой или буквой (рис. 1.14, *г*). Наличие другого идентичного соединителя (с той же цифрой или буквой) означает, что прерванная в месте расположения первого соединителя линия продолжается с того места, где находится второй подобный соединитель. Использование соединителей упрощает внешний вид схемы алгоритма, что позволяет избежать пересечения линий и даёт возможность размещать схему алгоритма на нескольких страницах.

Для обозначения процедур ввода или вывода применяется оператор, имеющий форму параллелограмма (рис. 1.14, *д*). Внутри параллелограмма указывают обычно переменные, подлежащие вводу или выводу.

Оператор, изображенный на рис. 1.14, *е*, используется для обозначения вызова подпрограммы. Внутри него обычно помещается имя вызываемой подпрограммы.

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

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

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

Циклический алгоритм предписывает многократные циклические проходы группы операторов.

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

1.4.2. Общие правила Ассемблера

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

Далее после комментариев (если они есть) и перед первыми командами программы размещаются строки директив. Директивы – это указания Ассемблеру, как именно работать. Список директив Ассемблера довольно обширный, но на данном этапе будем использовать только самые необходимые. Комментарии и директивы не включаются в исполняемый код программы и, следовательно, не попадают в память программ. Строки директив также могут быть в любом месте программы, но некоторые директивы обязательно должны быть заданы до первой строки команд.

Рассмотрим обязательные элементы программы.

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

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

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

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

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

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

В третьем поле задаются операнды. От мнемоники они отделяются одним или более пробелом или символом табуляции. Операндов должно быть

столько, сколько требует формат команды. Между собой операнды разделяются запятой. Если команда предусматривает переменное количество операндов, то они считаются до конца строки или до четвертого поля (начало комментария).

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

Программа обязательно заканчивается директивой END.

1.5. Разработка программ в интегрированной среде MPLAB IDE

1.5.1. Начало работы с MPLAB IDE

Интегрированная среда разработки MPLAB IDE предназначена для написания и отладки программ однокристальных микроконтроллеров PICmicro, производимых компанией Microchip.

MPLAB IDE поддерживает следующие функции:

– создание и редактирование исходных текстов программы;

– сборка (link) и компилирование (compile) исходных текстов программ различными компиляторами;

– отладка кода программы с использованием симулятора или эмулятора (требуется аппаратная часть – debugger).

– MPLAB IDE состоит из нескольких модулей, обеспечивающих единую среду разработки:

– менеджер проекта MPLAB – используется для создания и работы с файлами, относящимися к проекту;

– редактор MPLAB – предназначен для написания и редактирования исходного текста программы, шаблонов и файлов сценария линкера;

– симулятор MPLAB-SIM – программный симулятор моделирует выполнение программы в микроконтроллере с учетом состояния портов ввода/вывода;

– ассемблер MPASM – компилирует исходный текст программы. Дополнительная информация по MPASM представлена в приложении и литературе [3].

1.5.2. Запуск среды проектирования

Запустить MPLAB IDE можно через меню программ (*Пуск ->Программы->Microchip->MPLAB IDE v8.xx->MPLAB IDE*) или через пиктограмму «  » на рабочем столе.

После запуска отобразится основное ***окно программы MPLAB IDE*** (рис. 1.15).

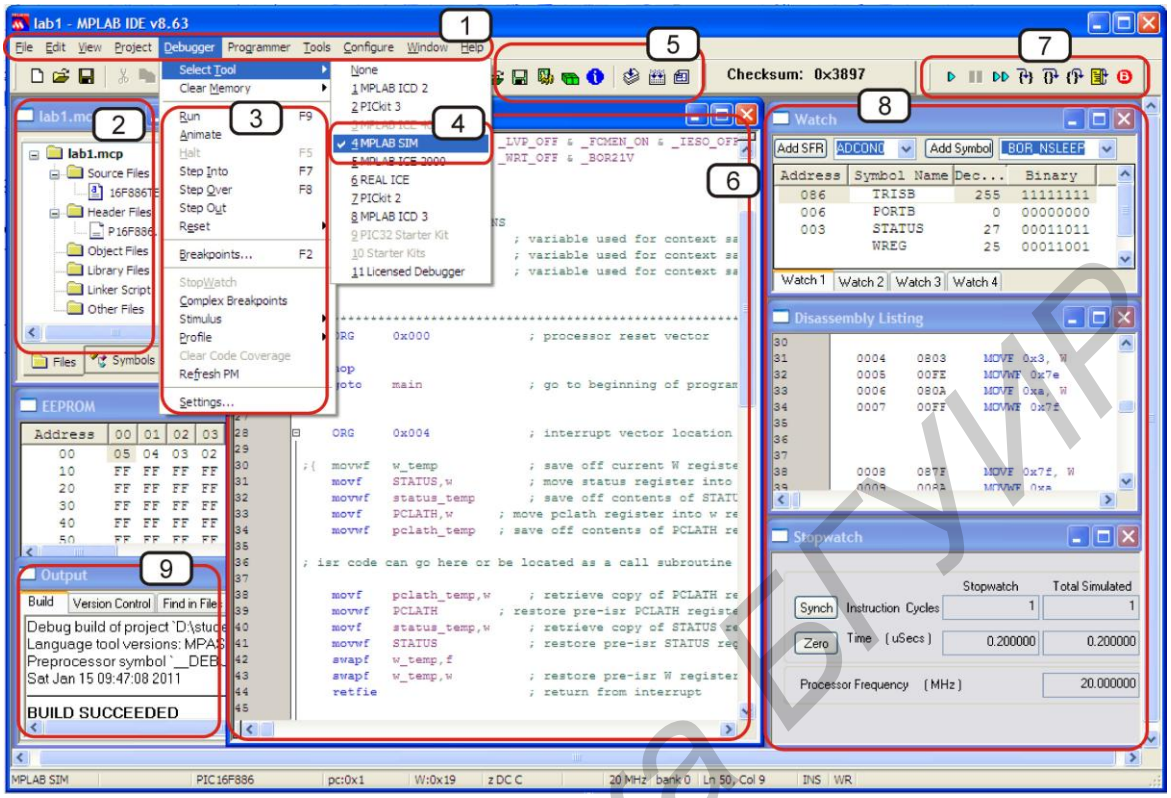


Рис. 1.15. Основное окно программы MPLAB IDE

Рабочий стол среды содержит открытые окна с файлами, диалогами или другой информацией.

– Область программы 1 – это основное меню программы, с которого осуществляется доступ ко всем настройкам *MPLAB IDE.*

– Окно 2 «Project» (*View-> Project*) отображает структуру проекта, т. е. все участвующие при создании проекта файлы.

– Область меню 3, 4, 7 предназначена для включения симулятора и его управления, а окна 8 отображают данные, необходимые на стадии отладки программного кода.

– В Области 5 осуществляется создание нового проекта и компиляция исходного кода.

– Окно 9 «Output» (*View-> Output*) – здесь отображаются все данные о процессе компиляции проекта.

1.5.3. Создание нового проекта

Для того чтобы начать работу, нужно создать новый или открыть существующий проект. Проект – это совокупность файлов, которые используются при создании исполняемого кода, выполняемого микроконтроллером, а

также все индивидуальные настройки, такие как положения и размеры окон. Если создается новый проект, то необходимо сначала подготовить рабочий каталог и файлы. Для этого необходимо создать каталог **Lb1** (**D:\student\CIMPU\041202\Lb1**) и скопировать в него два файла из C:\Program Files\Microchip\MPASM Suite**P16F886.INC** и C:\Program Files\Microchip\ MPASM Suite\Template\Code\ **16F886TEMP.ASM**. Данные файлы содержат исходные коды, которые помогут начать любой проект. **Необходимо обратить внимание на то, чтобы рабочий каталог и путь к нему не содержал русские символы!**

Для создания проекта можно использовать мастер проектов *Project->Project Wizard…* либо меню *«Project->New…»*. Мастер проектов предназначен для быстрого создания нового проекта.

При работе мастера проектов (*Project->Project Wizard…*) появится шесть окон, показанных на рис. 1.16. После нажатия кнопки «Далее» необходимо в появившемся окне 2 выбрать из раскрывающегося списка модель микроконтроллера, которая будет использоваться в проекте, например PIC16F886, и нажать «Далее». В следующем окне 3 необходимо выбрать язык программирования. Из выпадающего меню *«Active Toolsuite»* выбираем «Microchip MPASM Toolsuite» и нажимаем «Далее».

На следующем шаге 4 вводится название проекта и созданная рабочая директория (**D:\student\CIMPU\041202\Lb1\lb1**). Нажмите «Далее». В следующем окне 5 необходимо определить исходные файлы, которые будут использоваться в проекте (кнопка Add). После того как все необходимые файлы добавлены, нажимаем «Далее». Последнее окно 6 «Summary» отображает сведения о том, что было выбрано на предыдущих шагах. Нажимаем «Готово».

В окне «Project» (см. рис. 1.16, 2) появился файл 16F886TEMP.ASM, который автоматически будет включен в группу «Source files» (исходные файлы). Если два раза щелкнуть мышью по этой записи, то файл 16F886TEMP.ASM откроется в окне редактора. Дополним программу после **main**:

```
      ; Учебная программа №1
      list        p=16f886          ; Директива Ассемблеру с указанием типа МК
      #include    <p16f886.inc>     ; Библиотека определений регистров МК
main                                ; Начало основной программы
      banksel     ANSEL             ; Переключение банка памяти
      clrf        ANSEL             ; Очистка регистров
      clrf        ANSELH            ;
      banksel     TRISB             ; Переключение банка памяти
```

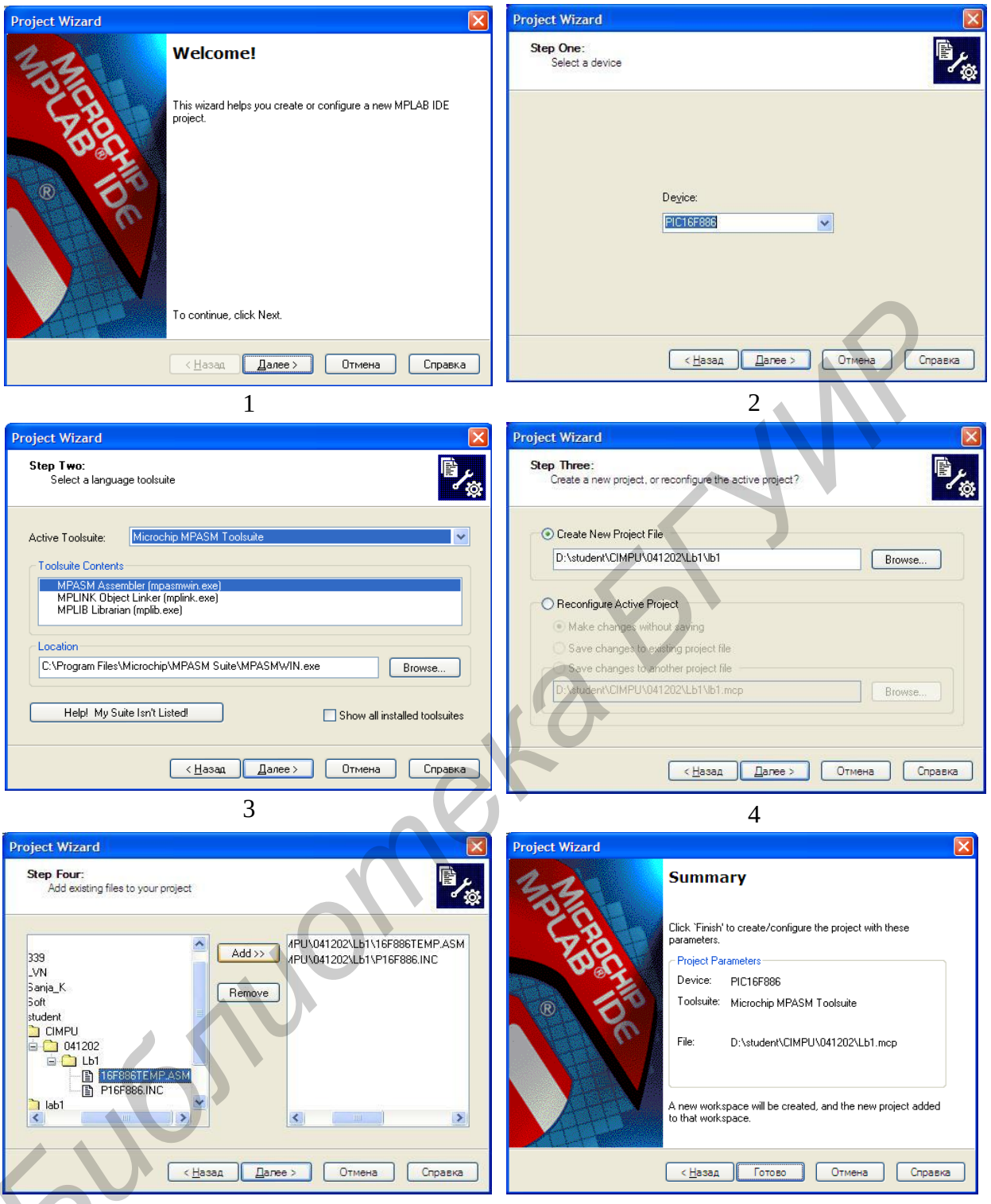


Рис. 1.16. Окна мастера проектов

```
        bcf     TRISB,0          ; Линия RB0 на вывод данных
        bsf     TRISC,0          ; Линия RC0 на ввод данных
        bcf     OPTION_REG,NOT_RBPU
                                 ; Включение подтягивающих резисторов
        banksel PORTC            ; Переключение банка памяти
wait    btfsc   PORTC,0          ; Проверка состояния линии RC0
```

```
        goto    wait
        bsf     PORTB,0
        call    Delay           ; Вызов подпрограммы задержки
        bcf     PORTB,0
        call    Delay
        goto    wait
Delay                           ; Подпрограмма задержки
        nop
        nop
        nop
        return
        end
```

Теперь можно скомпилировать исходную программу. Нажимаем на пиктограмму «  » на панели инструментов.

При успешной компиляции в ***окне «Output»*** отобразится следующая информация:

«BUILD SUCCEEDED» – обозначает, что компиляция прошла успешно. «BUILD FAILED» – исходный код содержит ошибки.

1.5.4. Использование симулятора в среде MPLAB IDE

После написания исходного текста программы и компиляции проекта в MPLAP IDE необходимо проверить, насколько правильно работает программа и насколько верен её алгоритм. Как правило, с первого раза новая программа может работать неправильно; также следует учитывать, что память микроконтроллера изнашивается при каждой операции программирования, поэтому удобно отлаживать исходный текст с помощью компьютерного симулятора MPLAB SIM.

Данный симулятор моделирует выполнение программы любого типа PICmicro с учетом состояния портов ввода/вывода на скорости, которая зависит от быстродействия персонального компьютера и сложности кода программы.

Симулятор поддерживает следующие функции:

– эмуляция памяти программ;

– прерывание выполнения программы в точках остановки;

– работа по шагам;

– контроль регистров общего и специального назначения;

– измерение времени выполнения кода программы.

Все указанные функции используются с проекта MPLAB IDE. Символьные метки могут использоваться для указания точек остановки и трассировки, а также изменения значений регистров памяти данных.

В симуляторе предусмотрена автоматическая работа по шагам. Программа выполняется непрерывно с обновлением всех значений в регистрах после исполнения каждой инструкции.

1.5.5. Выбор и настройка симулятора

Чтобы активировать симулятор, необходимо выбрать пункт меню *«Debugger->Select Tool->MPLAB SIM»*. После выбора в меню произойдут следующие изменения на *рабочем столе среды MPLAB IDE*:

1. Появилась надпись «MPLAB SIM» на панели состояния.
2. Появились дополнительные пункты меню в меню *«Debugger»*.
3. Появилась панель отладки на панели инструментов.
4. Добавилась закладка «MPLAB SIM» в окне «Output».

Панель отладки содержит:

– запуск (F9); – пауза (F5); – анимация; – шаг в (F7); – шаг через (F8); – шаг из; – сброс (F6).

MPLAB-SIM рассчитывает время работы микроконтроллера исходя из его тактовой частоты, так как от неё зависит, сколько времени займет рабочий цикл контроллера.

