

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ВЫСШИЙ КОЛЛЕДЖ ИНФОРМАТИКИ  
КАФЕДРА КОМПЬЮТЕРНЫХ СИСТЕМ

Ю. М. Прокопьев

**Архитектура и проектирование  
микроконтроллеров**

**Практикум по проектированию микроконтроллеров  
на примере компонентов фирмы  
Microchip Technology**

*Учебное пособие*

Новосибирск  
1999

ББК 3973.2-02  
УДК 621.396.001.66

Прокопьев, Ю. М. Архитектура и проектирование микроконтроллеров. Практикум по проектированию микроконтроллеров на примере компонентов фирмы Microchip Technology: Учебное пособие / Новосибирск: Новосиб. ун-т, 1999. 80 с.

В данном учебном пособии содержатся материалы для практической части курса "Архитектура микроконтроллеров".

Пособие предназначено для студентов физического факультета НГУ и ВКИ.

Рецензент профессор В. И. Нифонтов

БИБЛИОТЕКА  
НГУ

© Новосибирский государственный университет, 1999

## Введение

В этой книге на примере микроконтроллеров фирмы Microchip Technology рассматривается архитектура, а также основные приёмы, аппаратные и программные средства разработки программного обеспечения.

Выбор именно этих микроконтроллеров был сделан не случайно. На него повлияло множество факторов, из которых следует выделить следующие:

- большая популярность и доступность,
- низкая стоимость,
- простота архитектуры и системы команд,
- необычайно широкая номенклатура семейств,
- удобный интерфейс для программирования микроконтроллеров,
- полная и дружественная информационная поддержка фирм-изготовителя,
- свободное распространение фирменных программных средств разработки.

Микроконтроллеры фирмы Microchip Technology представляют определенный интерес там, где не требуется большой вычислительной мощности и сложного управления. Особенно эффективно применение этих микросхем в том случае, когда разрабатываемая система хорошо разделяется на независимые части с минимумом обмена данными между ними. Почти все семейства контроллеров имеют в своем составе последовательные интерфейсы, с помощью которых очень удобно организовывать необходимый обмен информацией.

Важной особенностью этих микроконтроллеров является использование в них гарвардской архитектуры (т. е. раздельный доступ к командам и данным). В результате существенно упрощается система команд и программный код более компактен по сравнению с традиционным построением системы команд процессора.

В большинстве микросхем этих семейств не предусматривается возможность использования внешней программной памяти. Это позволяет применять для них очень компактные корпуса, с небольшим числом выводов. А так как стоимость этих микросхем невелика, то это всегда можно скомпенсировать использованием нескольких кристаллов, соединяя их между собой с помощью последовательного интерфейса.

Широкая номенклатура семейств микроконтроллеров, разнообразие внешних устройств, присутствующих в каждом микроконтроллере внутри

каждого семейства и иерархическое построение ряда предлагаемых контроллеров позволяет очень точно подобрать конкретную микросхему (или набор микросхем) для какой-либо разработки, оптимальный как с точки зрения доступных аппаратных и программных ресурсов, так и стоимости.

Большинство микросхем используют тактовую частоту до 20-25 МГц. Команды выполняются за 4 такта. Причем команды управления программой устроены так, что все ветвления требуют одинакового времени для их выполнения. Соблюдение этого условия может оказаться полезным в программах, требующих точного соблюдения временных соотношений.

Основные параметры некоторых микроконтроллеров приведены в таблице:

Тип микроконтроллера	Максимальная частота генератора (МГц)	Программная память	Память данных	Число выводов корпуса
PIC14000	20	4 К	192	28
PIC12C5x	4	512, 1 К	25, 41	8
PIC16C5x	4-20	384 - 2 К	24 - 73	18 - 28
PIC16C6x	20	512 - 4 К	80 - 192	18 - 44
PIC16C7x	20	512 - 4 К	36 - 192	18 - 44
PIC16C8x	10	512, 1 К	36, 68	18
PIC17C4x	25	2 К - 8 К	232 - 454	40, 44

Разрядность командных слов составляет 12-16 бит для семейств различной степени сложности. Все микроконтроллеры работают с 8-разрядными данными.

Большинство семейств микроконтроллеров представлено в двух вариантах исполнения программной памяти. Обычные приборы позволяют произвести однократное программирование, а для отработки макетного или первого пробного варианта можно использовать микросхемы с памятью с ультрафиолетовым стиранием.

Некоторые семейства имеют в своем составе приборы с электрически перепрограммируемой программной памятью.

Для программирования микроконтроллеров используется последовательный интерфейс, а все программирующее оборудование сосредоточено внутри микросхемы.

Простота подключения программирующего устройства удачно дополняется возможностью так встраивать микроконтроллер в разрабатываемую систему, что для его программирования не требуется вынимать сам микропроцессор и вставлять в программатор. Надо просто предусмотреть отдельный 5-штырьковый разъем для внешнего подключения программатора.

Как и у большинства семейств микроконтроллеров других фирм, имеется богатый набор периферийных устройств, состав которых может сильно изменяться от семейства к семейству и внутри семейства. В одном контроллере, например, можно встретить сразу все три варианта последовательных портов (SPI, I2C, USART). Многие микроконтроллеры имеют средства программно-аппаратной реализации широтно-импульсной модуляции и разнообразные таймерные секции.

Есть также очень оригинальные изделия, как, например, контроллер PIC12C5x в восьмивыводном корпусе, интересный тем, имеет внутренний тактовый генератор и состоит фактически из таймерной секции и нескольких контактов для ввода/вывода. При этом в нем есть стандартный для всех семейств набор устройств типа WatchDog, схемы начального пуска, контроля питания и тактового генератора.

Некоторые параметры микроконтроллеров, относящиеся к внешнему интерфейсу, приведены в таблице:

Тип микроконтроллера	Число таймерных модулей	Последовательные порты	Число контактов ввода/вывода	АЦП (разрядность, каналы)
PIC14000	1	SPI, I2C	22	16, 8
PIC12C5x	1	-	1	-
PIC16C5x	1	-	12, 20	-
PIC16C6x	1- 3	SPI, I2C, USART	13 - 33	-
PIC16C7x	1 - 3	SPI, I2C, USART	13 - 33	8, 4-8
PIC16C8x	1	-	13	-
PIC17C4x	4	USART	33	-

Для работы с аналоговыми сигналами можно использовать контроллеры, в составе которых есть компараторы и аналогово-цифровые преобразователи. Обычно АЦП в этих микросхемах имеет 8-10 разрядов и частоту дискретизации до 50 кГц.

Для специальных применений может оказаться полезным контроллер PIC14000, в состав которого входит 16-разрядный АЦП, а точность и время преобразования можно изменять программно. В этом контроллере есть встроенный датчик температуры.

Более производительные разновидности микроконтроллеров могут использовать внешнюю программную память до 64 Кбайт и имеют аппаратный умножитель 8x8 (время выполнения операции 160 нс).

Важным компонентом навыков разработчика является умение пользоваться справочной технической литературой. Поэтому, кроме этого пособия, для более полного понимания принципов работы микроконтроллера

совершенно обязательно использование описаний его внутренней архитектуры и системы команд. Часть этой информации имеется в печатном виде и на русском языке. Однако описания специфических устройств, микросхем и т. д. имеются только в электронном виде (в PDF-формате) и доступны с помощью программы Acrobat Reader.

Фирма Microchip Technology имеет свой WWW-сервер (<http://www.microchip.com/>), на котором можно найти исчерпывающую информацию обо всем спектре продукции, технических параметрах и о средствах разработки.

Для знакомства с такой литературой желательно знание технического английского.

Вместе с изучением системы команд и архитектуры самих микроконтроллеров осваивается и используемый для разработки и отладки программного обеспечения пакет MPLab, свободно распространяемый фирмой-производителем.

Для начала работы не требуется наличие каких-либо навыков работы с микропроцессорами. Если таковые уже имеются, то некоторые задания могут быть, по усмотрению преподавателя, опущены.

Поскольку весь комплект программного обеспечения рассчитан на операционную среду Windows 95, то, разумеется, желательно наличие хотя бы небольшого опыта работы с ней (запуск программ, редактирование текстовых файлов и пр.).

К моменту написания этого пособия разработана учебно-демонстрационная плата для микроконтроллеров PIC16C8x. В дальнейшем предполагается расширить номенклатуру контроллеров и реализуемых устройств, дополнив их возможностью работы с аналоговыми сигналами, различными вариантами последовательных интерфейсов и средств управления.

Для успешного практического выполнения даже не очень сложных заданий нужно хорошо представлять структуру микроконтроллера, систему команд, режимы работы и правила программирования. Все необходимые сведения представлены в первом разделе. Более подробную информацию можно найти в [1] и в фирменных руководствах на интересующий микроконтроллер.

Следующие две главы посвящены описанию среды разработки MPLab и кросс-ассемблера. Это основной программный инструментарий разработчика. Его также можно (и полезно) использовать для изучения особенностей какого-либо семейства микроконтроллеров, даже не имея таких под руками.

Далее следует небольшая инструкция для работы с программатором.

Фактически все эти разделы являются вступительными к практической работе. Однако без тщательного их изучения выполнение самих заданий представляется невозможным.

Следующим шагом должно стать ясное понимание назначения и принципов работы разрабатываемого устройства или прибора. Без этого невозможно мобилизовать и правильно распределить все доступные ресурсы используемого микроконтроллера. Поэтому рекомендуется, прежде чем начинать разрабатывать схему и писать программу, точно очертить круг решаемых устройством задач, внешний интерфейс для связи с датчиками и другим оборудованием, варианты распределения ресурсов, алгоритмы программной реализации.

Тем, кого не испугали все эти наставления, желаю успехов в своих разработках.

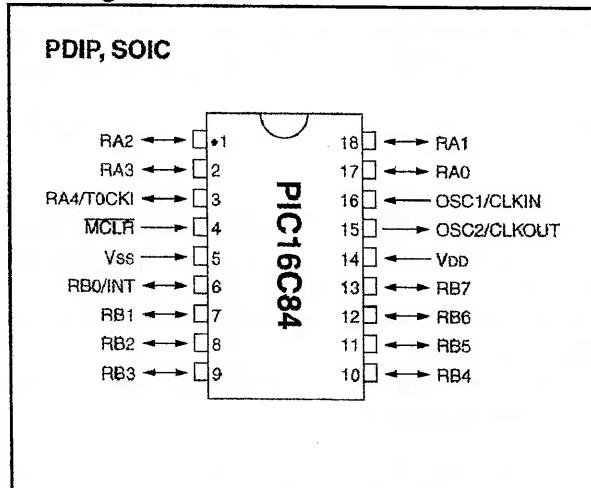
Автор выражает благодарность профессору Нифонтову В. И. и заведующему лабораторией кафедры АФТИ Феофанову А. В. за полезные замечания по содержанию пособия, а также заведующему лабораторией ВКИ НГУ Лукошенко А. Л. и студентам технического факультета НГУ за практическую помощь в создании практикума.

Хотелось бы также отметить поддержку подготовки к выпуску данного пособия программой "Интеграция" (проект 274).

## Архитектура микроконтроллеров PIC16C84 (PIC16F84)

Рассмотрим более подробно микроконтроллер PIC16C84, который обладает большинством свойств, присущих всем семействам PIC-контроллеров фирмы Microchip Technology.

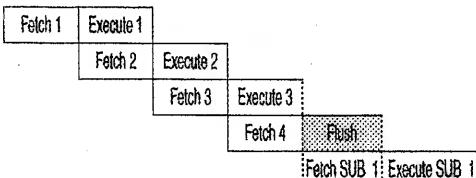
### Pin Diagram



Как уже отмечалось, во всех микроконтроллерах используется гарвардская архитектура, в которой доступ к программам и данным происходит с использованием разных магистралей. Такое разделение позволяет иметь разрядность команд, отличающуюся от 8-разрядных слов данных. В описываемом микроконтроллере

используется 14 бит для кодов команд. Выборка каждого слова команды производится за один цикл, ее выполнение происходит в следующем цикле, а в это время происходит выборка следующей команды. Таким образом, создается двухступенчатый поток.

MOVlw	55h	
Movwf	PortB	Fetch 1 Execute 1
Call	SUB_1	Fetch 2 Execute 2
BSF	PortA, BIT3	Fetch 3 Execute 3



Один цикл работы микроконтроллера состоит из четырех тактов системной частоты синхронизации. Для 20 МГц (один такт – 50 нс) каждая

команда выполняется за 200 нс. Единственным исключением являются команды ветвления, для которых необходимо 2 цикла.

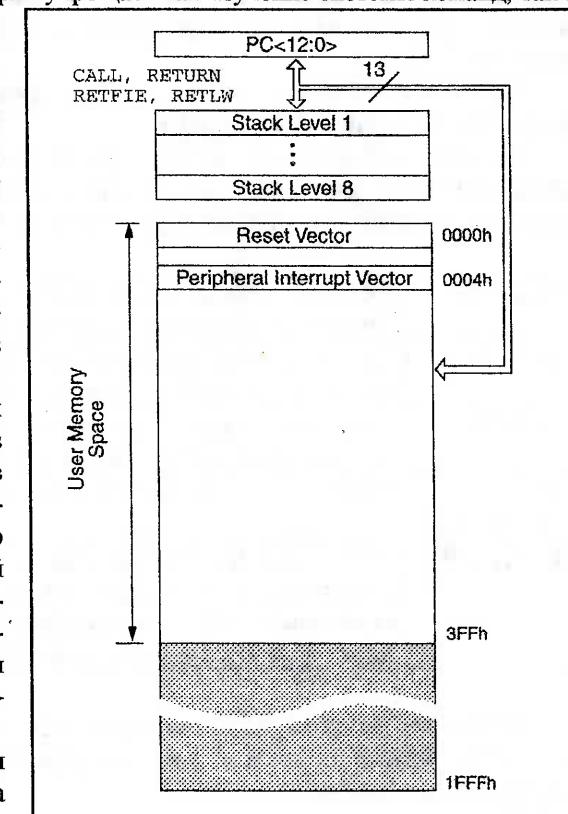
Микроконтроллер имеет 1Kx14 разрядов программной памяти и 36 байт памяти данных (68 для PIC16F84), может использовать прямую и косвенную адресацию для доступа к регистрам общего и специального назначения. Все регистры управления, включая счетчик команд, отображаются в области памяти данных.