В реальной системе частота микроконтроллера задаётся опорным генератором (это может быть кварцевый или керамический резонатор, внешняя или внутренняя RC-цепочка), а в «виртуальном» микроконтроллере частоту можно указать вручную, для чего необходимо выбрать пункт меню «*Debugger->Settings…*». В появившемся ***окне «Simulator settings»*** (рис. 1.17) выберите вкладку «Osc / Trace». В поле «Processor Frequency» введите 10, а в поле «Units» выберите «MHz», таким образом указав частоту в 20 МГц. Для того чтобы все изменения вступили в силу, нужно нажать кнопку «OK».

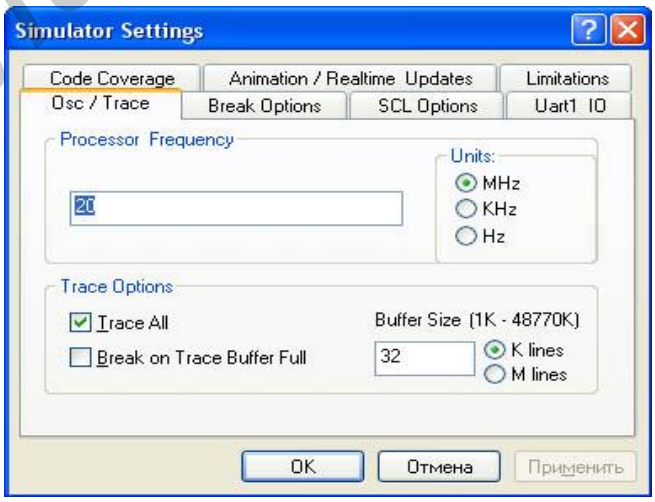


Рис. 1.17. Окно настройки симулятора

1.5.6. Симуляция выполнения программы

Рассмотрим симуляцию выполнения программы на примере циклической программы задержки. Для этого необходимо дописать код программы, которая выполняет увеличение на единицу значение регистра COUNT с интервалом, равным задержке выполнения подпрограммы «Delay». Далее содержимое регистра COUNT сохраняется в регистре PORTB.

```
        List    p=16f886                ; Директива Ассемблеру с указанием типа МК
        #include    <p16f886.inc>       ; Библиотека определений регистров МК
            ; Необходимые  переменные:
            COUNT   EQU     0x20        ; Значение счетчика
            DVAR    EQU     0x21        ; Переменная задержки
            DVAR2   EQU     0x22        ; Переменная задержки
            ; После Main вставить текст программы
Main        ; Начало основной программы
            banksel ANSEL               ; Переключение банка памяти
            clrf    ANSEL               ; Очистка регистров
            clrf    ANSELH
            banksel TRISB               ; Переключение банка памяти     «Bank1»
            movlw   B'00000000'
            movwf   TRISB               ; Настраиваем весь PORTB на выход
            banksel COUNT               ; Переключение банка памяти «Bank0»
Init
            clrf    COUNT               ; Инициализация счетчика
IncCount
            incf    COUNT,F             ; Инкремент счетчика
            movf    COUNT,W
            movwf   PORTB               ; Вывод счетчика на PORTB
            call    Delay               ; Вызов подпрограммы задержки
            goto    IncCount ;
Delay       ; Подпрограмма задержки
            movlw   0x40
            movwf   DVAR2               ; Количество внешних циклов
DelayOuter
            movlw   0xFF
            movwf   DVAR                ; Количество внутренних циклов
DelayInner
            decfsz  DVAR,F
            goto    DelayInner
            decfsz  DVAR2,F
            goto    DelayOuter
            return
            end
```

Симулятор позволяет отслеживать все изменения регистров. Для этого:

а) выберите пункт меню *View->Watch.* Появится окно наблюдения за состоянием регистров и переменных (Watch);

б) добавьте регистр COUNT в список наблюдаемых регистров из правого раскрывающегося списка, нажмите кнопку «Add Symbol»;

в) выберите PORTB из левого раскрывающегося списка и нажмите кнопку «Add SFR».

Окно Watch примет вид, представленный на рис. 1.18:

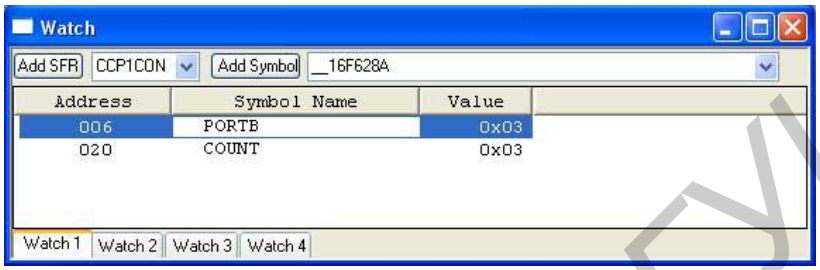


Рис. 1.18. Окно изменений регистров и переменных

Для начала симуляции программы выбираем пункт меню «*Debugger>Reset>Processor Reset»* или «», после этого зеленый указатель «» появится в начале кода программы. Если нажать на иконку «» или выбрать пункт меню «*Debugger>Step Into»,* то произойдет выполнение одной команды (одного шага). Чтобы запустить симуляцию в режиме анимации, нажмите на иконку «» на панели инструментов MPLAB. Остановить симуляцию можно с помощью иконки пауза «».

MPLAB-SIM позволяет рассчитывать время выполнения программы. Пусть требуется узнать время выполнения подпрограммы «Delay». Для этого установите зеленый указатель на команде вызова подпрограммы задержки (щелкните правой кнопкой мыши по команде «call Delay», выберите в открывшемся контекстном меню *Set PC at Cursor*). Далее установите точку остановки на команде «goto IncCount».

1.5.7. Точки остановки

Точки остановки (breakpoint) – это указание симулятору остановиться на выполнении какой-либо определённой команды. Есть несколько способов установить точку остановки:

- Выбрать пункт «Set Breakpoint» из контекстного меню, появляющегося при нажатии правой кнопкой мыши по нужной строке программы.
- Сделать двойной щелчок левой кнопкой мыши по строке программы.

В результате выполнения любого из указанных методов слева от строки программы должна появиться пиктограмма «», как показано на рис. 1.19:

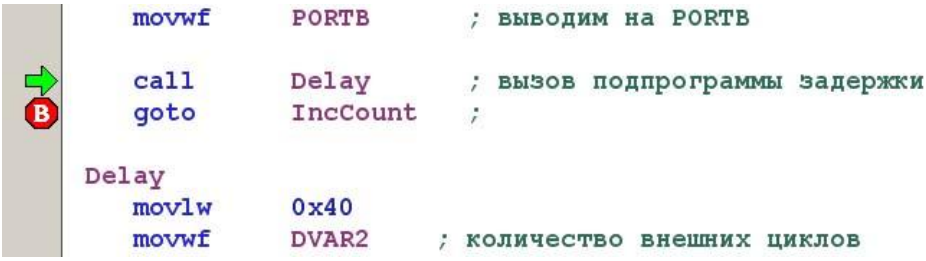


Рис. 1.19. Отображение точки остановки

Убираются точки остановки теми же способами, которыми устанавливаются (правый щелчок мышью по команде Remove Breakpoint или двойной щелчок левой кнопкой мыши по полю слева от команды).

Для измерения временных интервалов в симуляторе MPLAB SIM предусмотрен инструмент StopWatch. Чтобы запустить его, выберите пункт меню «*Debugger->StopWatch*». Появится ***окно StopWatch***, показанное на рис. 1.20.

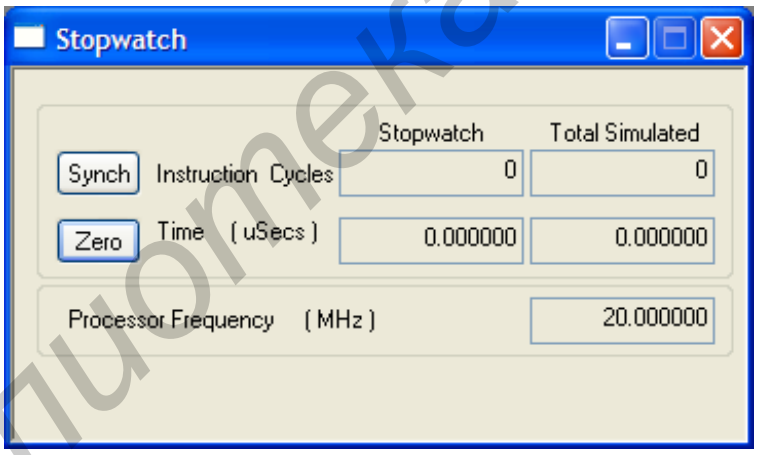


Рис. 1.20. Окно измерения временных интервалов

Запустите программу, выбрав пункт меню «*Debugger>Run*» или «». Когда симулятор остановит выполнение программы в точке остановки, в окне Stopwatch в поле Time отобразится информация о времени, прошедшем за период выполнения, а в поле Instruction Cycles – количество выполненных циклов (в PIC16 один цикл равен четырем тактам задающего генератора). Если всё сделано правильно, в окне «Time» отобразится время 9.844 mSecs (миллисекунд). Кнопка «Zero» сбрасывает результат измерения.

Также в симуляторе есть возможность устанавливать точки остановки по определенному событию или по определенной цепочке событий. Напри-

мер, необходимо, чтобы выполнение программы было остановлено, когда значение счетчика COUNT равнялось 10h. Для этого через пункт меню «*Debugger → Complex Breakpoints*» откройте ***окно «Simulator Complex Breakpoints»*** (рис. 1.21), выберите «Add Breakpoints» и в открывшемся ***окне «Set Breakpoint»*** (рис. 1.22) перейдите на закладку «Data Memory». В поле «Address» из выпадающего списка «Symbol/Hex» выберите регистр «COUNT». В следующем поле из списка «Breakpoint Type» выберите «Read Specific Byte» и задайте значение «Specific Value», равное 10.

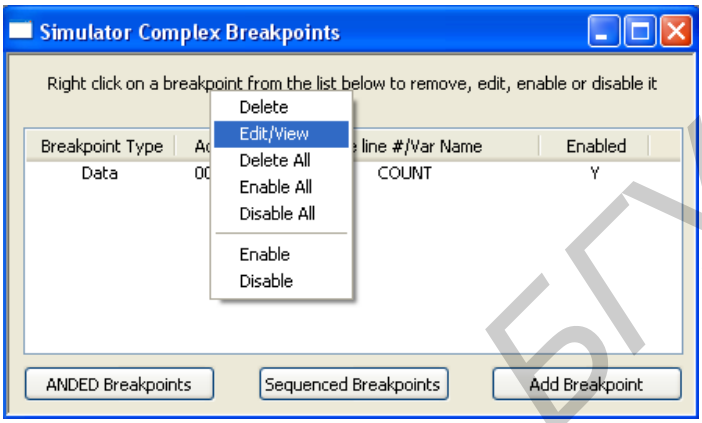


Рис. 1.21. Окно «Simulator Complex Breakpoints»

Теперь симулятор будет прерывать выполнение программы каждый раз, когда значение регистра «COUNT» будет равно 10h. Если необходимо изменить параметры прерывания выполнения программы, ***в окне «Simulator Complex Breakpoints»*** нажмите правой кнопкой мыши по строке «Data…» и из выпадающего списка выберите пункт «Edit», как показано на рис. 1.21. Также с помощью этого меню можно разрешить или запретить использование такого типа точки остановки.

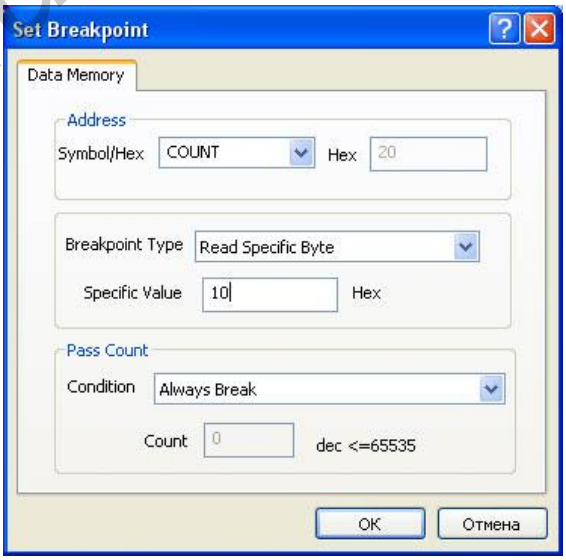


Рис. 1.22. Окно настройки точек остановки по событию

1.5.8. Точки трассировки

Дополнительным средством контроля над ходом выполнения программы являются точки трассировки.

Симулятор MPLAB SIM поддерживает буфер трассировки, в котором сохраняются данные точек трассировки. Допускается запись в буфер при переполнении с удалением более старых значений (если не выбран параметр остановки программы при переполнении буфера).

Чтобы разрешить использование буфера трассировки, необходимо выбрать пункт меню «*Debugger->Settings…*». В окне «*Simulator settings*» выберите закладку «Osc / Trace» (см. рис. 1.17). В поле «Trace Options» поставьте галочку возле параметра «Trace All» и нажмите «OK». После этого в меню *«View->Simulator Trace»* станет доступно ***окно трассировки «Trace»***.

Запустите выполнение программы и остановите. Окно трассировки примет вид, как на рис. 1.23. Оно будет содержать следующую информацию:

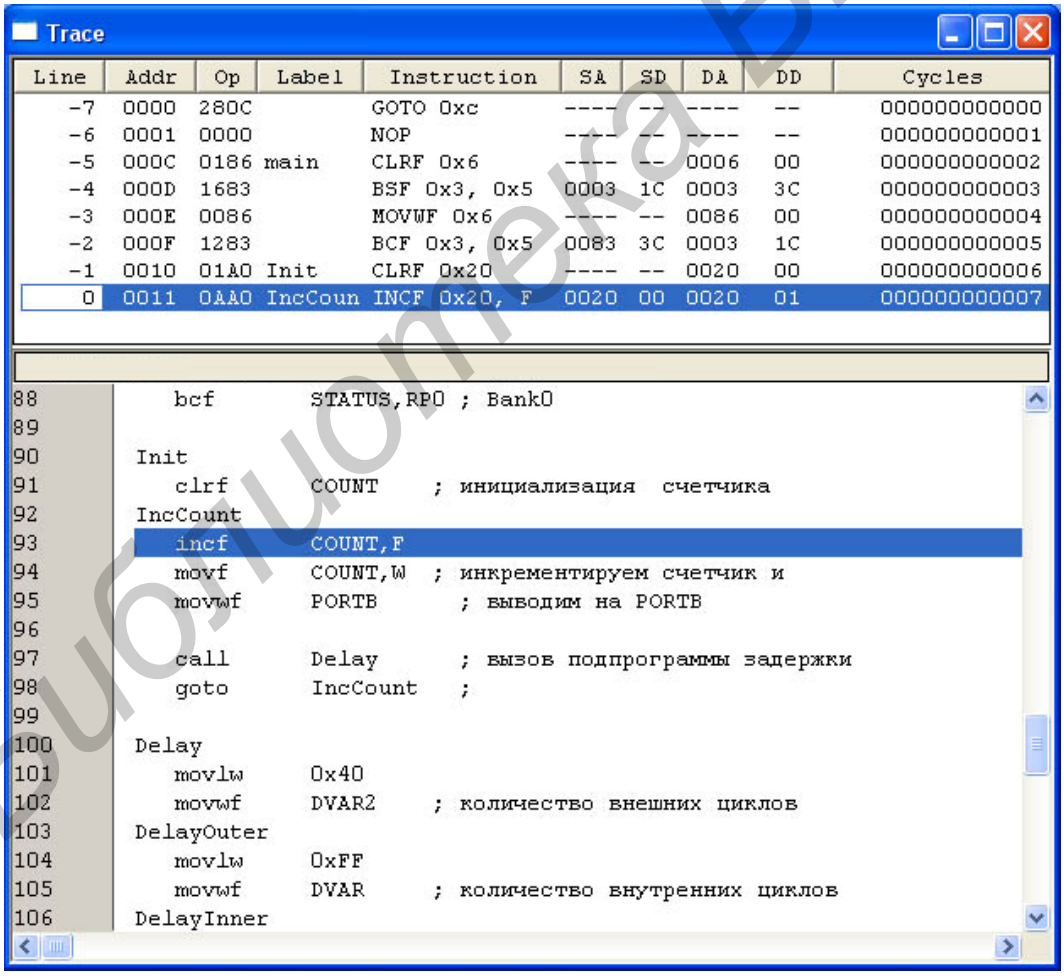


Рис. 1.23. Окно трассировки «Trace»

- **Line** (Line Number) – позиция относительно пункта остановки;

- **Addr** (Address) – адрес в памяти программ;
- **Op** (Opcode) – код команды;
- **Label** (Label) – метка переходов в программе;
- **Instruction** (Instruction) – мнемоника команды;
- **SA** (Source Data Address) – адрес исходных данных;
- **SD** (Source Data Value) – значение исходных данных;
- **DA** (Destination Data Address) – адрес данных назначения;
- **DD** (Destination Data Value) – значение данных назначения;
- **Cycles/Time** (Time Stamp) – интервал времени от сброса в циклах или в секундах.