Контроллеры имеют ортогональную систему команд, что позволяет использовать любую операцию для любого регистра с любым режимом адресации. Такая симметрия упрощает как изучение системы команд, так и программирование микроконтроллеров.

В контроллере находится 8-разрядное АЛУ и один рабочий регистр W. АЛУ может выполнять арифметические и логические операции, используя данные, хранящиеся в рабочем регистре и любом из файловых регистров. Для двухоперандных команд обычно одним из используемых регистров является рабочий регистр (рабочий регистр W – неявно адресуемый регистр.). Другим – регистры общего и специального назначения или непосредственный operand.

Результат операции в АЛУ может влиять на бит переноса C, цифровой перенос DC и флаг нуля Z в регистре состояния (STATUS).

Семейство PIC16C8x имеет 13-разрядный счетчик команд, что позволяет адресовать программную память размером 8Kx14. Размер реальной физической памяти различный в разных модификациях. Обращение к па-



мяти с адресом большим, чем имеется физической памяти, приводит к циклическому сворачиванию адреса, т. е. происходит доступ к тому же адресному полю. Вектор прерывания по сбросу имеет адрес 0000h, вектор всех остальных прерываний 0004h. Аппаратный стек допускает 8 уровней вложенности подпрограмм и прерываний.

Регистры общего и специального назначения разбиты на два банка размером по 48 (80) байт каждый. Начальные адреса обоих банков заняты регистрами специального назначения, а далее располагаются регистры общего назначения. Некоторые наиболее часто используемые специальные регистры для уменьшения размера программ и ускорения доступа отображаются в оба банка.

Доступ к регистрам осуществляется непосредственно или косвенно через регистр FSR (**F**ile **S**elected **R**egister).

Регистры специального назначения используются для управления функциями микроконтроллера. Их можно условно разбить на две группы (ядро и периферия). К регистрам ядра относятся следующие регистры.

STATUS	содержит биты результатов операций в АЛУ, бит выбора банка памяти данных и флаги состояния после сброса
OPTION	Содержит биты конфигурации таймера WatchDog, внешнего прерывания INT, таймера 0 и портом B
INTCON	Содержат флаги и биты управления прерываниями от различных периферийных устройств
PCL, PCLATH	Младший и старший байт счетчика команд
INDF, FSR	INDF - это фиктивный регистр, обращение к которому реализует косвенную адресацию. Любая инструкция, использующая INDF, реально будет обращаться к операнду, на который указывает регистр FSR

Остальные регистры тесно связаны с работой портов ввода/вывода и служат для задания режимов их работы и управления во время исполнения программы.

Более подробную информацию об аппаратной архитектуре микроконтроллера и его электрических характеристиках можно найти в [1].

Пример использования микроконтроллера приведен в приложении, где представлена полная схема учебно-демонстрационной платы.

## Описание системы команд

Каждая команда PIC16C8X представляет собой 14-разрядное слово, содержащее поле кода операции OPCODE и поле операндов. Система команд PIC16C8X приведена в таблице. Система команд включает в себя команды работы с байтами, команды работы с битами, команды управления и операции с константами.

## Описание полей команд

Поле	Описание
f	Адрес
w	Рабочий регистр
b	Номер бита в 8-разрядном регистре
k	Константа
x	Не используется. Ассемблер формирует код с x=0
d	Регистр назначения: d=0 - результат в регистре W d=1 - результат в регистре f По умолчанию d=1
label	Имя метки
TOS	Top Of Stack - вершина стека
PC	Program Counter - счетчик команд
TO	Time-Out - тайм-аут
PD	Power Down - выключение питания
dest	Регистр назначения: рабочий регистр W или регистр, заданный в команде
[ ]	Необязательные параметры
( )	Содержание
→	Присвоение
<>	Битовое поле
∈	Из набора

Для команд работы с битами **f** обозначает регистр, с которым производится действие, а бит **d** определяет регистр назначения. При **d=0** результат помещается в регистр **W**, при **d=1** результат помещается в регистр **f**, заданный в команде.

Для команд управления с битами **b** обозначает номер бита, участвующего в команде, а **f** - регистр, в котором он содержится.

Для команд управления и операций с константами **k** обозначает 8- или 11-битовую константу или идентификатор.

Все команды выполняются в течение одного командного цикла, кроме следующих двух случаев:

- переход по проверке условия, если результат проверки условия - истина;
- изменение счетчика команд как результат выполнения команды.

## ОБЩИЙ ФОРМАТ КОМАНД

### Команды работы с байтами:

13	8 7 6	0
OPCODE	d	f(FILE#)

b = 0 для назначения W

b = 1 для назначения f

f = 7-разрядный адрес регистра

### Команды работы с битами:

13	10 9	7 6	0
OPCODE	b(BIT#)	f(FILE#)	

b = 3-разрядный номер бита

f = 7-разрядный адрес регистра

### Команды управления и операции с константами:

13	8 7	0
OPCODE	k (literal)	

k = 8

В этих случаях команда выполняется за два цикла с выполнением второго цикла как NOP. Один командный цикл состоит из четырех периодов генератора. Таким образом, для генератора с частотой 4 МГц время выполнения команды составит 1 мкс. Если выполняется переход по проверке условия или в результате выполнения команды изменится счетчик команд, время выполнения этой команды при тактовой частоте 4 МГц составит 2 мкс.

## Система команд PIC16C8X

Мнемокод	Название команды	Пик-лы	Код команды	Биты сост.	Прим
<b>Команды работы с байтами</b>					
ADDWF f,d	Сложение W и f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF f,d	Логическое И W и f	1	00 0101 dfff ffff	Z	1,2
CLRF f	Сброс регистра f	1	00 0001 1fff ffff	Z	2
CLRW	Сброс регистра W	1	00 0001 0xxx xxxx	Z	
COMF f,d	Инверсия регистра f	1	00 1001 dfff ffff	Z	1,2
DECf f,d	Декремент регистра f	1	00 0011 dfff ffff	Z	1,2
DECFSZ f,d	Декремент f, пропустить команду, если 0	1(2)	00 1011 dfff ffff		1,2,3
INCF f,d	Инкремент регистра f	1	00 1010 dfff ffff	Z	1,2
INCFSZ f,d	Инкремент f, пропустить команду, если 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF f,d	Логическое ИЛИ W и f	1	00 0100 dfff ffff	Z	1,2
MOVE f,d	Пересылка регистра f	1	00 1000 dfff ffff	Z	1,2
MOVWF f	Пересылка W в f	1	00 0000 1fff ffff		
NOP	Холостая команда	1	00 0000 0xxx 0000		
RLF f,d	Сдвиг f влево через перенос	1	00 1101 dfff ffff	C	1,2
RRF f,d	Сдвиг f вправо через перенос	1	00 1100 dfff ffff	C	1,2
SUBWF f,d	Вычитание W из f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF f,d	Обмен местами тетрад в f	1	00 1110 dfff ffff		1,2
XORWF f,d	Исключающее ИЛИ W и f	1	00 0110 dfff ffff	Z	1,2
<b>Команды работы с битами</b>					
BCF f,b	Сброс бита в регистре f	1	01 00bb bfff ffff		1,2
BSF f,b	Установка бита в регистре f	1	01 01bb bfff ffff		1,2
BTFSC f,b	Пропустить команду, если бит в f равен нулю	1(2)	01 10bb bfff ffff		3
BTFSS f,b	Пропустить команду, если бит в f равен единице	1(2)	01 11bb bfff ffff		3
<b>Команды передачи управления и операции с константами</b>					
ADDLW k	Сложение константы и W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW k	Логическое И константы и W	1	11 1001 kkkk kkkk	Z	
CALL k	Вызов подпрограммы	1	00 0kkk kkkk kkkk		
CLRWD	Сброс сторожевого таймера WDT	2	00 0000 0110 0100	TO,PD	
GOTO k	Переход по адресу	1	10 1kkk kkkk kkkk		
IORLW k	Логическое ИЛИ константы и W	1	11 1000 kkkk kkkk	Z	
MOVLW k	Пересылка константы в W	1	11 00xx kkkk kkkk		
RETIE	Возврат из прерывания	2	00 0000 0000 1001		
RETLW k	Возврат из подпрограммы с загрузкой константы в W	2	11 01xx kkkk kkkk		
RETURN	Возврат из подпрограммы	2	00 0000 0000 1000		
SLEEP	Переход в режим SLEEP	1	00 0000 0110 0011	TO,PD	
SUBLW k	Вычитание W из константы	1	11 110x kkkk kkkk	C,DC,Z	
XORLW k	Исключающее ИЛИ константы и W	1	11 1010 kkkk kkkk	Z	

**Примечания:**

- Если модифицируется регистр ввода/вывода (например, MOVF PORT,1), то используется значение, считываемое с выводов. Например, если в выходной защелке порта, включенного на ввод, находится '1', а внешнее устройство формирует на этом выводе '0', то в этом разряде будет записан '0'.
- Если операндом команды является содержимое регистра TMR0 (и, если допустимо, d=1), то предварительный делитель, если он подключен к TMR0, будет сброшен.
- Если в результате выполнения команды изменяется счетчик команд, или выполняется переход по проверке условия, то команда выполняется за два цикла. Второй цикл выполняется как NOP.

**ADDLW**

Add literal to f

Сложение константы и f

Синтаксис: ADDLW k

Операнд:  $0 \leq k \leq 255$ Операция:  $(W)+(k) \rightarrow W$ 

Биты состояния: C, DC, Z

Код: 11 111x kkkk kkkk

Описание: Содержимое регистра W складывается с 8-битовой константой k. Результат сохраняется в регистре W.

Циклов: 1

Пример: ADDLW 0x15

Перед выполнением команды: W=0x10

После выполнения команды: W=0x25

**ADDWF**

Add W and f

Сложение W и f

Синтаксис: ADDWF f,d

Операнды:  $0 \leq f \leq 255, d \in [0,1]$ Операция:  $(W)+(f) \rightarrow (\text{dest})$ 

Биты состояния: C, DC, Z

Код: 00 0111 dfff ffff

Описание: Содержимое регистра W добавляется к содержимому регистра f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f.

Циклов: 1

Пример: ADDWF FSR, 0

Перед выполнением команды: W=0x17 FSR=0xC2

После выполнения команды: W=0xD9 FSR=0xC2

**ANDLW**

AND literal to W

Логическое И константы и W

ANDLW k

 $0 \leq k \leq 255$ (W).AND.(k)  $\rightarrow W$ 

Z

Код: 11 1001 kkkk kkkk

Описание: Содержимое регистра W логически умножается на 8-битовую константу k. Результат сохраняется в регистре W.

Циклов: 1

Пример: ANDLW 0x5F

Перед выполнением команды: W=0xA3

После выполнения команды: W=0x03

**ANDWF**

AND W and f

Логическое W и f

ANDWF f,d

 $0 \leq f \leq 255, d \in [0,1]$ (W).AND.(f)  $\rightarrow (\text{dest})$ 

Z

Код: 00 0101 dfff ffff

Описание: Содержимое регистра W логически умножается на содержимое регистра f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.

Циклов: 1

Пример: ANDWF FSR, 1

Перед выполнением команды: W=0x17 FSR=0xC2

После выполнения команды: W=0x17 FSR=0x02

<b>BCF</b>	Bit Clear f Сброс бита в регистре f Синтаксис: BCF f,b Операнды: 0≤f≤127, 0≤b≤7 Операция: 0→f(b) Биты состояния: Не изменяются Код: 01 00bb bfff ffff Описание: Бит b регистра f сбрасываются в 0 Циклов: 1 Пример: BCF FLAG_REG, 7 Перед выполнением команды: FLAG_REG=0xC7 После выполнения команды: FLAG_REG=0x47	<b>BTFS C</b> Bit Test f, Skip if Clear Пропустить команду, если бит равен нулю Синтаксис: BTFSC f,b Операнды: 0≤f≤127, 0≤b≤7 Операция: Пропустить, если f(b)=0 Биты состояния: Не изменяются Код: 01 10bb bfff ffff Описание: Если бит b регистра f равен 1, выполняется следующая команда. Если бит b регистра f равен 0, следующая команда, считываемая в текущем командном цикле, игнорируется, вместо нее в следующем цикле выполняется NOP, то есть команда выполняется за два цикла. Циклов: 1(2) Пример: HERE BTFSC FLG, 1 FALSE GOTO Process_Code TRUE Перед выполнением команды: PC=адрес HERE После выполнения команды: если FLG=0, PC=адрес True если FLG=1, PC=адрес False
<b>BSF</b>	Bit Set f Установка бита в регистре f Синтаксис: BSF f,b Операнды: 0≤f≤127, 0≤b≤7 Операция: 1→f(b) Биты состояния: Не изменяются Код: 01 01bb bfff ffff Описание: Бит b регистра f устанавливается в 1. Циклов: 1 Пример: BSF FLAG_REG, 7 Перед выполнением команды: FLAG_REG=0x0A После выполнения команды: FLAG_REG=0x8A	<b>BTFS S</b> Bit Test f, Skip if Set Пропустить команду, если бит равен единице Синтаксис: BTFSS f,b Операнды: 0≤f≤127, 0≤b≤7 Операция: Пропустить, если f(b)=1 Биты состояния: Не изменяются Код: 01 11bb bfff ffff Описание: Если бит b регистра f равен 0, выполняется следующая команда. Если бит b регистра f равен 1, следующая команда, считываемая в текущем командном цикле, игнорируется, и вместо нее в следующем цикле выполняется NOP, то есть команда выполняется за два цикла. Циклов: 1(2) Пример: HERE BTFSC FLG, 1 FALSE GOTO Process_code TRUE Перед выполнением команды: PC=адрес HERE После выполнения команды: если FLG=0, PC=адрес False если FLG=1, PC=адрес True