Использование фильтров «Filter in trace» или «Filter out trace» позволяет добавлять или исключать точки трассировки из буфера трассировки.

Для того чтобы установить фильтр трассировки, необходимо нажать правой кнопкой мыши по нужной строке программы и из контекстного меню выбрать «Filter in trace» или «Filter out trace».

Слева от строки программы появиться пиктограмма «☐» (Filter in trace) или «☒» (Filter out trace).

Примечание. В MPLAB SIM можно использовать или «Filter in trace», или «Filter out trace», так как при выборе одного из параметров второй будет недоступен.

1.5.9. Использование стимулов

Во время симуляции выполнения программы MPLAB SIM позволяет имитировать с помощью так называемых «стимулов» внешние сигналы на портах ввода/вывода, а также изменять данные регистров специального (SFR) и общего (GPR) назначения.

Существуют два основных типа стимулов:

- Асинхронный стимул – непосредственное управление состоянием портов ввода/вывода.
- Синхронный стимул – определенная последовательность изменения сигналов на портах ввода/вывода или последовательное изменение значения регистров SFR и GPR.

Для создания синхронных и асинхронных стимулов откройте ***окно диалога «Stimulus»***, которое представлено на рис. 1.24, с помощью меню *Debugger-> Stimulus ->New Workbook.*

Асинхронный стимул используется для моделирования логического состояния порта ввода/вывода, настроенного на вход (устанавливаются значения +5 В или 0 В). В диалоговом окне асинхронных стимулов «Async» выбирается в поле «Pin/SFR» вывод порта микроконтроллера (МК). В поле «Action» выставляется уровень входного сигнала на выбранном выводе МК.

Нажатие по соответствующей кнопке «Fire» приведет к изменению состояния порта RB1, и в окне «Output» на закладке «MPLAB SIM» (см. рис. 1.15) появится надпись «SIM-N0001 Note: Asynchronous Stimulus Set Low RB1 fired».

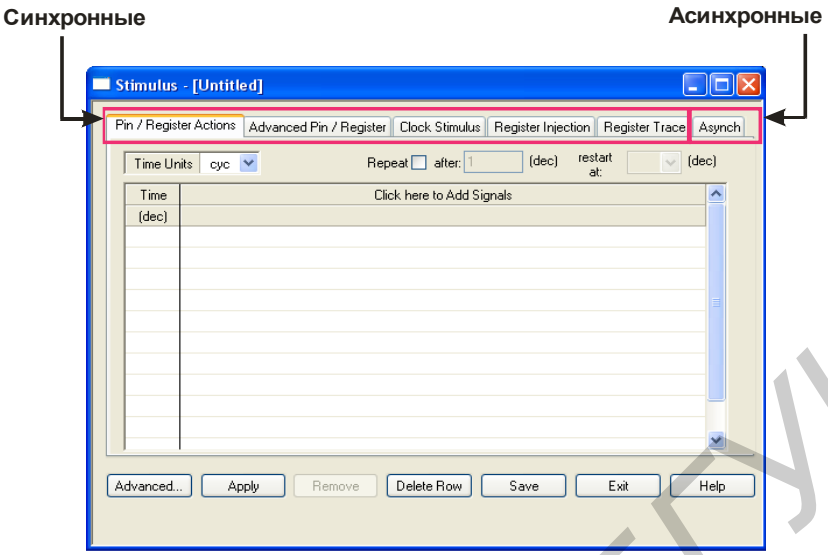


Рис. 1.24. Окно настройки стимулов

Для того чтобы использовать **синхронный стимул,** откройте закладку «Pin/Register Actions» в окне «Stimulus». Выберите из выпадающего списка «Time Units» интервал времени срабатывания стимула в микросекундах «us».

Нажмите левой кнопкой мыши по надписи «Click here to Add Signals», в появившемся окне «Add/Remove Pin/Registers» выберите в поле «Available Signals» необходимый вывод, например, RB1. Далее нажмите кнопку «Add» и «OK». В появившемся столбце «RB1» задаются значение нуля или единицы, которые соответствуют низкому или высокому уровню на порте RB1 микроконтроллера (рис. 1.25).

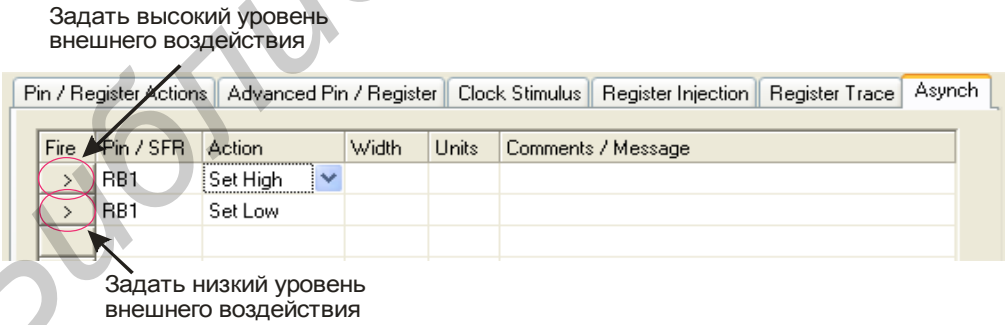


Рис. 1.25. Окно настройки синхронного стимула

В столбце «Time» задаются интервалы времени от момента нажатия кнопки «Apply» в окне «Stimulus», например 100, 200,…, 900, 1000 мкс по ANSI. При выполнении симуляции программы каждый интервал времени будет отображаться в окне «Output» на закладке «MPLAB SIM». Нажмите кнопки «Apply» и «Save», чтобы применить и сохранить параметры воздействия.

На закладке «Advanced Pin/Register» можно задавать изменения сигналов на портах ввода/вывода или изменение значения регистров SFR и GPR по определенному событию. В поле «Define Conditions» задаются условия, в соответствии с которыми будут происходить изменения воздействия (рис. 1.26).

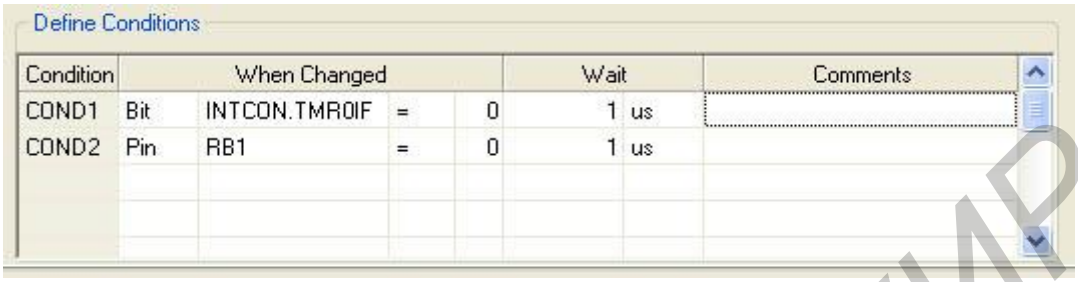


Рис. 1.26. Окно задания условий

Оно содержит:

- «Condition» – имя события;
- «When Changed» – условие события;
- «Wait» – задержка перед реагированием стимула, если условие истинно.

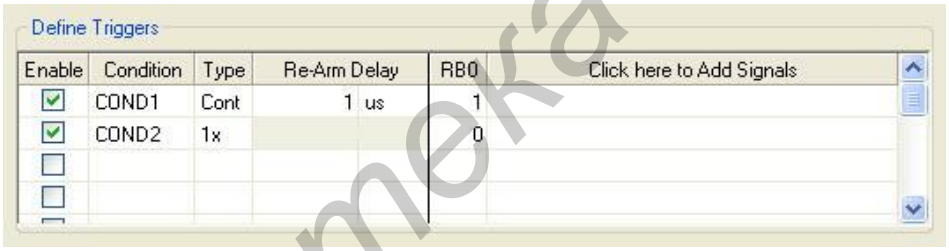


Рис. 1.27. Окно настройки реакции стимула

Поле «Define Triggers» (рис. 1.27) позволяет настраивать реакцию стимула на событие и содержит следующие столбцы:

- «Enable» – разрешение или запрещение реагирования стимула;
- «Condition» – имя события;
- «Tape» – тип воздействия («Count» – многократное, «1x» – однократное);
- «Re-Arm Delay» – задержка до проверки следующего события.

Сохранение и загрузка **файла сценария стимулов** осуществляется при помощи окна «Advanced SCL Operations», которое вызывается при нажатии кнопки «Advanced…» в окне «Stimulus». Создайте файл, нажав по кнопке «Generate SCL File», и сохраните его в директории с программой. Кнопки «Attach» и «Detach» подключают и отключают сохраненный файл сценария.

2. Лабораторная работа «Программирование и отладка процедур опроса бинарных датчиков и управления единичными индикаторами»

2.1. Цель работы

Изучить архитектуру микропроцессорного устройства и приемы работы в интегрированной среде MPLAB на примерах составления и отладки программ ввода и вывода бинарных данных.

2.2. Краткие теоретические сведения

Пусть требуется спроектировать устройство на микроконтроллере (МК) PIC16F886, которое обеспечивает мигание светодиода на время нажатия и удержания кнопки. Другими словами: пока нажата кнопка, светодиод мигает, при отпущенной кнопке светодиод погашен.

Решение. В этой задаче один источник информации и один объект управления. Источник информации представляет собой два нормально разомкнутых контакта SB. При нажатии кнопки контакты замыкаются, при отпускании – размыкаются. Микроконтроллер должен определять состояния кнопки. Таких состояний два: «не нажата» и «нажата». Эти состояния можно условно описать одной булевой переменной x, которая может принимать два значения: 0 и 1. Таким образом, кнопка как источник даёт один бит информации. Для приёма этой информации достаточно задействовать лишь какой-либо один из входов МК. Сама по себе кнопка является электрически пассивным источником, и для определения ее состояния с помощью порта МК необходимо преобразовать состояние контактов в электрический сигнал напряжения. Наиболее просто это можно выполнить по схеме, изображенной на рис. 2.1.

В состоянии «не нажата» по порту МК будет читаться логическая единица, а в состоянии «нажата» – логический нуль. Подключим в проектируемом устройстве кнопку SB к порту RC7.

Объектом управления в устройстве является светодиод. Его состояние «включен» обеспечивается пропусканием тока $I_Д$ силой в несколько миллиампер (обычно 5–10 мА). Поскольку нагрузочная способность портов МК 25 мА, то светодиод можно подключать без дополнительного усилителя непосредственно к порту, предусмотрев лишь токоограничивающий резистор R, как показано на рис. 2.2. Сопротивление резистора можно рассчитать по формуле $R = (E^1 – U_{VD}) / I_Д$, где $E^1 = 5$ В – уровень логической единицы, $U_{VD} = 2{,}4$ В – падение напряжения на открытом светодиоде. Зададимся $I_Д = 5$ мА, тогда $R = (5–2{,}4) / (5 \cdot 10^{-3}) = 2{,}6 / (5 \cdot 10^{-3}) = 2600 / 5 = 520$ Ом.

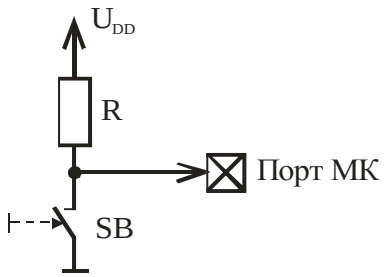


Рис. 2.1. Схема подключения кнопки к порту МК

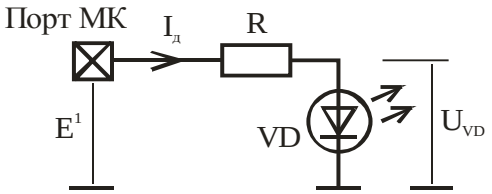


Рис. 2.2. Схема подключения светодиода к порту МК

Выберем для управления светодиодом порт RB0.

Для обеспечения требований к временным параметрам устройства необходимо выбрать частоту и вариант тактирования работы МК. Выберем вариант синхронизации с помощью кварцевого резонатора, имеющего резонансную частоту 10 МГц. Таким образом, длительность машинного цикла (время выполнения большинства команд) составит 0,4 мкс.

Составим схему алгоритма управляющей программы (рис. 2.3).

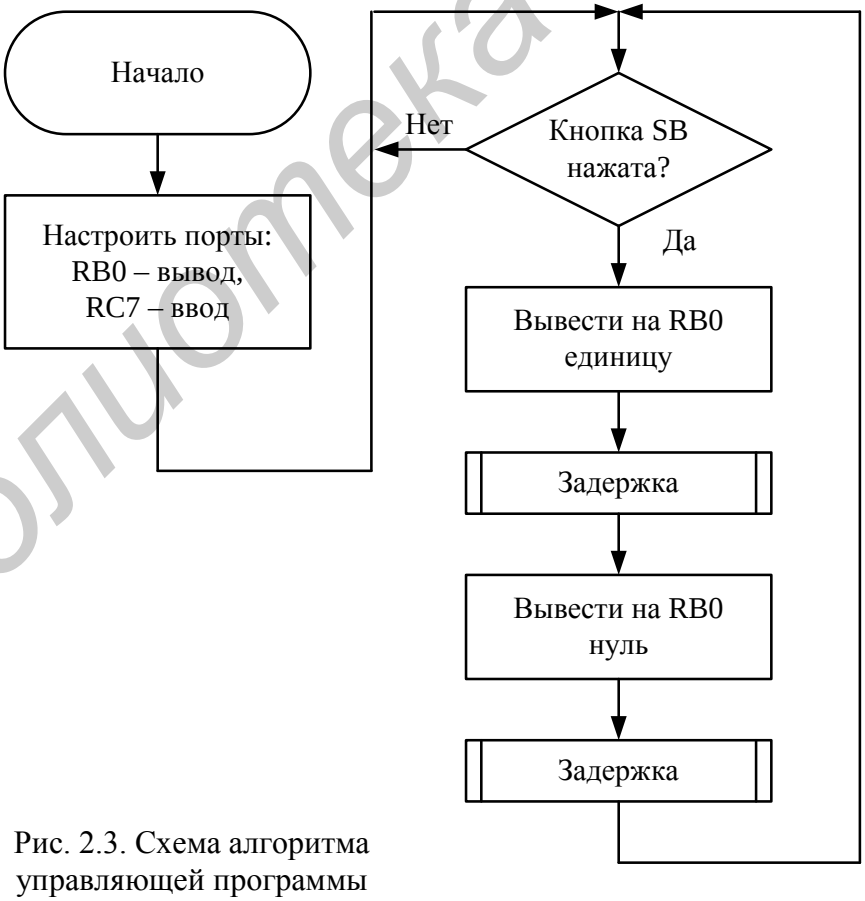


Рис. 2.3. Схема алгоритма управляющей программы

От схемы алгоритма легко перейти к учебной программе №1, приведенной в п. 1.5.3 на с. 39.

2.3. Лабораторное задание

1. Изучить описание лабораторной установки, структурную схему, организацию памяти программы, назначение и структуры регистров STATUS, OPTION_REG и INTCON микроконтроллера PIC16F886. Ознакомиться с системой команд микроконтроллера.

2. Руководствуясь указаниями по работе с MPLab IDE (см. подразд. 1.5) ввести, скомпилировать, устранить ошибки и загрузить в симулятор программу, приведенную в п. 1.5.3.

3. Провести моделирование работы программы в пошаговом режиме. При этом обратить внимание на изменение состояний программного счетчика и вывода RB0 микроконтроллера.

4. Загрузить программу в МК PIC16F886 и проверить ее работу, используя для контроля генерируемых импульсов на выводе RB0 осциллограф.

5. Подготовить отчет по работе, который должен содержать: титульный лист, цель работы, структурную схему ядра микроконтроллера PIC16F886, систему команд, структуры регистров STATUS, OPTION_REG, INTCON, а также листинг отлаженной программы и выводы по работе.

2.4. Контрольные вопросы

1. Поясните назначение памяти программ и памяти данных, а также различия гарвардской и принстонской (фон-неймановской) архитектур в микропроцессорных устройствах.