<b>CALL</b>	Call subroutine Вызов подпрограммы	<b>CLRW</b>	Clear W Сброс регистра W
Синтаксис:	CALL k	Синтаксис:	CLRF
Операнд:	$0 \leq k \leq 2047$	Операнд:	Нет
Операция:	$(PC)+1 \rightarrow TOS, \quad k \rightarrow PC<10:0>, (PCLATH<4:3>) \rightarrow PC<12:11>$	Операция:	$00h \rightarrow W, 1 \rightarrow Z$
Биты состояния:	Не изменяются	Биты состояния:	Z
Код:	10 0kkk kkkk kkkk	Код:	00 0001 0xxx xxxx
Описание:	Адрес возврата (PC+1) сохраняется в стеке. Младшие 11 бит адреса загружаются в PC из кода команды. Старшие два бита адреса загружаются в PC из регистра PCLATH<4:3>.	Описание:	Содержимое регистра W сбрасывается в 0 и бит Z устанавливается в 1
Циклов:	2	Циклов:	1
Пример:	HERE CALL THERE	Пример:	CLRW
	Перед выполнением команды: PC = адрес HERE		Перед выполнением команды: W=0x5A
	После выполнения команды: PC = адрес THERE		После выполнения команды: W=0x00, Z=1
	TOS = адрес HERE+1		
<b>CLRF</b>	Clear f Сброс регистра f	<b>CLRWDT</b>	Clear WatchDog Timer Сброс сторожевого таймера WDT
Синтаксис:	CLRF f	Синтаксис:	CLRWDT
Операнд:	$0 \leq f \leq 127$	Операнд:	Нет
Операция:	$00h \rightarrow f, 1 \rightarrow Z$	Операция:	$00h \rightarrow WDT, 0 \rightarrow WDT prescaler, 1 \rightarrow TO, 0 \rightarrow PD$
Биты состояния:	Z	Биты состояния:	TO, PD
Код:	00 0001 1fff ffff	Код:	00 0000 0110 0100
Описание:	Содержимое регистра f сбрасывается в 0 и бит Z устанавливается в 1	Описание:	Содержимое регистра W сбрасывается в 0 и бит Z устанавливается в 1
Циклов:	1	Циклов:	1
Пример:	CLRF G_REG	Пример:	CLRWDT
	Перед выполнением команды: G_REG=0x5A		После выполнения команды: WDT counter=0x00,
	После выполнения команды: G_REG=0x00, Z=1		WDT prescaler=0x00,
			TO=1, PD=1

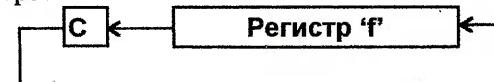
<b>COMF</b>	Complement F Инверсия регистра F Синтаксис: COMF f, d Операнд: $0 \leq f \leq 127, d \in [0,1]$ Операция: $(f) \rightarrow (\text{dest})$ Биты состояния: Z Код: 00 1001 dfff ffff Описание: Содержимое регистра f инвертируется. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f. Циклов: 1 Пример: Перед выполнением команды: REG1=0x13 После выполнения команды: REG1=0x13 W=0xEC	<b>DECFSZ</b> Decrement F, Skip if 0 Декремент F, пропустить команду, если 0 Синтаксис: DECFSZ f,d Операнд: $0 \leq f \leq 127, d \in [0,1]$ Операция: $(f)-1 \rightarrow (\text{dest}) ; \text{пропустить, если } (\text{dest})=0$ Биты состояния: не изменяются Код: 00 1011 dfff ffff Описание: Регистр f уменьшается на 1. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f. Если результат не равен 0, выполняется следующая команда. Если результат равен 0, следующая команда, считанная в текущем командном цикле, игнорируется, и вместо нее в следующем цикле выполняется NOP, в результате команда выполняется за 2 цикла. Циклов: 1(2) Пример: HERE DECFSZ CNT, 1 GOTO LOOP CONTINUE Перед выполнением коман- PC= адрес HERE ды: CNT=CNT-1 После выполнения коман- если CNT=0, PC= адрес Continue ды: если CNT≠0, PC= адрес HERE+1
<b>DECF</b>	Decrement F Декремент регистра F Синтаксис: DECF f,d Операнд: $0 \leq f \leq 127, d \in [0,1]$ Операция: $(f)-1 \rightarrow (\text{dest})$ Биты состояния: Z Код: 00 0011 dfff ffff Описание: Регистр f уменьшается на 1. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f. Циклов: 1 Пример: DECF CNT, 1 Перед выполнением команды: CNT=0x01 Z=0 После выполнения команды: CNT=0x00 Z=1	<b>GOTO</b> Goto address Переход по адресу Синтаксис: GOTO k Операнд: $0 \leq k \leq 2047$ Операция: $k \rightarrow \text{PC}\langle 10:0 \rangle, (\text{PCLATH}\langle 4:3 \rangle) \rightarrow \text{PC}\langle 12:11 \rangle$ Биты состояния: не изменяются Код: 10 1kkk kkkk kkkk Описание: Младшие 11 бит адреса загружаются в PC из кода команды. Старшие два бита адреса загружаются в PC из регистра PCLATH<4:3> Циклов: 2 Пример: HERE GOTO THERE Перед выполнением команды: PC= адрес HERE После выполнения команды: PC= адрес THERE

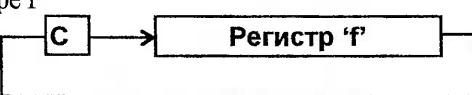
<b>INCF</b>	Increment F Инкремент регистра F	<b>IORLW</b>	Inclusive OR literal to W Логическое ИЛИ константы и W
Синтаксис:	INCF f, d	Синтаксис:	IORLW k
Операнд:	$0 \leq f \leq 127, d \in [0,1]$	Операнд:	$0 \leq k \leq 255$
Операция:	$(f)+1 \rightarrow (\text{dest})$	Операция:	$(W).\text{OR.}(k) \rightarrow W$
Биты состояния:	Z	Биты состояния:	Z
Код:	00 1010 dfff ffff	Код:	11 1000 kkkk kkkk
Описание:	Регистр f увеличивается на 1. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f.	Описание:	Содержимое регистра W логически складывается с 8-битовой константой k. Результат сохраняется в регистре W.
Циклов:	1	Циклов:	1
Пример:	INCF CNT, 1	Пример:	IORWL 0x35
	Перед выполнением команды: CNT=0xFF Z=0		Перед выполнением команды: W=0x9A
	После выполнения команды: CNT=0x00 Z=1		После выполнения команды: W=0xBF
<b>INCFSZ</b>	Increment F, Skip if 0 Инкремент F, пропустить команду, если 0	<b>IORWF</b>	Inclusive OR W and F Логическое ИЛИ W и F
Синтаксис:	INCFSZ f, d	Синтаксис:	IORWF f,k
Операнд:	$0 \leq f \leq 127, d \in [0,1]$	Операнд:	$0 \leq f \leq 127, d \in [0,1]$
Операция:	$(f)+1 \rightarrow (\text{dest}) ; \text{пропустить, если } (\text{dest})=0$	Операция:	$(W).\text{OR.}(k) \rightarrow (\text{dest})$
Биты состояния:	не изменяются	Биты состояния:	Z
Код:	00 1111 dfff ffff	Код:	00 0100 dfff ffff
Описание:	Регистр f увеличивается на 1. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f. Если результат не равен 0, выполняется следующая команда. Если результат равен 0, следующая команда, считанная в текущем командном цикле, игнорируется, и вместо нее в следующем цикле выполняется NOP, в результате команда выполняется за 2 цикла.	Описание:	Содержимое регистра W логически складывается с содержимым регистра f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f.
Циклов:	1(2)	Циклов:	1
Пример:	HERE INCFSZ CNT, 1 GOTO LOOP	Пример:	IORWF RES, 0
	CONTINUE		Перед выполнением команды: RES=0x13 W=0x91
	Перед выполнением команды: PC=адрес HERE		После выполнения команды: RES=0x13 W=0x93
	если CNT=0, PC=адрес Continue		
	если CNT≠0, PC=адрес HERE+1		

<b>MOVF</b>	Move F Пересылка регистра F  Синтаксис: MOVF f,d Операнд: 0 ≤ f ≤ 127, d ∈ [0,1] Операция: (f) → (dest) Биты состояния: Z Код: 00 1000 dfff ffff Описание: Содержимое регистра f пересылается в регистр W, если d=0, и в тот же регистр f, если d=1. Назначение d=1 имеет смысл использовать для проверки регистра на ноль, поскольку команда изменяет флаг Z.  Циклов: 1 Пример: MOVF FSR, 0 После выполнения команды: W= значение FSR	<b>MOVWF</b> Move W to F Пересылка W в F  Синтаксис: MOVWF f Операнд: 0 ≤ f ≤ 127 Операция: (W) → (f) Биты состояния: не изменяются Код: 00 0000 1fff ffff Описание: Содержимое регистра W пересылается в регистр F. Циклов: 1 Пример: MOVWF OPT Перед выполнением команды: OPT=0xFF      W=0x4F После выполнения команды: OPT=0x4F      W=0x4F
<b>MOVLW</b>	Move literal to W Пересылка константы в W  Синтаксис: MOVLW k Операнд: 0 ≤ k ≤ 255 Операция: k → W Биты состояния: не изменяются Код: 11 00xx kkkk kkkk Описание: 8-битовая константа загружается в регистр W. Циклов: 1 Пример: MOVLW 0x5A После выполнения команды: W=0x5A	<b>NOP</b> No operation Холостая команда  Синтаксис: NOP Операнд: нет Операция: нет Биты состояния: не изменяются Код: 00 0000 0xx0 0000 Описание: Нет операции. Циклов: 1 Пример: NOP
		<b>OPTION</b> Load OPTION Register Загрузка регистра OPTION  Синтаксис: OPTION Операнд: нет Операция: (W) → OPTION Биты состояния: не изменяются Код: 00 0000 0110 0010 Описание: Содержимое рег-ра W загружается в регистр OPTION Циклов: 1 Пример: OPTION Перед выполнением команды: W=0x07 После выполнения команды: OPTION=0x07  Примечание: Для обеспечения совместимости с будущими продуктами семейства PIC16Cxx не рекомендуется использовать эту команду.

<b>RETFIE</b>	Return from interrupt Возврат из прерывания
Синтаксис:	RETFIE
Операнд:	нет
Операция:	(TOC) → (PC), (1) → (GIE)
Биты состояния:	не изменяются
Код:	00 0000 0000 1001
Описание:	Адрес возврата восстанавливается из вершины стека (TOC) в (PC). Разрешаются прерывания (устанавливается бит GIE).
Циклов:	2
Пример:	RETFIE
После прерывания:	PC=TOS      GIE=1
<b>RETLW</b>	Return with literal in W Возврат из подпрограммы с загрузкой константы в W
Синтаксис:	RETLW k
Операнд:	0 ≤ k ≤ 255
Операция:	k → W, (TOC) → (PC)
Биты состояния:	не изменяются
Код:	11 01xx kkkk kkkk
Описание:	8-битовая константа загружается в регистр W. Адрес возврата восстанавливается из вершины стека (TOC) в (PC).
Циклов:	2
Пример:	CALL TABLE
	W содержит смещение в таблице. Теперь в W значение из таблицы.
TABLE ADDWF PC	W= смещение
RETLW K1	Начало таблицы.
RETLW Kn	Конец таблицы.

<b>RETURN</b>	Return from Subroutine Возврат из подпрограммы
Синтаксис:	RETURN
Операнд:	нет
Операция:	(TOC) → (PC)
Биты состояния:	не изменяются
Код:	00 0000 0000 1000
Описание:	Адрес возврата восстанавливается из вершины стека (TOC) в (PC).
Циклов:	2
Пример:	RETURN
После прерывания:	PC=TOS
<b>RLF</b>	Rotate Left F through carry Сдвиг влево через перенос
Синтаксис:	RLF f,d
Операнд:	0 ≤ f ≤ 127, d ∈ [0,1]
Операция:	f<n> → d<n+1>, f<7> → C, C → d<0>
Биты состояния:	C
Код:	00 1101 dfff ffff
Описание:	Содержимое регистра f сдвигается на 1 бит влево через бит переноса C. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f
Циклов:	1
Пример:	RLF REG, 0
Перед выполнением команды:	REG1=11100110 C=0
После выполнения команды:	REG1=11100110 C=1 W=11001100



<b>RRF</b>	Rotate Right F through carry Сдвиг вправо через перенос	<b>SUBLW</b>	Subtract W from literal Вычитание W из константы
Синтаксис:	RRF f,d	Синтаксис:	SUBLW k
Операнд:	$0 \leq f \leq 127$ , $d \in [0,1]$	Операнд:	$0 \leq k \leq 255$
Операция:	$f<n> \rightarrow d<n-1>, f<0> \rightarrow C, C \rightarrow d<7>$	Операция:	$(k)-(W) \rightarrow W$
Биты состояния:	C	Биты состояния:	C, DC, Z
Код:	00 1100 dfff ffff	Код:	11 110x kkkk kkkk
Описание:	Содержимое регистра f сдвигается на 1 бит вправо через бит переноса C. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре f	Описание:	Содержимое регистра W вычитается из 8-битовой константы k. Результат помещается в регистр W. Биты C и DC устанавливаются в 1 в случае отсутствия заема из старшего разряда.
			