2. Поясните назначение программного счетчика, стека, регистра команд, декодера команд, арифметико-логического устройства, рабочего регистра W и регистра состояния STATUS.

3. Поясните, как изменяется состояние программного счетчика в процессе выполнения линейного участка программы.

4. Как изменяется состояние программного счетчика в процессе выполнения команды перехода GOTO?

5. Как изменяются состояния программного счетчика, регистра команды и стека в процессе выполнения команд вызова подпрограммы CALL и возврата из подпрограммы RETURN?

6. Поясните назначение и функции регистров OPTION_REG и INTCON.

3. Лабораторная работа «Программирование и отладка процедур арифметических и логических преобразований данных»

3.1. Цель работы

Углубить знания архитектуры микропроцессорных устройств и закрепить практические навыки работы в интегрированной среде MPLAB на примерах составления и отладки программ процедур арифметических и логических преобразований данных.

3.2. Краткие теоретические сведения

В нижеприводимой программе выполняется присвоение двум переменным конкретных значений, выполнение над ними преобразований и вывод результата на порт.

```
;*****************************************************************
;*  Выполнение арифметических и логических операций
;*****************************************************************
        list            p=16f886            ; Выбираем тип контроллера
        #include        <p16f886.inc>       ; Подключаем файл с описанием имён
                                            ; регистров и разрядов используемого
                                            ; микроконтроллера
; Формируем слово конфигурации микроконтроллера
        __CONFIG    _CONFIG1, _LVP_OFF & _FCMEN_ON & _IESO_OFF & _BOR_OFF
& _CPD_OFF & _CP_OFF & _MCLRE_ON & _PWRTE_ON & _WDT_OFF & _HS_OSC
        __CONFIG    _CONFIG2, _WRT_OFF & _BOR21V
;*****************************************************************
;*      Блок объявлений пользовательских переменных и констант                *
_A      equ     0x20    ; Регистр для хранения числа A
_B      equ     0x21    ; Регистр для хранения числа B
;*****************************************************************
;               Начало текста программы

        ORG     0x000               ; Вектор сброса микроконтроллера
        goto    main                ; Перейти на начало основной программы
        ORG     0x004               ; Вектор прерывания
main:                               ; Метка начала основной программы
        banksel         PORTA       ; Директива выбора банка памяти, где
                                    ; расположен указанный регистр
        clrf    PORTA               ; Инициализировать PORTA
        clrf    PORTB               ; Инициализировать PORTB
        clrf    PORTC               ; Инициализировать PORTC
        Banksel         ANSEL       ; Выбрать банк памяти
        clrf    ANSEL               ; Очистить регистры ANSEL и ANSELH,
        clrf            ANSELH      ; установить порты ввода/вывода как цифровые
```

```
        banksel     TRISA       ; Выбрать банк памяти
        bcf         TRISA,5     ; Настроить вывод PORTA RA5 как выход
        clrf        TRISB       ; Настроить все выводы PORTB как выходы
        movlw       b'00010000' ; Загрузить в W константу
        movwf       TRISC       ; Настроить вывод RC4 на вход, остальные на выходы
        banksel     PORTC       ; Выбрать банк памяти

metka:                          ; Метка основного бесконечного цикла программы
        movlw       .100        ; Присвоить переменной A значение 100
        movwf       _A          ;
        movlw       0x24        ; Присвоить переменной B значение 36
        movwf       _B          ;
        btfss       PORTC,4     ; Проверить вывод RC4. Если он равен
        goto        summa       ; нулю, то выполнить сложение A + B
        btfsc       PORTC,4     ; Иначе – выполнить вычитание A – B
        goto        raznost     ;

summa:                          ; Метка подпрограммы сложения A и B
        movf        _A,W        ; Занести в W число A
        addwf       _B,W        ; Выполнить сложение A + B
        movwf       PORTB       ; Вывести на PORTB сумму чисел A и B
        btfss       STATUS,C    ; Проверить состояние бита заема C
        bcf         PORTA,5     ; в регистре STATUS и вывести его
        btfsc       STATUS,C    ; значение на RA5
        bsf         PORTA,5     ;
        goto        metka       ; Повторить выполнение программы

raznost:                        ; Метка подпрограммы вычитания A и B
        movf        _B,W        ; Занести в W число B
        subwf       _A,W        ; Выполнить вычитание A – B
        movwf       PORTB       ; Вывести на PORTB разность чисел A и B
        btfss       STATUS,C    ; Проверить состояние бита заема C
        bcf         PORTA,5     ; в регистре STATUS и вывести его
        btfsc       STATUS,C    ; значение на RA5
        bsf         PORTA,5     ;
        goto        metka       ; Повторить выполнение программы

        END                     ; Конец текста программы
```

3.3. Лабораторное задание

1. Изучить систему команд, организацию памяти данных, способы адресации и систему флагов МК PIC16F886.

2. Ввести программу, приведенную в подразд. 3.2, в среду MPLab IDE, проверить ее работоспособность. Загрузить программу в МК PIC16F886 и проверить правильность ее работы на аппаратном уровне.

3. Получить у преподавателя задание, которое должно содержать значения переменных A и B, а также набор логических и арифметических операций над ними.

4. Выполнить задание в среде MPLab IDE и на макетной плате с установленным МК PIC16F886. Результаты выполнения операций контролировать на выводах PORTB.

5. Подготовить отчет по работе, который должен содержать: титульный лист, цель работы, лабораторное задание, принципиальную схему лабораторного макета, листинг с результатами выполненных преобразований и выводы по работе.

3.4. Контрольные вопросы

1. Поясните понятия «структура команды» и «формат команды».
2. На какие группы подразделяются команды, исполняемые МК PIC16F886?
3. Поясните особенности форматов команд PIC16F886 различных групп.
4. Поясните состав, назначение и использование программистом флагов в регистре STATUS.
5. Поясните, как в микроконтроллере реализована прямая адресация данных.
6. Каким образом в МК PIC16F886 организуется косвенная адресация операндов?
7. Выделите в системе команд все логические операции и поясните суть выполняемых преобразований.
8. Выделите в системе команд все арифметические команды и пояснить суть выполняемых преобразований.

4. Лабораторная работа «Программирование и отладка процедур формирования импульсных сигналов с заданными временными параметрами»

4.1. Цель работы

Изучить программные и аппаратно-программные методы формирования задержек в микропроцессорных устройствах.

Приобрести навыки программирования процедур формирования импульсных сигналов с заданными временными параметрами.

4.2. Краткие теоретические сведения

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

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

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

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

Рассмотрим фрагмент программы с циклическим участком для формирования задержки.

```
N       EQU   20h     ; Резервируем ячейку памяти по адресу 20h под счетчик циклов
        …
        MOVLW     07h     ; Заносим число 7 в рабочий регистр,
        MOVWF     N       ; а затем переписываем его в счетчик циклов
Cycl                      ; Начало циклического участка программы
```

```
        NOP             ; Команды NOP для удлинения времени выполнения участка
        NOP             ; В реальной программе команды NOP могут отсутствовать
        NOP             ;
        DECFSZ  N,1     ; Вычитание из счетчика циклов единицы и проверка условия
        GOTO    Cycl    ; выхода из цикла
```

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

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

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

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

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

Рассмотрим реализацию генератора прямоугольных импульсов (меандра) на базе таймера TMR0. Пусть выходной сигнал формируется на выводе

RA0, а частота синхронизации микроконтроллера равна 10 МГц. Тогда длительность командного цикла $t_{кц}$ = 0,4 мкс.

```
; Пример программной  реализации генератора прямоугольных импульсов
        LIST        p=16f886
        #INCLUDE    p16f886.inc    ; Подключение файла p16f886.inc, в котором описаны
                                   ; имена регистров МК и их физические адреса

; Определяем константы и адреса регистров, флагов и переменных

K       EQU   0AAh                 ; Начальное значение таймера TMR0=170
W_TEMP      EQU   20h              ; Ячейка памяти для сохранения регистра W
                                   ; на время обработки прерывания
STATUS_TEMP       EQU   21h        ; Ячейка памяти для сохранения регистра
                                   ; STATUS на время обработки прерывания
                                   ; Начало программы
        ORG    0                   ; Разместить программу с адреса 0
        GOTO        Begin          ; Перейти к началу основной программы
        ORG   4                    ; Разместить программу с адреса 04
        GOTO        Int            ; Перейти на программу обработки прерывания

Begin           ; Начало основной программы

; Выполняем настройки режимов работы функциональных узлов микроконтроллера
        banksel     ANSEL          ; Выбрать банк  с регистрами  ANSEL и ANSELH
        clrf    ANSEL              ; Очистить регистры ANSEL и ANSELH,
        clrf    ANSELH             ; настроить порты ввода/вывода как цифровые
        banksel     TRISA          ; Выбрать банк памяти 1
        BCF         TRISA,RA0      ; Порт RA0 включаем как выход
        BCF     OPTION_REG,T0CS    ;Включить внутреннее тактирование TMR0
        BSF     INTCON,T0IE        ; Разрешить прерывания по переполнению TMR0
        BCF         STATUS,RP0;     Включить банк регистров 0
        CLRF        PORTA          ; Обнулить выходной сигнал
        MOVLW       K              ; Загрузка TMR0
        MOVWF       TMR0           ; начальным значением
        BSF     INTCON,GIE         ; Разрешить прерывания по переполнению TMR0

; Здесь могут располагаться команды, выполняющие некоторые полезные действия,
; например, обслуживание индикатора, ввод информации с клавиатуры и т.д.

Wait    GOTO    Wait               ; Зацикливание участка программы в ожидании
                                   ;прерывания. Выход отсюда возможен только по
                                   ; прерыванию.

Int     ; Начало подпрограммы обработки прерывания. Здесь на каждое прерывание
        ; производится инверсия порта А. Поскольку на выход настроен только
        ; вывод RA0, то инверсии будет подвергаться только он.
```

```
MOVWF   W_TEMP          ; Сохранить W в регистре TEMP
SWAPF   STATUS,0        ;
MOVWF   STATUS_TEMP     ; Сохранить STATUS в регистре TEMP
BCF     STATUS,RP0      ; Включаем банк регистров 0
COMF    PORTA,1         ; Инверсия выходной линии
MOVLW   K               ; Запись начальной константы K
MOVWF   TMR0            ; в таймер TMR0
BCF     INTCON, T0IF    ; Сброс флага прерывания по переполнению
                        ; TMR0
SWAPF   STATUS_TEMP,0   ; Восстановление STATUS и W
MOVWF   STATUS          ;
SWAPF   W_TEMP,1        ;
SWAPF   W_TEMP,0        ;
RETFIE          ; Вернуться из прерывания, разрешить следующее прерывание
END             ; Конец программы
```

4.3. Лабораторное задание

1. Изучить систему прерываний, модуль счетчика-таймера МК PIC16F886, методику разработки схем алгоритмов и методы формирования задержек.

2. Получить задание у преподавателя.

3. Составить схему алгоритма, написать и отладить в среде MPLAB программу генерирования на заданном выводе микроконтроллера периодического импульсного сигнала с заданной длительностью импульсов и заданной паузой между импульсами.

Для формирования длительности импульса использовать метод многократного циклического повторения цепочки команд, а для формирования паузы – метод с применением счетчика-таймера.

Длительности импульсов рекомендуется задавать в пределах от 30 до 1000 мкс, длительности пауз – от 1 до 24 мс.

Частоту тактирования микроконтроллера принять равной 10 МГц.

5. Загрузить отлаженную программу в микроконтроллер PIC16F886, установленный на макетную плату, и проверить работу программы, контролируя выходной сигнал с помощью осциллографа.

5. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, разработанную схему алгоритма и листинг программы генерирования импульсов, принципиальную схему макета.

4.4. Контрольные вопросы

1. Поясните определение термина «алгоритм».

2. Назовите и поясните суть этапов решения задачи на компьютере.

3. Поясните правила составления схем алгоритмов.

4. Поясните основные правила записи команд на Ассемблере.

5. Назовите и поясните назначение основных директив Ассемблера.

6. Поясните принцип формирования задержек с помощью циклического повторения участка программы.

7. Поясните принцип формирования задержек с помощью счетчика-таймера.

8. С помощью интегрированной среды MPLAB IDE докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

5. Лабораторная работа «Программирование и отладка процедур управления устройствами отображения цифровой информации»

5.1. Цель работы

Изучить схемотехнику и алгоритмы функционирования устройств отображения данных в микропроцессорных устройствах.

Приобрести навыки программирования процедур управления индикаторными устройствами.

5.2. Краткие теоретические сведения

В микропроцессорных системах отображение информации осуществляется с помощью светодиодных или жидкокристаллических индикаторов. В зависимости от конфигурации отдельных сегментов и их взаимного размещения различают семисегментные и матричные индикаторы.

В настоящей работе ограничимся рассмотрением наиболее простого варианта отображения цифровых данных – на светодиодных семисегментных индикаторах.

На рис. 5.1 показано размещение и условное обозначение светодиодных сегментов в цифровых разрядах индикаторов. Для отображения какой-либо цифры соответствующую комбинацию сегментов активизируют, пропуская через них ток. Величина рабочего тока сегмента составляет несколько миллиампер.

На рис. 5.2 показана принципиальная электрическая схема одноразрядного семисегментного светодиодного индикатора, приведен вариант схемы с общим катодом. Промышленность выпускает также варианты схем с общим анодом. (В дальнейшем все пояснения будем проводить для варианта индикатора с общим катодом без дополнительных оговорок.) В процессе работы ток общего вывода r индикатора в семь раз больше, чем ток одного сегмента. В схеме подключения индикатора к МК при недостаточной нагрузочной способности портов в цепях управления общими выводами разрядов индикатора необходимо предусматривать усилители.

Используются два способа управления сегментными индикаторами: статический и динамический.

При статическом способе на каждый цифровой разряд индикатора предусматривается свой регистр, в регистры записываются и хранятся на все время отображения семисегментные коды соответствующих цифр. Выходы

регистров соединяются через токоограничивающие резисторы с входами соответствующих сегментов разрядов индикатора, общие выводы разрядов индикатора подключаются к общему проводу. Способ отличается простотой программного обслуживания, но требует повышенных аппаратных затрат (регистров и резисторов), применяется для малоразрядных индикаторов.

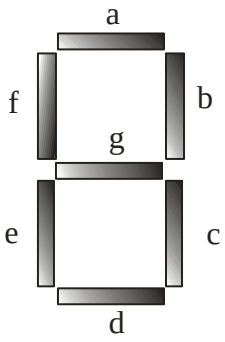


Рис. 5.1. Размещение сегментов в разрядах индикатора

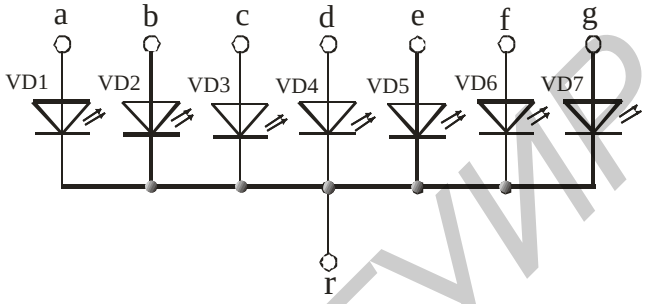


Рис. 5.2. Принципиальная электрическая схема разряда индикатора

При большом количестве разрядов применяют динамический способ управления индикатором. Способ характеризуется минимальными аппаратными затратами, но сравнительно сложной программной поддержкой.

Схема подключения четырехразрядного индикатора в динамическом режиме к микроконтроллеру показана на рис. 5.3. В схеме одноименные сегменты всех разрядов объединяются и обслуживаются семью выводами одного из портов микроконтроллера. Общие выводы разрядов индикатора обслуживаются индивидуально четырьмя выводами другого порта микроконтроллера. Первый порт обеспечивает возбуждение определенной для каждой цифры комбинации сегментов, а второй порт активизирует тот или иной разряд индикатора. Суть динамического управления индикатором сводится к поочередному циклическому возбуждению разрядов индикатора. Если разряды индикатора будут подсвечиваться с частотой выше 25 Гц, то в силу инерционности человеческого зрения мерцания изображения заметны не будут. Временные диаграммы работы схемы динамической индикации приведены на рис. 5.4.