Циклов:	1	Циклов:	1
Пример:	RRF REG, 0	Пример 1:	SUBLW 0x02
	Перед выполнением команды: REG1=11100110 C=0		Перед выполнением команды: W=1
	После выполнения команды: REG1=11100110 C=0		После выполнения команды: W=1 C=1
	W=01110011		Результат положительный
<b>SLEEP</b>	Go into Standby mode Переход в режим SLEEP	Пример 2:	SUBLW 0x02
Синтаксис:	SLEEP		Перед выполнением команды: W=2
Операнд:	нет		После выполнения команды: W=0 C=1
Операция:	00h → WDT, 00h → WDT prescaler, 1 → TO, 0 → PD		Результат ноль
Биты состояния:	TO, PD	Пример 3:	SUBLW 0x02
Код:	00 0000 0110 0011		Перед выполнением команды: W=3
Описание:	Команда сбрасывает сторожевой таймер WDT и предварительный делитель. В регистре состояния устанавливается бит TO и сбрасывается бит PD. Процессор переходит в режим SLEEP с выключенным генератором.		После выполнения команды: W=FF C=0
Циклов:	1		Результат отрицательный
Пример:	SLEEP		

<b>SUBWF</b>	Subtract W from F Вычитание W из F
Синтаксис:	SUBWF f, d
Операнд:	$0 \leq f \leq 127, d \in [0,1]$
Операция:	$(f) - (W) \rightarrow (\text{dest})$
Биты состояния:	C, DC, Z
Код:	00 0010 dfff ffff
Описание:	Содержимое регистра W вычитается из содержимого регистра f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре. Биты C и DC устанавливаются в 1 в случае отсутствия заема из старшего разряда.
Циклов:	1
Пример 1:	SUBWF REG1, 1
Перед выполнением команды:	REG1=3      W=2
После выполнения команды:	REG1=1      W=2      C=1
	Результат положительный
Пример 2:	SUBWF REG1, 1
Перед выполнением команды:	REG1=2      W=2
После выполнения команды:	REG1=0      W=2      C=1
	Результат ноль
Пример 3:	SUBWF REG1, 1
Перед выполнением команды:	REG1=1      W=2
После выполнения команды:	REG1=0xFF    W=2      C=0
	Результат отрицательный

<b>SWAPF</b>	Swap halves F Обмен тетрад F
Синтаксис:	SWAPF f, d
Операнд:	$0 \leq f \leq 127, d \in [0,1]$
Операция:	$f<0:3> \rightarrow d<4:7>, f<4:7> \rightarrow d<0:3>$
Биты состояния:	не изменяются
Код:	00 1110 dfff ffff
Описание:	Содержимое старшей и младшей тетрад регистра F обменивается. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в том же регистре F.
Циклов:	1
Пример:	SWAPF REG, 0
Перед выполнением команды:	REG=0xA5
После выполнения команды:	REG=0x5A
<b>TRIS</b>	Load TRIS Register Загрузка регистра TRIS
Синтаксис:	TRIS f
Операнд:	$5 \leq f \leq 7$
Операция:	$(W) \rightarrow \text{TRIS}$
Биты состояния:	не изменяются
Код:	00 0000 0110 0FFF
Описание:	Содержимое регистра W загружается в регистр TRISA или TRISB при f=5 или 6 соответственно.
Циклов:	1
Пример:	TRIS TRISB
Перед выполнением команды:	W=0xA5
После выполнения команды:	TRISB=0xA5
Примечание:	Для обеспечения совместимости с будущими продуктами семейства PIC16Cxx не рекомендуется использовать эту команду.

<b>XORLW</b>	Exclusive OR literal to W Исключающее ИЛИ константы и W
Синтаксис:	XORLW k
Операнд:	$0 \leq k \leq 255$
Операция:	$(W).XOR. (k) \rightarrow W$
Биты состояния:	Z
Код:	11 1010 kkkk kkkk
Описание:	Содержимое регистра W поразрядно складывается с 8-битовой константой k по модулю 2. Результат сохраняется в регистре W.
Циклов:	1
Пример:	XORWL 0xAF Перед выполнением команды: W=0xB5 После выполнения команды: W=0x1A
<b>XORWF</b>	Exclusive OR W and F Исключающее ИЛИ W и F
Синтаксис:	XORWF f,k
Операнд:	$0 \leq f \leq 127$ , $d \in [0,1]$
Операция:	$(W).XOR. (k) \rightarrow (\text{dest})$
Биты состояния:	Z
Код:	00 0110 dfff ffff
Описание:	Содержимое регистра W поразрядно складывается с содержимым регистра f по модулю 2. Если $d=0$ , результат сохраняется в регистре W. Если $d=1$ , результат сохраняется в том же регистре f.
Циклов:	1
Пример:	XORWF REG, I Перед выполнением команды: REG=0xAF W=0xB5 После выполнения команды: REG=0x1A W=0xB5

## Краткие сведения о среде разработки программ MPLab.

MPLab - это программа, разработанная фирмой Microchip Technology и исполняемая в Windows 95. MPLab позволяет создавать, отлаживать и оптимизировать программное обеспечение для микроконтроллеров PIC12/16/17. MPLab включает в себя редактор, ассемблер, программный эмулятор и средства управления проектом. Кроме того, он обеспечивает поддержку фирменных внутрисхемных эмуляторов PICMASTER и программатора PICStart.

### Обзор инструментария MPLab

**Средства управления проектом** (MPLab Project Manager) предоставляет возможность создавать проекты и работать с файлами, относящимися к данному проекту. Кроме того, с его помощью можно управлять трансляцией программы и загружать получающийся код в программный или внутрисхемный эмулятор.

**Текстовый редактор** (MPLab Editor) используется для написания исходного текста программ или для редакции любых других текстовых файлов.

**Универсальный ассемблер** (MPASM Universal PIC16/17 Microcontroller Assembler) компилирует исходную программу в исполняемый код, не выходя из среды MPLab. Ассемблер поддерживает макрорасширения, условную трансляцию, управление листингом программы, можно использовать несколько файлов для совместной компиляции.

**Программный эмулятор** (MPSim Software Simulator) позволяет отлаживать работу программы, не используя для этого целевой микроконтроллер. Процесс исполнения инструкций эмулирует сам компьютер.

**Внутрисхемный эмулятор** (PICMASTER Emulator) использует дополнительные аппаратные средства для проверки работы программ в реальном масштабе времени.

MPLab может работать в трех основных режимах.

**Simulator Mode** используется для быстрой отладки разрабатываемой программы.

**Editor Only Mode** позволяет писать, компилировать и контролировать ошибки, не используя эмуляторы.

**PICMASTER Emulator Mode** - наиболее мощное средство отладки программ в реальном времени, но для этого необходимы дополнительные аппаратные средства.

## Проект. Начало работы

Понятие проекта необходимо для того, чтобы четко определить файлы, относящиеся к разрабатываемой программе. Поэтому в начале работы над программой надо создать директорию для файлов проекта, в которой будут находиться все относящиеся к нему файлы.

Располагать проект можно в любой директории. Например: C:\Design\Prog1.

Эта директория может быть создана самим пользователем, либо ее можно создать в процессе диалога создания **Нового Проекта** (Project->New Project).

Если эта директория уже существует, то можно лишь указать программе ее местонахождение.

В процессе создания нового проекта MPLab автоматически открывает меню **Редактирование Проекта**. Это меню используется для того, чтобы убрать или добавить файлы, используемые в проекте. Как правило, это исходные тексты программ с расширением .asm. Если после создания проекта необходимо добавить или убрать какие-то файлы из перечисленных в проекте, то следует воспользоваться пунктом меню **Редактирование Проекта** (Project->Edit Project).