Ниже приводится программа управления индикатором в динамическом режиме. Программа составлена для схемы подключения индикатора к микроконтроллеру, изображенной на рис. 1.5. Отличие этого варианта схемы от

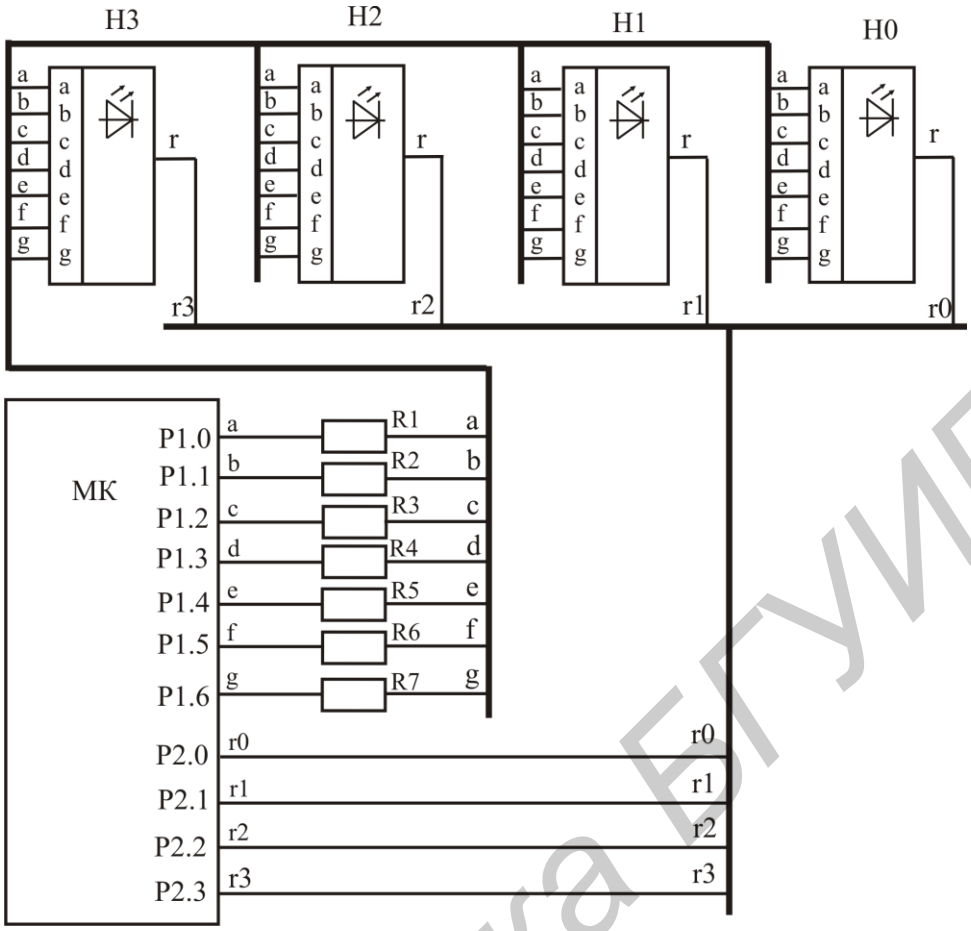


Рис. 5.3. Схема подключения индикатора к микроконтроллеру в динамическом режиме

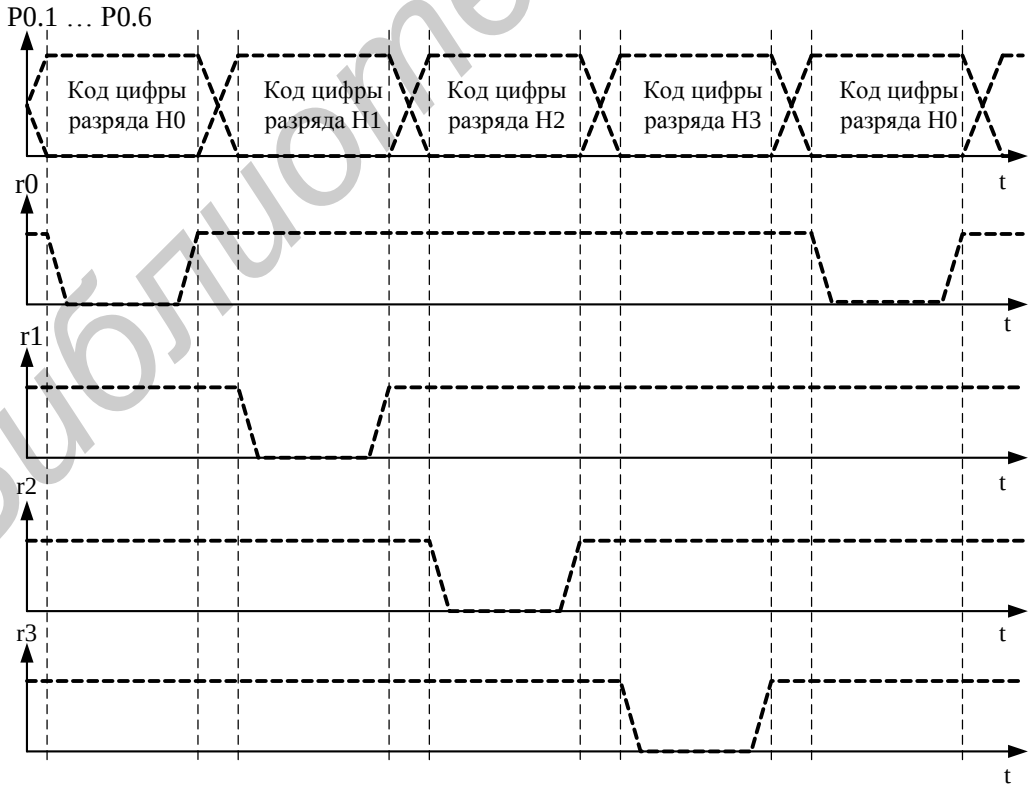


Рис. 5.4. Временные диаграммы работы схемы динамической индикации

рассмотренного выше в том, что сигналы возбуждения разрядов индикатора усиливаются электронными ключами на биполярных транзисторах. При этом сигналы подвергаются инверсиям, так что активными уровнями на выходе порта микроконтроллера становятся логические единицы вместо нулей, как показано в примере на рис. 5.3 и 5.4.

```
; Программа управления четырехразрядным цифровым индикатором
        list            p=16f886                ; Выбираем тип контроллера
        #include        <p16f886.inc>           ; Подключаем файл с описанием имён регистров
                                                ; и битов используемого микроконтроллера
        errorlevel      -302    ; Отключаем вывод сообщений о некорректном банке памяти
; Формируем слово конфигурации микроконтроллера
        __CONFIG    _CONFIG1, _LVP_OFF & _FCMEN_ON & _IESO_OFF & _BOR_OFF
& _CPD_OFF &_CP_OFF & _MCLRE_ON & _PWRTE_ON & _WDT_OFF & _HS_OSC
        __CONFIG    _CONFIG2, _WRT_OFF & _BOR21V
; Задаём адреса для хранения переменных при сохранении контекста прерывания в ОЗУ
; быстрого доступа
W_TEMP          EQU   0x7D          ; Переменная для сохранения регистра W
STATUS_TEMP     EQU   0x7E          ; Переменная для сохранения регистра STATUS
PCLATH_TEMP     EQU   0x7F          ; Переменная для сохранения регистра PCLATH
; Задаём блок пользовательских переменных с расположением первой по адресу 0x20
        CBLOCK      0x20
INB                             ; Номер текущего разряда
BUF0                            ; Резервируем четыре ячейки для хранения числа
BUF1                            ;
BUF2                            ;
BUF3                            ;
        ENDC                    ; Конец блока пользовательских переменных
;-----------------------------------------------------------------------------
;       Начало текста программы
;-----------------------------------------------------------------------------
        ORG   0x000             ; Установить адрес начала программы (вектор сброса)
        goto    MAIN            ; Перейти на начало основной программы
        ORG   0x004             ; Установить адрес начала обработки прерываний
INTERRUPT:                      ; Подпрограмма обработки прерываний
    ; Сохранение контекста
        movwf       W_TEMP          ; Сохранение текущего значения регистра W
        movf        STATUS,W        ; Пересылка регистра STATUS в W
        movwf       STATUS_TEMP     ; Сохраняем текущее значение регистра STATUS
        Movf        PCLATH,w        ; Пересылка регистра PCLATH в W
        movwf     PCLATH_TEMP       ; Сохранение текущего состояния PCLATH
INT_TMR0:                           ; Обработка прерывания по переполнению TMR0
        btfsc    INTCON,T0IF        ; Было ли прерывание от TMR0?
```

```
        btfss   INTCON,T0IE         ; Было ли разрешено прерывание от TMR0?
        goto    INT_TMR0_SKIP       ; Прерывания не было или оно запрещено, поэтому
                                    ; Пропускаем обработку и переходим к следующему
                                      прерыванию
        banksel         TMR0        ; Выбираем банк памяти регистра TMR0
        movlw           .210        ; Корректируем значение TMR0 для получения
        movwf           TMR0        ; требуемой задержки
        bcf     INTCON,T0IF         ; Сбрасываем флаг прерывания по TMR0
        call    INDIC               ; Вызываем подпрограмму обслуживания индикатора
INT_TMR0_SKIP:                      ; Сюда производится переход при отсутствии
                                    ; прерывания от TMR0 или при его запрете.
;-------------------------------------------------------------------------------
; Здесь могут быть коды обработчиков прерываний от других источников
;-------------------------------------------------------------------------------
; Восстанавливаем контекст и возвращаемся из подпрограммы обработки прерываний
INT_RETURN:
        movf    PCLATH_TEMP,W   ; Восстанавливаем значение регистра PCLATH в W
        movwf    PCLATH         ; Восстанавливаем значение регистра PCLATH
        movf    STATUS_TEMP,W   ; Восстанавливаем значение регистра STATUS в W
        movwf           STATUS  ; Восстанавливаем значение регистра STATUS
        swapf   W_TEMP,F        ; Восстанавливаем значение регистра W
        swapf   W_TEMP,W        ;
        retfie                  ; Возвращаемся из подпрограммы обработки прерываний

MAIN:           ;  ОСНОВНАЯ ПРОГРАММА
                ; Производим инициализацию микроконтроллера
        call    INIT_PORT       ; Вызываем подпрограмму настройки портов
        call    INIT_TMR0       ; Вызываем подпрограмму настройки TMR0
        banksel         INB     ; Включаем в банк памяти, где расположен регистр INB
        clrf    INB             ; Очищаем буфер номера текущего разряда
        clrf    BUF0            ; Очищаем буфер разряда 0
        clrf    BUF1            ; Очищаем буфер разряда 1
        clrf    BUF2            ; Очищаем буфер разряда 2
        clrf    BUF3            ; Очищаем буфер разряда 3
        bcf     INTCON,T0IF     ; Сбрасываем флаг прерывания по таймеру TMR0
        bsf     INTCON,T0IE     ; Разрешаем прерывания по таймеру TMR0
        bsf     INTCON,PEIE     ; Разрешаем прерывания от периферийных модулей
        bsf     INTCON,GIE      ; Разрешаем все немаскированные прерывания
;-------------------------------------------------------------------------------
;       Здесь может располагаться основная программа микроконтроллера, которая будет
;       выполняться циклически бесконечно (пока подано питающее напряжение)
;-------------------------------------------------------------------------------
MAIN_LOOP:
; Эта программа никаких действий, кроме индикации, не выполняет, поэтому заносим
; числа в буфер индикации и постоянно ждем прерывания
```

```
        movlw   .3              ; Заносим число в W
        movwf   BUF0            ; Записываем его в буфер индикации 0-го разряда
        movlw   .2              ; Заносим число в W
        movwf   BUF1            ; Записываем его в буфер индикации 1-го разряда
        movlw   .1              ; Заносим число в W
        movwf   BUF2            ; Записываем его в буфер индикации 3-го разряда
        movlw   .0              ; Заносим число в W
        movwf   BUF3            ; Записываем его в буфер индикации 3-го разряда
        goto    $               ; Ждем прерывание в бесконечном цикле
;-------------------------------------------------------------------------------
;       ПОДПРОГРАММЫ
;-------------------------------------------------------------------------------
; Подпрограмма инициализации портов ввода/вывода
; Распределение портов микроконтроллера для управления сегментами индикатора:
; a – RC0, b – RC1, c – RC2, d – RC3, e – RC4, f – RC5,   g – RC6, h – RC7
; Распределение портов микроконтроллера для управления  выводами разрядов:
; 0-й разряд – RA1, 1-й разряд – RA2, 2-й разряд – RA3, 3-й разряд  – RA4
INIT_PORT:
        banksel PORTA           ; Выбираем банк памяти с регистром PORTA
        clrf    PORTA           ; Инициализируем PORTA
        clrf    PORTC           ; Инициализируем PORTC
        banksel ANSEL           ; Выбираем банк памяти с регистром ANSEL
        clrf    ANSEL           ; Настраиваем все порты как цифровые
        clrf    ANSELH          ; Настраиваем все порты как цифровые
        banksel TRISA           ; Выбираем банк памяти с регистром TRISA
        movlw   .0              ; Все выводы PORTA на выход
        movwf   TRISA           ;
        movlw   .0              ; Все выводы PORTC на выход
        movwf   TRISC           ;
        return                  ; Возврат из подпрограммы

; Подпрограмма инициализации Timer0
INIT_TMR0:
        banksel TMR0            ; Выбираем банк памяти с регистром TMR0
        clrf    TMR0            ; Инициализируем TMR0
        banksel OPTION_REG      ; Выбираем банк памяти с регистром OPTION_REG
        movlw   b'11000100'     ; Настраиваем TMR0: тактирование – внутреннее,
        movwf   OPTION_REG      ; предделитель 1:32 включен перед TMR0
        return                  ; Возврат из подпрограммы

; Подпрограмма обслуживания индикатора
INDIC:
        banksel PORTA           ; Выбираем банк памяти с регистром PORTA
        movlw   b'11100001'     ; Гасим индикатор. Для этого выполняем побитное И
        andwf   PORTA,F         ; содержимого порта и маски в регистре W
                                ; Данная операция необходима для того, чтобы не
```

```
                                ; происходило изменение состояния других выводов
                                ; порта
        movlw       BUF0        ; Пересылаем в W адрес начала буфера индикации
        addwf       INB,W       ; Вычисляем косвенный адрес текущего разряда
        movw  f     FSR         ; Пересылаем косвенный адрес в регистр FSR
        movf        INDF,W      ; В W имеем двоичный код цифры в текущем разряде
        call        TAB1        ; Вызываем подпрограмму перекодировки его в
                                ; семисегментный двоичный код
        movwf       PORTC       ; Выводим полученный код на PORTC
        movf        INB,W       ; Пересылаем двоичный код номера текущего разряда в W
        call        TAB2        ; Вызываем подпрограмму преобразования кода
                                ; текущего разряда в позиционный код
        iorwf       PORTA,F     ; Подсвечиваем текущий разряд командой поразрядного
                                ; ИЛИ
; над текущим значением PORTA и маской текущего разряда. При этом состояние
; других выводов PORTA, не связанных с работой индикатора, не изменяется.
        incf        INB,F       ; Меняем номер текущего разряда. INB = INB + 1
        btfsc       INB,2       ; Если отобразили четыре разряда,
        clrf        INB         ; то сбрасываем в ноль счётчик текущего разряда
        return                  ; Возврат из подпрограммы

; Подпрограмма перекодировки двоичного кода числа в семисегментный код
TAB1:
        addwf     PCL,F     ; Прибавляем к значению счетчика команд кодируемое число
        retlw   b'00111111'         ; Возвращаем в W семисегментный код цифры 0
        retlw   b'00000110'         ; Возвращаем в W семисегментный код цифры 1
        retlw   b'01011011'         ; Возвращаем в W семисегментный код цифры 2
        retlw   b'01001111'         ; Возвращаем в W семисегментный код цифры 3
        retlw   b'01100110'         ; Возвращаем в W семисегментный код цифры 4
        retlw   b'01101101'         ; Возвращаем в W семисегментный код цифры 5
        retlw   b'01111101'         ; Возвращаем в W семисегментный код цифры 6
        retlw   b'00000111'         ; Возвращаем в W семисегментный код цифры 7
        retlw   b'01111111'         ; Возвращаем в W семисегментный код цифры 8
        retlw   b'01101111'         ; Возвращаем в W семисегментный код цифры 9

;  Подпрограмма преобразования двоичного кода текущего разряда в позиционный
TAB2:
        addwf    PCL,F      ; Прибавляем к значению счетчика команд кодируемое число
        retlw   b'00000010'         ; Маска позиционного кода разряда 0
        retlw   b'00000100'         ; Маска позиционного кода разряда 1
        retlw   b'00001000'         ; Маска позиционного кода разряда 2
        retlw   b'00010000'         ; Маска позиционного кода разряда 3

        END                         ; Директива "Окончание текста программы"
```

5.3. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 5.2.

2. Получить задание у преподавателя. В задании должны быть указаны:

а) разрядность (два или три) обслуживаемого индикатора в динамическом режиме;

б) десятичное число для отображения на индикаторе.

3. Составить схему подключения индикатора к микроконтроллеру, схему алгоритма программы, а также написать и отладить в среде MPLAB программу управления индикатором в динамическом режиме в соответствии с индивидуальным заданием.

4. Загрузить отлаженную программу в микроконтроллер PIC16F886, установленный на макетную плату, и проверить работу программы, визуально контролируя отображаемое число на индикаторе.

5. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, схему подключения индикатора, схему алгоритма работы программы и листинг программы управления индикатором.

5.4. Контрольные вопросы

1. Поясните принцип статического режима работы индикатора.

2. Поясните принцип динамического режима работы индикатора.

3. Из каких соображений выбирается частота переключения разрядов индикатора в динамическом режиме?

4. Поясните алгоритм работы программы табличного преобразования данных для микроконтроллера PIC16F886.

5. Поясните схему алгоритма работы программы управления индикатором для вашего варианта задания.

6. С помощью интегрированной среды MPLAB докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

6. Лабораторная работа
«Программирование и отладка процедур ввода данных с клавиатуры»

6.1. Цель работы

Изучить принципы организации ввода данных с клавиатуры в микропроцессорных устройствах.

Приобрести навыки программирования процедур ввода данных с клавиатуры.

6.2. Краткие теоретические сведения

Функцию ввода данных с клавиатуры в микропроцессорных устройствах реализуют как отдельный самостоятельный аппаратно-программный модуль. На него отводятся определенные аппаратные и программные ресурсы микроконтроллера.

Для небольшого количества клавиш для обслуживания каждой из них выделяется отдельный вывод микроконтроллера, как показано на рис. 2.1.

Для большого количества клавиш используют матричную схему, которая требует значительно меньшего количества выводов микроконтроллера. Схема подключения к микроконтроллеру (МК) 16-кнопочной клавиатуры показана на рис. 6.1. Как видно, для обслуживания 16 кнопок требуется всего лишь восемь выводов МК. Выводы P1.0 … P1.3 микроконтроллера настраиваются как выходы и являются линиями сканирования состояния клавиатуры. Выводы P2.0 … P2.3 настраиваются как входы и являются линиями опроса состояний кнопок клавиатуры. Линии сканирования образуют строки, а линии опроса – столбцы матрицы клавиатуры. Потенциалы линий опроса с помощью резисторов R1 … R4 подтягиваются до уровня напряжения питания. При опросе состояния клавиатуры при незамкнутых клавишах на этих линиях читаются логические единицы. В случае нажатия какой-либо клавиши происходит электрическое замыкание линий строки и столбца, на пересечении которых находится нажатая кнопка. Если при этом на эту линию строки выведен логический нуль, то на линии столбца с нажатой кнопкой будет читаться также логический нуль. Таким образом, при опросе состояния клавиатуры микроконтроллер по линиям опроса будет читать для каждой нажатой кнопки свой уникальный код возврата KW, который может быть впоследствии преобразован в двоичный код нажатой кнопки KNK. Код возврата нечувствителен к строке, в которой находится нажатая кнопка, поэтому необходима корректировка KNK. Она может быть выполнена по формуле KNK = KNK + 4 *i*, где *i* – номер строки с нажатой кнопкой, принимает значения от 0 до 3.

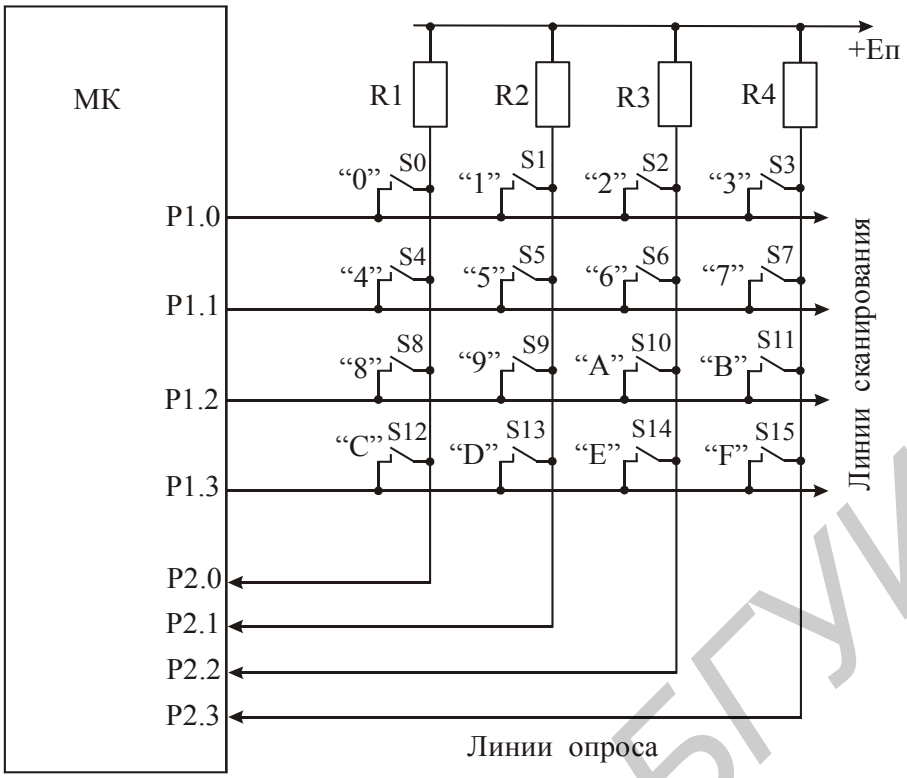


Рис. 6.1. Схема подключения клавиатуры к микроконтроллеру

Взаимодействие процедуры обслуживания клавиатуры, назовем ее KLAV, с другими процедурами программы лучше всего обеспечивать через буфер клавиатуры BK, куда процедура KLAV заносит двоичный код нажатой кнопки.

При программном обслуживании клавиатуры необходимо решать так или иначе следующий набор задач:

1) контроль и ожидание освобождения буфера клавиатуры,
2) определение факта нажатия кнопки,
3) устранение дребезга контактов,
4) определение кода нажатой кнопки,
5) занесение в буфер кода нажатой кнопки,
6) возведение флага буфера клавиатуры.

Рассмотрим подробнее способы решения названных задач.

Алгоритм контроля и ожидания освобождения буфера клавиатуры может базироваться на контроле значения флага буфера клавиатуры FBK. Если FBK равно нулю, то буфер пуст, если FBK = 1, то буфер занят. Процедура KLAV заносит в буфер код нажатой клавиши, возводит в единицу FBK, а другие процедуры забирают оттуда код и сбрасывают флаг в нуль. При этом пока флаг остается возведенным в единицу, процедура KLAV игнорирует нажатие кнопок.

Определение факта нажатия кнопки удобно сделать через дополнительную переменную – флаг нажатия кнопки – FNK, которая отражает текущее состояние кнопок: нажата – FNK = 1 или не нажата – FNK = 0, как показано на рис. 6.2. Производя логический анализ двух соседних значений FNK (текущего FNKI и предыдущего FNKJ), легко фиксировать момент нажатия кнопок. Нажатие будет иметь место, если выполняется условие $FNKi \wedge \overline{FNKj} = 1$.

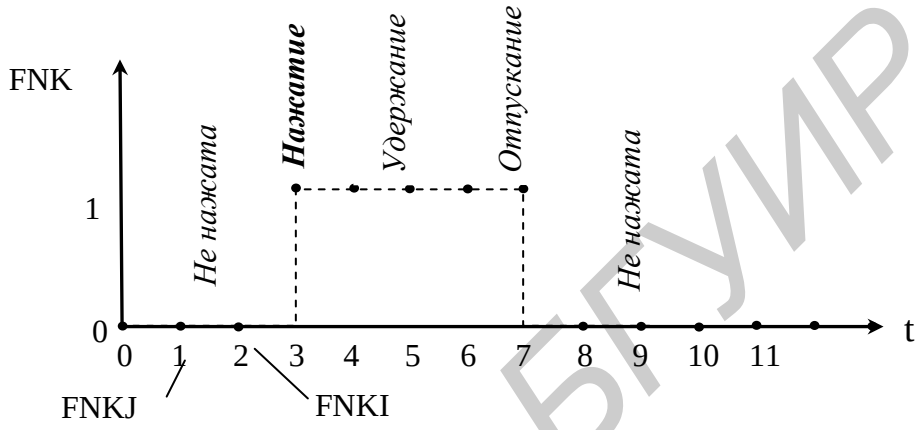


Рис. 6.2. Состояния кнопок

Значения FNK можно определить по следующему алгоритму:

– по линиям сканирования выставить логические нули;

– прочитать код возврата KW на линиях опроса;

– если KW = F, то FNK = 0, кнопки не нажаты;

– если KW ≠ F, то FNK =1, одна или более кнопок нажаты.

Устранение дребезга контактов лучше всего делать путем опроса состояния клавиатуры с частотой (5…20 Гц). При этом все переходные процессы обычно завершаются до следующего опроса состояния кнопок. Периодический опрос клавиатуры лучше всего организовать по прерываниям от переполнения счетчика-таймера. В реальных микропроцессорных системах опрос клавиатуры обычно совмещают в цикле с обслуживанием индикатора в динамическом режиме.

Определение кода нажатой кнопки KNK проводят методом последовательной проверки гипотез: нажатая кнопка находится на линии 0-й строки, 1-й строки, 2-й строки, 3-й строки.

Для проверки, находится ли нажатая кнопка в *i*-й строке (где $i = 0 \ldots 3$), на ней выставляется логический нуль, а на всех других – логические единицы.

Читают код возврата KW на линиях опроса. Если KW = F, то нажатая кнопка не в этой строке. Если KW ≠ F, то нажатая кнопка в этой строке.

Далее преобразуют позиционный код нажатой кнопки в двоичный по правилам: если KW = E, то KNK = 0; если KW = D, то KNK = 1; если KW = B, то KNK = 2; если KW = 7, то KNK = 3.

И последний шаг этой процедуры – корректировка KNK с учетом веса строки с нажатой кнопкой по формуле KNK = KNK + 4*i*, где *i* – номер строки с нажатой кнопкой.

Занесение в буфер кода нажатой кнопки обеспечивается операцией BK = KNK.

Возведение флага буфера клавиатуры реализуется операцией FBK = 1.

; Программа обслуживания 16-кнопочной клавиатуры

```
        list        p=16f886            ; Выбираем тип контроллера
        #include    <p16f886.inc>       ; Подключаем файл с описанием имен регистров
                                        ; и битов используемого микроконтроллера
        errorlevel  -302    ; Отключаем вывод сообщений о некорректном банке памяти

; Формируем слово конфигурации микроконтроллера
        __CONFIG    _CONFIG1, _LVP_OFF & _FCMEN_ON & _IESO_OFF & _BOR_OFF
& _CPD_OFF &_CP_OFF & _MCLRE_ON & _PWRTE_ON & _WDT_OFF & _HS_OSC
        __CONFIG    _CONFIG2, _WRT_OFF & _BOR21V

; Задаём адреса для хранения переменных при сохранении контекста прерывания в ОЗУ
; быстрого доступа
W_TEMP          EQU   0x7D  ; Переменная для сохранения регистра W
STATUS_TEMP     EQU   0x7E  ; Переменная для сохранения регистра STATUS
PCLATH_TEMP     EQU   0x7F  ; Переменная для сохранения регистра PCLATH

; Задаём блок пользовательских переменных с расположением первой по адресу 0x20
        CBLOCK      0x30
BK              ; Буфер для хранения кода нажатой клавиши
KW              ; Регистр для хранения кода возврата при опросе состояния клавиш
FBK             ; Флаг буфера клавиатуры, сигнализирует о наличии
                ; необработанной нажатой клавиши
FNK             ; Флаг нажатия клавиши
FNKI            ; Текущее значение FNK
FNKJ            ; Предыдущее значение FNK
KNK             ; Код нажатой клавиши
        ENDC            ; Конец блока пользовательских переменных
;-------------------------------------------------------------------------------
;       Начало текста программы
;-------------------------------------------------------------------------------
        ORG   0x000         ; Установить адрес начала программы (вектор сброса)
        Goto  MAIN          ; Перейти на начало основной программы
```

```
;-------------------------------------------------------------------------------
;       Подпрограмма обработки прерываний
;-------------------------------------------------------------------------------
        ORG   0x004         ; Установить адрес начала подпрограммы обработки
                            ; прерываний (вектор прерывания)
INTERRUPT:
; Сохраняем контекст
        movwf   W_TEMP      ; Сохраняем текущее значение регистра W
        movf    STATUS,W    ; Пересылаем содержимое регистра STATUS в W
        movwf   STATUS_TEMP ; Сохраняем текущее значение регистра STATUS
        movf    PCLATH,W    ; Пересылаем содержимое регистра PCLATH в W
        movwf   PCLATH_TEMP ; Сохраняем текущее значение регистра PCLATH
;-------------------------------------------------------------------------------
;- Обработка прерывания по переполнению TMR0
;-------------------------------------------------------------------------------
INT_TMR0:
        btfsc   INTCON,T0IF     ; Было ли прерывание от TMR0?
        btfss   INTCON,T0IE     ; Было ли разрешено прерывание от TMR0?
        goto    INT_TMR0_SKIP   ; Прерывания не было или оно запрещено, поэтому
; пропускаем обработку и переходим к следующему  обработчику прерываний
        banksel     TMR0        ; Выбираем банк памяти регистра TMR0
        movlw       .210        ; Корректируем значение TMR0 для получения
        movwf       TMR0        ; требуемой задержки
        bcf       INTCON,T0IF   ; Сбрасываем флаг прерывания по TMR0
        call        KLAV        ; Вызываем подпрограмму обслуживания клавиатуры
        movf        BK,W        ; Полученный код клавиши выводим на индикатор
        movwf       BUF0        ;
        call    INDIC           ; Вызываем подпрограмму обслуживания индикатора
        clrf    FBK             ; Очищаем флаг буфера клавиатуры
INT_TMR0_SKIP:                  ; Сюда производится переход при отсутствии
                                ; прерывания от TMR0 или при его запрете
;-------------------------------------------------------------------------------------------------------
;  Здесь могут быть коды обработчиков прерываний от других источников
;-------------------------------------------------------------------------------------------------------
; Восстанавливаем контекст и возвращаемся из подпрограммы обработки прерываний
        movf   PCLATH_TEMP,W  ; Восстанавливаем значение регистра PCLATH в W
        movwf       PCLATH    ; Восстанавливаем значение регистра PCLATH
        movf   STATUS_TEMP,W  ; Восстанавливаем значение регистра STATUS в W
        movwf       STATUS    ; Восстанавливаем значение регистра STATUS
        swapf   W_TEMP,F      ; Восстанавливаем значение регистра W
        swapf   W_TEMP,W      ;
        retfie                ; Возвращаемся из подпрограммы обработки прерываний
```

```
;-------------------------------------------------------------------------------
MAIN:           ; ОСНОВНАЯ ПРОГРАММА
;-------------------------------------------------------------------------------
; Производим инициализацию микроконтроллера
        call    INIT_PORT   ; Вызываем подпрограмму настройки портов ввода/вывода
        call    INIT_TMR0           ; Вызываем подпрограмму настройки TMR0
        call    INIT_KLAV           ; Вызываем подпрограмму настройки PORTB для
                                    ; работы с клавиатурой
        banksel INTCON      ; Переходим в банк памяти, где расположен регистр INTCON
        bcf     INTCON,T0IF         ; Сбрасываем флаг прерывания по таймеру TMR0
        bsf     INTCON,T0IE         ; Разрешаем прерывания по таймеру TMR0
        bsf     INTCON,GIE          ; Разрешаем все немаскированные прерывания
;-------------------------------------------------------------------------------------------------------------
;       Здесь может располагаться основная программа микроконтроллера, которая будет
;       выполняться циклически бесконечно (пока подано питающее напряжение)
;-------------------------------------------------------------------------------------------------------------
MAIN_LOOP:
; Наша программа никаких действий, кроме обслуживания клавиатуры, не выполняет,
; поэтому постоянно ждём прерывания
        goto    $                   ; Ждём прерывание в бесконечном цикле
;-------------------------------------------------------------------------------
;       ПОДПРОГРАММЫ
;-------------------------------------------------------------------------------
INIT_PORT:          ; Подпрограмма инициализации аортов
        banksel     ANSEL       ; Выбираем банк памяти с регистром ANSEL
        clrf        ANSEL       ; Настраиваем все порты как цифровые
        clrf        ANSELH      ; Настраиваем все порты как цифровые
        return                  ; Возврат из подпрограммы

INIT_TMR0:          ; Подпрограмма инициализации Timer0
        banksel     TMR0        ; Выбираем банк памяти с регистром TMR0
        clrf        TMR0        ; Инициализируем TMR0
        banksel     OPTION_REG  ; Выбираем банк памяти с регистром OPTION_REG
        movlw       b'11000100' ; Настраиваем TMR0: тактирование – внутреннее Fosc/4,
        movwf       OPTION_REG  ; предделитель 1:32 включен перед TMR0
        return                  ; Возврат из подпрограммы
;-------------------------------------------------------------------------------
INIT_KLAV:          ; Подпрограмма настройки PORTB для работы с клавиатурой
        banksel     PORTB               ; Выбор банка памяти
        clrf        PORTB               ; Инициализировать PORTB
        banksel     TRISB               ;
        movlw       b'00001111'         ; Настроить направление выводов порта
        movwf       TRISB               ;
        banksel     WPUB                ; Выбор банка памяти
```

```
        movlw   b'00001111'     ; Включить подтягивающие резисторы
        movwf   WPUB            ; на входах PORTB
        banksel OPTION_REG      ;
        bcf     OPTION_REG,7    ;
        banksel PORTB           ; Выбор банка памяти
        movlw   b'11110000'     ; Установить нули на линиях сканирования
        movwf   PORTB           ;
        clrf    BK              ; Очистить переменные, которые используются
        clrf    KW              ; подпрограммой обслуживания клавиатуры
        clrf    FBK             ;
        clrf    FNK             ;
        clrf    FNKI            ;
        clrf    FNKJ            ;
        clrf    KNK             ;
        return                  ; Возврат из подпрограммы
;-------------------------------------------------------------------------------
KLAV:                           ; Подпрограмма обслуживания клавиатуры
        banksel FNK             ; Выбор банка памяти
        clrf    FNK             ; Обнуляем флаг нажатия клавиши
        btfsc   FBK,0           ; Буфер клавиатуры свободен?
        return                  ; Если нет, то выход из подпрограммы
; Определяем факт нажатия кнопки
        movlw   b'00001111'     ; Обнуляем линии сканирования
        andwf   PORTB,F         ;
        nop                     ; Задержка для установления
        nop                     ; уровней сигнала на выводах PORTB
        nop                     ;
        movf    PORTB,W         ; Чтение состояния PORTB в W
        andlw   b'00001111'     ; Выделяем код на линиях возврата и
        movwf   KW              ; сохраняем его в регистре KW
        sublw   .15             ; Проверяем KW = 15?
        btfss   STATUS,Z        ; Если нет,
        bsf     FNK,0           ; то FNK = 1
        movf    FNKI,W          ; Записываем текущее значение флага
        movwf   FNKJ            ; нажатия клавиши
        movf    FNK,W           ;
        movwf   FNKI            ;
        comf    FNKJ,W          ; Проверяем факт нажатия клавиши путём
        andlw   b'00000001'     ; сравнения с предыдущим значением
        andwf   FNKI,W          ;
        btfsc   STATUS,Z        ;
        return                  ; Нажатия не было, возврат из подпрограммы
; Зафиксировали факт нажатия клавиши. Теперь находим её код
; Подпрограмма определения, в какой строке нажата клавиша
```

```
KLAV_0:
        movlw   b'11101111'     ; Устанавливаем на линии сканирования
        movwf   PORTB           ; RB4 лог. "0"
        nop                     ; Задержка для установления уровней сигнала
        nop                     ; на выводах PORTB
        nop                     ;
        movf    PORTB,W         ; Читаем состояние PORTB в W
        andlw   b'00001111'     ; Выделяем код на линиях возврата
        movwf   KW              ; Сохраняем его в ячейке KW
        sublw   0x0F            ; Проверяем, в этой ли строке была
        btfsc   STATUS,Z        ; нажата клавиша
        goto    KLAV_1          ; Не в этой строке, переходим к проверке
                                ; следующей
        call    KOD             ; Вызываем подпрограмму определения кода
                                ; нажатой клавиши
        movf    KNK,W           ; Определили код. Записываем его в буфер
        movwf   BK              ; клавиатуры
        incf    FBK,F           ; Устанавливаем флаг загрузки в буфер
                                ; клавиатуры числа
        return                  ; Возврат из подпрограммы
KLAV_1:
        movlw   b'11011111'     ; Устанавливаем на линии сканирования
        movwf   PORTB           ; RB5 лог. "0"
        nop                     ; Задержка для установления уровней сигнала
        nop                     ; на выводах PORTB
        nop                     ;
        movf    PORTB,W         ; Читаем состояние PORTB в W
        andlw   b'00001111'     ; Выделяем код на линиях возврата
        movwf   KW              ; Сохраняем его в ячейке KW
        sublw   0x0F            ; Проверяем, в этой ли строке была
        btfsc   STATUS,Z        ; нажата клавиша
        goto    KLAV_2          ; Не в этой строке, переходим к проверке
                                ; следующей
        call    KOD             ; Вызываем подпрограмму определения кода
                                ; нажатой клавиши
        movf    KNK,W           ; Определили код
        addlw   .4              ; Корректируем его с учётом строки
        movwf   BK              ; Записываем его в буфер клавиатуры
        incf    FBK,F           ; Устанавливаем флаг загрузки в буфер
                                ; клавиатуры числа
        return                  ; Возврат из подпрограммы
KLAV_2:
        movlw   b'10111111      ; Устанавливаем на линии сканирования
        movwf   PORTB           ; RB6 лог. "0"
```

```
        nop                     ; Задержка для установления уровней сигнала
        nop                     ; на выводах PORTB
        nop                     ;
        movf    PORTB,W         ; Читаем состояние PORTB в W
        andlw   b'00001111'     ; Выделяем код на линиях возврата
        movwf   KW              ; Сохраняем его в ячейке KW
        sublw   0x0F            ; Проверяем, в этой ли строке была
        btfsc   STATUS,Z        ; нажата клавиша
        goto    KLAV_3          ; Не в этой строке, переходим к проверке
                                ; следующей
        call    KOD             ; Вызываем подпрограмму определения кода
                                ; нажатой клавиши
        movf    KNK,W           ; Определили код
        addlw   .8              ; Корректируем его с учётом строки
        movwf   BK              ; Записываем его в буфер клавиатуры
        incf    FBK,F           ; Устанавливаем флаг загрузки в буфер
                                ; клавиатуры числа
        return                  ; Возврат из подпрограммы
KLAV_3:
        movlw   b'01111111'     ; Устанавливаем на линии сканирования
        movwf   PORTB           ; RB7 лог. "0"
        nop                     ; Задержка для установления уровней сигнала
        nop                     ; на выводах PORTB
        nop                     ;
        movf    PORTB,W         ; Читаем состояние PORTB в W
        andlw   b'00001111'     ; Выделяем код на линиях возврата
        movwf   KW              ; Сохраняем его в ячейке KW
        sublw   0x0F            ; Проверяем, в этой ли строке была нажата
        btfsc   STATUS,Z        ; клавиша
        return                  ; Ложное срабатывание. Возврат
        call    KOD             ; Вызываем подпрограмму определения кода
                                ; нажатой клавиши
        movf    KNK,W           ; Определили код
        addlw   .12             ; Корректируем его с учётом строки
        movwf   BK              ; Записываем его в буфер клавиатуры
        incf    FBK,F           ; Устанавливаем флаг загрузки в буфер
                                ; клавиатуры числа
        return                  ; Возврат из подпрограммы

; Подпрограмма преобразования кода возврата в двоичный код нажатой
; клавиши в строке KOD:
        movf    KW,W            ; Проверяем KW = 0x0E?
        sublw   0x0E            ;
        btfss   STATUS,Z        ; Если нет,
```

```
        goto    KOD_1           ; то переход к следующей проверке
        movlw   .0              ;
        movwf   KNK             ; Возвращаем код нажатой клавиши 0
        return                  ;
KOD_1:
        movf    KW,W            ; Проверяем KW = 0x0D?
        sublw   0x0D            ;
        btfss   STATUS,Z        ; Если нет,
        goto    KOD_2           ; то переход к следующей проверке
        movlw   .1              ;
        movwf   KNK             ; Возвращаем код нажатой клавиши 1
        return                  ; Возврат из подпрограммы
KOD_2:
        movf    KW,W            ; Проверяем KW = 0x0D?
        sublw   0x0B            ;
        btfss   STATUS,Z        ; Если нет,
        goto    KOD_3           ; то переход к следующей проверке
        movlw   .2              ;
        movwf   KNK             ; Возвращаем код нажатой клавиши 2
        return                  ; Возврат из подпрограммы
KOD_3:
        movlw   .3              ; Проверять больше нечего. Остался только
                                ; один вариант
        movwf   KNK             ; Возвращаем код нажатой клавиши 3
        return                  ; Возврат из подпрограммы
;-------------------------------------------------------------------------------
        END                     ; Директива "Окончание текста программы"
```

6.3. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 6.2.

2. Получить у преподавателя индивидуальное задание.

В вариантах заданий использовать:

– количество клавиш меньше 16;

– вариант подключения клавиатуры к микроконтроллеру матричный;

– варианты конфигурации клавиатуры, заданной в виде (количество линий сканирования)x(количество линий опроса) 2 x 2, 2 x 3, 2 x 4, 3 x 2, 3 x 3, 3 x 4;

– в качестве линий сканирования и линий опроса использовать выводы портов A, B или C микроконтроллера PIC16F886;

– в качестве буфера клавиатуры использовать четыре младших вывода порта B.

3. В соответствии с индивидуальным заданием, выданным преподавателем, составить схему подключения клавиатуры к микроконтроллеру, разработать схему алгоритма, составить и отладить в среде MPLAB IDE программу управления клавиатурой.

4. Загрузить отлаженную программу в микроконтроллер PIC16F886, установленный на макетную плату, и проверить работу программы, визуально контролируя двоичный код нажатой клавиши с помощью единичных индикаторов, подключенных на макетной плате к младшим четырем выводам порта B.

5. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, схему подключения клавиатуры к микроконтроллеру, схему алгоритма управления заданным вариантом клавиатуры и программу на Ассемблере управления заданным вариантом клавиатуры.

6.4. Контрольные вопросы

1. Поясните принцип матричного подключения клавиатуры к микроконтроллеру.

2. Назовите задачи, решаемые управляющей программой по обслуживанию клавиатуры.

3. Поясните алгоритмы решения задач по обслуживанию клавиатуры.

4. Поясните схему алгоритма работы программы управления клавиатурой, которая разработана вами в соответствии с полученным заданием.

5. С помощью интегрированной среды MPLAB докажите, что разработанная вами программа функционирует в соответствии с выданным вам заданием.

7. Лабораторная работа «Формирование аналоговых сигналов в микропроцессорном устройстве»

7.1. Цель работы

Изучить метод цифроаналогового преобразования на основе широтно-импульсной модуляции (ШИМ).

Приобрести практические навыки разработки и отладки программ процедур формирования аналоговых сигналов в микропроцессорном устройстве.

7.2. Краткие теоретические сведения

В микропроцессорных устройствах для формирования аналоговых сигналов в виде постоянного напряжения заданного уровня или напряжения, изменяющегося во времени по заданному закону, широкое применение нашел метод формирования на основе широтно-импульсной модуляции.

На рис. 7.1 показан ШИМ-сигнал. Он описывается тремя параметрами: амплитудой E, периодом T и длительностью импульсов t. При этом параметры E и T остаются постоянными, а t изменяется дискретно с точностью $1/2^N$, где N – разрядность кода для представления параметра t.

Принцип цифроаналогового преобразования (ЦАП) на основе ШИМ заключается в выделении с помощью фильтра нижних частот (ФНЧ) постоянной составляющей U_0 ШИМ-сигнала. При этом $U_0 = E\,t / T$.

Частота среза ФНЧ должна выбираться из условия требуемого подавления несущей частоты (1/T) ШИМ-сигнала. Если, к примеру, в качестве ФНЧ используется интегрирующая RC-цепочка с постоянной времени $\tau_{ФНЧ} = RC$, то для обеспечения пульсаций выходного сигнала меньше единицы младшего разряда ЦАП ($E/2^N$) постоянная времени ФНЧ должна выбираться из условия

$$RC \geq T \cdot 2^N.$$

При этом частота среза F_{CP} ФНЧ может быть найдена из выражения

$$F_{CP} = 1/2\pi RC.$$

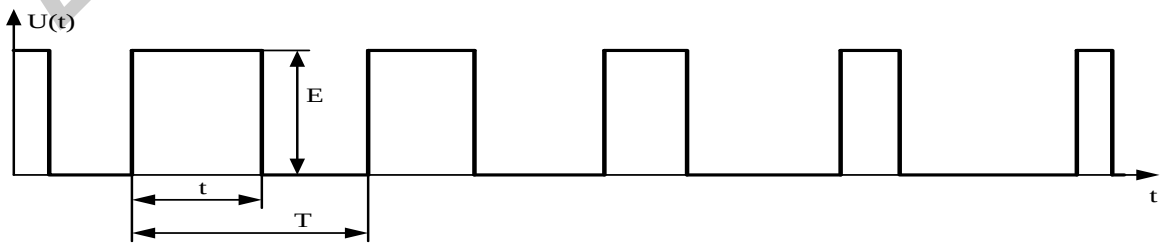


Рис. 7.1. Широтно-импульсно модулированный сигнал

В большинстве моделей PIC контроллеров имеются встроенные на кристалле специальные аппаратные модули генерации ШИМ-сигналов.

С целью более глубокого изучения материала рассмотрим вариант программной реализации метода.

Ниже приведен пример программы для МК PIC16F886, реализующей формирование ШИМ-сигнала, постоянная составляющая которого изменяется во времени по пилообразному закону. Для реализации задержек в программе используется счетчик-таймер TMR0 и задействуется система прерываний.

В основной программе реализуется следующая последовательность действий.

1. Настройка портов ввода/вывода и модуля таймера-счётчика TMR0.
2. Инициализация первоначальных значений переменных.
3. Настройка системы прерываний, разрешение прерывания от TMR0.
4. Ожидание прерывания по переполнению TMR0 в бесконечном цикле. TMR0 задаёт период повторения ШИМ-сигнала.

В подпрограмме обработки прерываний для формирования ШИМ-сигнала используются три переменные:

PWM_Per – для задания периода ШИМ,

PWM_Pulse – для задания длительности импульса ШИМ,

PWMRep – для задания количества повторений текущего периода.

Для формирования одного периода ШИМ-сигнала в программе используется 255 периодов переполнения TMR0. При этом в каждом периоде переполнения TMR0 выполняются следующие действия:

1. Сохранение контекста (состояния общих регистров).
2. Корректировка начального состояния TMR0 и сброс флага переполнения таймера.
3. Формирование периода ШИМ через цепочку операций:

PWM_Per = PWM_Per + 1;

если PWM_Per = 0, то вывод на порт «1»;

если PWM_Per = PWM_Pulse, вывод на порт «0».

4. Проверка, закончено ли формирование текущего периода ШИМ, т.е. PWM_Per = 255? Если да, то переход на п. 5. Если нет, то восстановление контекста и возврат из подпрограммы обработки прерываний.
5. Вычитание единицы из счётчика повторений периода ШИМ PWMRep.

Если PWMRep не равно нулю, то восстановление контекста и возврат из подпрограммы обработки прерываний.

Если PWMRep = 0, то переход на п. 6.

6. Загрузка нового значения длительности импульса ШИМ PWM_Pulse.
7. Восстановление контекста и выход из подпрограммы обработки прерываний.

```
;*****************************************************************
;   Программа формирования пилообразного напряжения
;*****************************************************************
        list            p=16f886            ; Выбираем тип контроллера
        #include        <p16f886.inc>       ; Подключаем файл с описанием имён
                                            ; регистров и битов используемого
                                            ; микроконтроллера
        errorlevel      -302    ; Отключаем вывод сообщений о некорректном банке памяти
; Формируем слово конфигурации микроконтроллера
        __CONFIG    _CONFIG1, _LVP_OFF & _FCMEN_ON & _IESO_OFF & _BOR_OFF
& _CPD_OFF & _CP_OFF & _MCLRE_ON & _PWRTE_ON & _WDT_OFF & _HS_OSC
        __CONFIG    _CONFIG2, _WRT_OFF & _BOR21V
; Задаём адреса регистров для хранения переменных при сохранении  контекста
; прерывания в ОЗУ быстрого доступа
W_TEMP          EQU   0x7D          ; Переменная для регистра W
STATUS_TEMP     EQU   0x7E          ; Переменная для регистра STATUS
PCLATH_TEMP     EQU   0x7F          ; Переменная для регистра PCLATH
; Задаём пользовательские константы
T       EQU   .240              ; Начальное значение TMR0
K       EQU   .20               ; Количество повторений периода ШИМ
        Cblock          0x20
PWM_Per                         ; Ячейка для счёта периода ШИМ
PWM_Pulse                       ; Ячейка для счёта длительности импульса
PWMP                            ; Ячейка для счёта длительности импульса
PWMRep                          ; Ячейка для счёта повторения периода ШИМ
        endc
        #define         OUT         PORTA,5     ; Определяем замену текста
;*****************************************************************
;       Начало текста программы
        ORG   0x000             ; Установить адрес начала программы (вектор сброса)
        goto    main            ; Перейти на начало основной программы

;       Начало подпрограммы обработки прерываний
        ORG   0x004             ; Установить адрес начала подпрограммы
                                ; обработки прерываний (вектор прерывания)
interrupt:                      ; Метка начала подпрограммы обработки  прерываний
    ; Сохраняем контекст
        movwf    W_TEMP         ; Сохраняем текущее значение регистра W
        movf     STATUS,W       ; Пересылаем содержимое регистра STATUS в W
        movwf    STATUS_TEMP    ; Сохраняем текущее значение регистра STATUS
        movf     PCLATH,W       ; Пересылаем содержимое регистра PCLATH в W
        movwf    PCLATH_TEMP    ; Сохраняем текущее значение регистра PCLATH
; Сохранены значения регистров, теперь можно обрабатывать прерывание
; Прерывание только от TMR0, поэтому реагируем только на него
        banksel         TMR0        ; Выбрать банк памяти
```

```
        movlw   T                       ; Корректируем начальное значение TMR0
        movwf   TMR0                    ;
        bcf     INTCON,T0IF             ; Сбрасываем флаг прерывания по TMR0
        banksel  PORTA                  ; Выбрать банк памяти
PWM_run:                                ; Генерируем импульс
        incfsz  PWM_Per,F               ; Прибавляем 1 к периоду ШИМ
        goto    $+2                     ; Продолжаем работу
        bsf     PORTA,5                 ; Устанавливаем 1 на выводе порта, начинаем
                                        ; формирование ШИМ
        movf    PWM_Pulse,W     ; Заносим значение длительности импульса в WREG
        subwf   PWM_Per,W               ; Вычитаем текущее значение периода ШИМ
        btfsc   STATUS,Z                ; Проверяем на нулевой результат
        bcf     PORTA,5                 ; Если да, тогда сбрасываем в 0 вывод порта
;Сгенерировали импульс
now_pulse:      ;. Формируем повторение цикла ШИМ
        movlw   .255            ; Проверяем, закончен ли текущий период ШИМ
        subwf   PWM_Per,W       ;
        btfss   STATUS,Z        ;
        goto    int_return      ; Не закончен, поэтому выходим из прерывания
        decfsz  PWMRep,F        ; Декремент счётчика повторения текущего импульса
        goto    int_return      ; Если не 0, то выходим из прерывания
        movlw   K               ; Восстанавливаем число повторений для следующего
        movwf   PWMRep          ; периода ШИМ
; Здесь надо менять значения длительности импульса ШИМ
next_pulse:
        incf    PWM_Pulse,F
int_return:     ; Обработали прерывание по TMR0. Восстанавливаем контекст
        movf    PCLATH_TEMP,W   ; Восстанавливаем значение PCLATH в W
        movwf   PCLATH          ; Восстанавливаем значение PCLATH
        movf    STATUS_TEMP,W   ; Восстанавливаем значение STATUS в W
        movwf   STATUS          ; Восстанавливаем значение STATUS
        swapf   W_TEMP,F        ; Восстанавливаем значение W
        swapf    W_TEMP,W
        retfie  ; Возвращаемся из подпрограммы обработки прерываний
;*****************************************************************
; Начало текста основной программы
main:                                   ; Метка начала основной программы
        call    init_port               ; Инициализация портов ввода/вывода
        call    init_TMR0               ; Инициализация TMR0
        banksel PWM_Per
        clrf    PWM_Per
        clrf    PWM_Pulse
        clrf    PWMRep
        banksel TMR0            ; Выбираем банк памяти
```

```
        movlw T                 ; Корректируем начальное значение TMR0
        movwf       TMR0
        bcf     INTCON,T0IF     ; Сбрасываем флаг прерывания по таймеру TMR0
        bsf     INTCON,T0IE     ; Разрешаем прерывания по таймеру TMR0
        bsf     INTCON,PEIE     ; Разрешаем прерывания от периферийных модулей
        bsf     INTCON,GIE      ; Разрешаем все немаскированные прерывания
                                ; в бесконечном цикле формируем сигнал
        movlw       .255
        movwf       PWM_Per
        movlw       .0
        movwf       PWM_Pulse
        movlw       K
        movwf       PWMRep
        goto    $               ; Бесконечный цикл. Ждём прерывания от TMR0
;*****************************************************************
; Подпрограммы
; Подпрограмма инициализации портов ввода/вывода
init_port:                      ; Метка начала подпрограммы
        banksel     PORTA       ; Директива выбора банка памяти, где
                                ; расположен указанный регистр
        clrf        PORTA       ; Инициализировать PORTA
        banksel      ANSEL      ; Выбрать банк памяти.
        Clrf        ANSEL       ; Очистить регистр ANSEL, установить
                                ; порты ввода/вывода как цифровые
        banksel     TRISA       ; Выбрать банк памяти
        bcf         TRISA,5     ; Сбросить бит 5 TRISA, настроить
                                ; вывод PORTA RA5 на выход
        banksel     PORTA       ; Выбрать банк памяти
        return                  ; Возврат из подпрограммы

; Подпрограмма настройки TMR0
init_TMR0:                          ; Метка начала подпрограммы
        banksel TMR0                ; Выбрать банк памяти
        clrf    TMR0                ; Инициализировать TMR0
        banksel OPTION_REG          ; Выбрать банк памяти
        movlw   b'11000010'         ; Включаем тактирование от внутреннего
        movwf   OPTION_REG          ; генератора Fosc/4, предделитель перед
                                    ; TMR0, коэффициент деления 1:256
        return                      ; Возврат из подпрограммы
;*****************************************************************
        END         ; Конец текста программы
```

7.3. Лабораторное задание

1. Изучить теоретический материал, изложенный в подразд. 7.2.

2. Загрузить приведенную в подразд. 7.2 программу в микроконтроллер PIC16F886, установленный на макетную плату, и проверить ее работу, визуально контролируя генерируемый сигнал с помощью цифрового осциллографа на выводе RA5.

3. Получить у преподавателя задание на модификацию исследуемой программы. Модифицировать форму генерируемого сигнала, амплитуду, длительность, частоту, выходные порты микроконтроллера.

4. Выполнить модификацию программы в соответствии с полученным заданием, отладить программу, используя среду MPLab IDE и макетную плату. Снять осциллограмму генерируемого сигнала.

5. Оформить отчет, который должен содержать: титульный лист, цель работы, вариант лабораторного задания, схему макета, листинг отлаженной программы, осциллограмму сгенерированного сигнала, выводы по работе.

7.4. Контрольные вопросы

1. Поясните принцип цифроаналогового преобразования на основе широтно-импульсной модуляции.

2. Из каких соображений выбирается частота среза фильтра нижних частот для выделения постоянной составляющей ШИМ-сигнала?

3. Поясните алгоритм работы модифицированной в соответствии с индивидуальным заданием программы.

4. Прокомментируйте полученные экспериментально осциллограммы сгенерированного сигнала.

ЛИТЕРАТУРА

1. PIC16F882/883/884/886/887. – Data Sheet. Microchip technology incorporated, 2006. [Электрон. ресурс]. – Режим доступа: http:// www.microchip.ru.

2. Казека, А. А. Разработка программ для микроконтроллеров фирмы MICROCHIP в интегрированной среде MPLAB IDE : метод. указание к лаб. работам по курсу «Микропроцессорные устройства» для студ. радиотехнич. спец. всех форм обуч. / А. А. Казека, А. В. Мартинович, И. Г. Давыдов. – Минск : БГУИР, 2008.

3. Левкович, В. Н. Конструирование программ на Ассемблере для микроконтроллеров семейства PICmicro: учеб. пособие по курсу «Цифровые и микропроцессорные устройства» для студ. спец. 39 01 01 «Радиотехника» и 39 01 02 «Радиоэлектронные системы» всех форм обуч./ В. Н. Левкович. – Минск : БГУИР, 2004.

4. PIC16F62X. Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip technology incorporated : пер. с англ. – М. : ООО «Микрочип», 2001. [Электрон. ресурс]. – Режим доступа : http:// www.microchip.ru.

Приложение 1

Директивы Ассемблера

ДИРЕКТИВА	ОПИСАНИЕ	ПРИМЕР
Директивы управления		
CONSTANT	Определение символьной константы	constant cnt=255
#DEFINE	Определение текстовой последовательности для замены	#define snd portsnd, 1
END	Конец блока программы	end
EQU	Определение константы	temp equ 0xF0
ERROR	Сообщение об ошибке	error "error line"
ERROR LEVEL	Установка типа сообщений об ошибках в файле листинга и файле ошибок	errorlevel 1, -202
INCLUDE	Вставить другой файл источника	include <addmain.asm>
LIST	Определение формата (тип микроконтроллера, количество символов в строке, табуляция и др.)	list p=16f628A, f=INHX32, r=DEC
MESSG	Создать пользовательское сообщение	messg "see here!"
NOLIST	Запретить вывод	nolist
ORG	Установить начальный адрес программы	org 0x100
PAGE	Вставить страницу в файл листинга	page
PROCESSOR	Установить тип микроконтроллера	processor 16F628A
RADIX	Установить систему счисления по умолчанию для выражения данных	radix dec
SET	Определение константы. Аналогична EQU, но впоследствии можно переопределить	temp set b'00110011'
SPACE	Вставить пустые строки в файл листинга	space 3
SUBTITLE	Вставить второй заголовок в файл листинга	subtitle "Main Project"
TITLE	Вставить заголовок в файл листинга	title "Project Of PIC"
#UNDEFINE	Удаление определенной текстовой последовательности	#undefine snd
VARIABLE	Определение символьной переменной	variable temp=0xF0
Условия		
ELSE	Начало блока альтернативного условия (IF)	else
ENDIF	Завершение блока условия	endif
ENDW	Завершение цикла ПОКА	endw
IF	Начало блока условия	if version == 100
IFDEF	Выполнить, если определено	ifdef testing
IFNDEF	Выполнить, если не определено	ifndef testing
WHILE	Цикл ПОКА	while i < count

Окончание прил. 1

ДИРЕКТИВА	ОПИСАНИЕ	ПРИМЕР
Данные		
CBLOCK	Определение блока констант	cblock 0x20
__CONFIG	Описание бит конфигурации микро-контроллера	__config H'FFFF'
DATA	Создание числовых и текстовых данных	txt data "please", 0x30
DB	Определение байта данных	temp db 0xFF
DE	Определение данных в EEPROM	temp de 0xF0, 0xF1
DT	Определение таблицы	temp dt "text", 0, 0x30
DW	Определение слова (два байта) данных	temp dw 39, "text"
ENDC	Окончание блока констант	endc
FILL	Заполнение области константой	fill 0x1009, 5
__IDLOCS	Определение ID	__idlocs H'FFEE'
RES	Резервирование памяти	buffer res 64
Макросы		
ENDM	Окончание макроса	endm
EXITM	Выход из макроса	exitm
EXPAND	Полный текст макроса в файле листинга	expand
LOCAL	Определение локальной переменной в макросе	local leng, tmp
MACRO	Определение макроса	out_sym macro temp
BANKSEL	Выбор банка для прямой адресации	Banksel registr

Приложение 2

Арифметические операторы Ассемблера

Оператор	Описание	Пример
$	Текущий счетчик программы	goto $ + 3
(	Левая скобка	1 + (d * 4)
)	Правая скобка	(leght + 1) * 255
!	Операция "НЕ" (логическая инверсия)	if ! (a - b)
~	Инверсия	flags = ~ flags
-	Отрицательное число (вторая инверсия)	- 1 * lenght
high	Выделить старший байт слова	movlw high llasid
low	Выделить младший байт слова	movlw low (llasid + .2551)
*	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

Окончание прил. 2

Оператор	Описание	Пример
>>	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 "ВКЛЮЧАЮЩЕЕ ИЛИ"	flags = flags | err_bit
&&	Логическое "И"	if (len == 512) && (b == c)
||	Логическое "ИЛИ"	if (len == 512) || (b == c)
=	Установить равному...	entry_index = 0
+=	Сложить и установить равному...	entry_index += 1
-=	Вычесть и установить равному...	entry_index -= 1
*=	Умножить и установить равному...	entry_index *= lenght
/=	Делить и установить равному...	entry_index /= lenght
%=	Модуль и установить равному...	entry_index %= 8
<<=	Сдвиг влево и установить равному...	entry_index << 3
>>=	Сдвиг вправо и установить равному...	entry_index >> 4
&=	"И" и установить равному...	entry_index %= err_flags
|=	"ВКЛЮЧАЮЩЕЕ ИЛИ" и установить равному...	entry_index |= err_flags
^=	"ИСКЛЮЧАЮЩЕЕ ИЛИ" и установить равному...	entry_index ^= err_flags
++	Увеличить на 1 (инкремент)	i ++
--	Уменьшить на 1 (декремент)	i --

Приложение 3

Форматы представления чисел

ФОРМАТ	СИНТАКСИС	ПРИМЕР
Десятичный	D'число' .число	D'100' .100
Шестнадцатеричный	H'число' 0хчисло	H'f9' 0xAF00
Восьмеричный	O'число'	O'777'
Двоичный	B'число'	B'11110000'
Символьный	'символ' A'символ'	'C' A'C'

Приложение 4

Стандартные расширения для файлов MPLAB

РАСШИРЕНИЕ	НАЗНАЧЕНИЕ ФАЙЛА
*.ASM	Исходный файл на Ассемблере
*.C	Исходный файл на С
*.CFG	Файл конфигурации
*.COD	Содержит символьную информацию и объектный код
*.CSV	Файл с записью трассировки (только для MPLAB-ICE 2000)
*.DAT	Файл данных симулятора
*.ERR	Файл обнаруженных ошибок, генерируется ассемблером или С при компиляции
*.H	Добавленный файл на С
*.HEX	Файл с машинными кодами в HEX-формате для PIC-микроконтроллеров
*.HLP	Файл помощи
*.INC	Добавленный файл на Ассемблере
*.INI	Конфигурация MPLAB и установленного языка программирования
*.KEY	Файл схемы кнопок MPLAB
*.LKR	Файл сценария компоновки MPLINK
*.LST	Абсолютный листинг, генерируется ассемблером или С при компиляции
*.MTC	Файл конфигурации языка программирования
*.PJT	Файл содержит главную информацию о проекте
*.REG	Файл, описывающий модификацию регистров при отладке
*.STI	Файл, описывающий входные сигналы на входах микроконтроллера
*.TB	Файл трассировки, точек остановки
*.TBR	Файл панели инструментов
*.TPL	Временный файл
*.TRC	Файл записи трассировки
*.TXT	Файл записи трассировки (только MPLAB-ICE 2000)
*.WAT	Файл окна просмотра

Учебное издание

Левкович Василий Николаевич
Каленкович Евгений Николаевич
Казека Александр Анатольевич

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

Редактор *Т. П. Андрейченко*
Корректор *Е. Н. Батурчик*
Компьютерная верстка *Ю. Ч. Клочкевич*

Подписано в печать 15.12.2011.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л. 5,46.
Уч.-изд. л. 5,5.	Тираж 120 экз.	Заказ 223.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6