Для того чтобы оттранслировать исходную программу, надо воспользоваться директивой **Создать Проект** (Project->Make Project) или **Построить Проект** (Project->Build All). Эти две директивы отличаются только тем, что первая транслирует только файлы, изменившиеся после последней компиляции, а вторая полностью обновляет все файлы независимо от того, изменились они или нет.

В процессе ассемблирования появляется окно с сообщением о результате компиляции. В случае ее успешного завершения можно нажать кнопку **OK**. Если же были ошибки, то автоматически открывается **Окно Файла Ошибок** (Error File Window), в котором можно просмотреть диагностику процесса трансляции.

Выход из программы MPLab - File->Exit, или Alt-F4.

## Основные функции отладчика

Вопросы, относящиеся к ассемблеру и написанию самой программы, более подробно рассмотрены в следующей главе. В этой главе рассматриваются основные функции отладчика.

Если есть уже написанная и откомпилированная программа, то в принципе можно воспользоваться программатором и занести программу в микроконтроллер. Однако с большой степенью вероятности эта программа будет работать не так, как вам бы того хотелось. Чтобы проверить, что происходит на самом деле, имеется несколько возможностей:

- потратить массу времени, внимательно просматривая текст программы и пытаться понять, что там не так;
- использовать внутрисхемный (аппаратный) эмулятор, который позволяет полностью заменить микроконтроллер управляющим компьютером;
- использовать программный эмулятор, отлаживая программу по шагам и анализируя отображающуюся на экране информацию о состоянии фиктивного микроконтроллера.

Поскольку в нашей работе будет использоваться последний способ, далее речь пойдет об инструментарии программного эмулятора.

MPLab-SIM - это программный эмулятор, который в отличии от внутрисхемного эмулятора работает с дискретными событиями. Поэтому основным направлением его использования является отладка программного кода, а не всего проектируемого устройства в целом. Поскольку основная часть работы, как правило, связана именно с этим процессом, то программный эмулятор - весьма эффективное средство разработки. Кроме того, он полностью поддерживает программную и аппаратную модель выбранного микроконтроллера и позволяет имитировать воздействие на внешние выводы микросхемы. Единственное ограничение для таких воздействий заключается в том, что просмотр изменения состояния внешних выводов производится не непрерывно, а перед выполнением очередной команды. Отсюда следует основное ограничение: невозможно эмулировать события, которые имеют дискретность меньше, чем время выполнения одной инструкции. Проблемы также возникают при эмуляции полностью асинхронных событий.

Продолжая сравнение с внутрисхемным эмулятором, можно сказать, что использование внутрисхемного эмулятора не позволяет оперативно заглянуть в содержимое регистров, а в режиме программного эмулятора эта информация всегда под рукой. Детальную информацию о состоянии про-

граммы и процессора можно, однако, получить, используя в критических местах точки останова программы (см. дальше по тексту).

В большинстве случаев предпочтительнее сначала воспользоваться программным эмулятором для отладки и оптимизации кода, а затем уже пользоваться внутрисхемным эмулятором или проверять работу устройства в целом.

Возможности программного эмулятора представлены ниже:

- эмуляция памяти,
- точки останова и трассировки,
- пошаговое выполнение,
- отображение регистров (как специальных, так и файлового регистра),
- режим анимации.

Сам процесс эмуляции выглядит примерно так. В пошаговом режиме при выполнении программы компьютер берет очередную команду из эмулируемой программной памяти, определяет, что она должна делать и в соответствии с этим обновляет необходимые компоненты программной модели микроконтроллера. Все произведенные изменения отображаются на экране в соответствующих окнах и полях.

В режиме анимации воспроизводится автоматический пошаговый режим и пользователю остается только наблюдать за изменением информации, отображаемой на экране.

В режиме реального времени (Real Time) программа исполняется с максимально возможной скоростью, которую позволяет компьютер. На контроль за процессом исполнения программы при этом могут воздействовать такие события:

- точки останова,
- точки трассировки,
- адреса счетчиков прохода.

Точки останова и точки трассировки действуют совершенно независимо друг от друга и их можно размещать в любом месте программы.

Точка останова - это состояние, в котором процессор обрабатывает команду и останавливается, если выполнены некоторые условия.

Для задания точек останова выберите пункт меню **Debug->Break Settings...**

Точки трассировки - это точки в программе (адреса или метки), информация о которых записывается в буфер размером 8К при попадании программы в эти адреса.

Для задания точек трассировки выберите пункт меню **Debug->Trace Settings...**

Для этих точек останова можно задать число проходов, через которые произойдет соответственно останов или данные об этой точке попадут в файл трассировки.

Установку этого параметра можно произвести в окне редактирования точек останова (трассировки) кнопкой **Set Pass Count** (Установить число проходов).

Определенные точки останова выделяются во всех окнах, в соответствующих местах красным (**Break Points**) и зеленым (**Trace Points**) цветом.

Определенные точки останова можно оперативно включать и выключать, используя правую кнопку мыши.

Данные о трассировке можно посмотреть в окне трассировки (**Trace Window**)

Для сохранения текущего окна в файл, для последующего анализа надо выбрать опцию меню **File->Export->Save Trace File**.

Более полную информацию о функциях отладчика можно найти в [3].

### Специальные окна

Перечислим еще окна, которые можно использовать для обзора различной информации о процессе исполнения программы.

**Program Memory** отображает инструкции программы, помещенные в пространство эмулируемой программной памяти.

**Trace Memory** отображает поход трассируемых точек в программе.

**EEPROM Memory**, если процессор имеет ячейки электрически стираемой памяти данных, то они отображаются в этом окне.

**Absolute Listing** - это листинг трансляции программы с исходным текстом и скомпилированным кодом.

**Stack** отображает каждый из адресов возврата из подпрограмм.

**File Register** отображает регистры общего назначения.

**Special Function Registers** отображает специальные регистры.

**Stopwatch** - показывает время исполнения программы. Это время подсчитывается только тогда, когда эмулируемый процессор исполняет команды.

**Custom Watch Window** - в этом окне можно собрать в одно место все, за чем хотелось бы наблюдать в ходе выполнения программы.

**Modify** - это окно диалога, в котором можно изменить текущее состояние данных, стека, программного кода или EEPROM.

## Функции внешних воздействий

Программный эмулятор MPLab-SIM позволяет задавать регулярные внешние воздействия, которые могут храниться в файле и воспроизводиться во время выполнения контроллером программы. Этот файл можно создавать и редактировать с помощью любого текстового редактора.

Асинхронные воздействия (*Asynchronous stimulus*) эмулируют непериодические сигналы на ножках микроконтроллера, такие как сброс или прерывание. При этом можно представить входное событие одним из четырех типов:

- High** переход в высокое состояние,
- Low** переход в низкое состояние,
- Toggle** переключение при каждом воздействии,
- Pulse** импульс при каждом воздействии.

Асинхронное воздействие можно предъявлять, когда эмулятор находится в режиме **Run**, если же используется пошаговый режим или процессор остановлен в какой-то точке останова, то можно просто изменять содержимое нужного регистра.

Воздействия на выводы (**Pin stimulus**) определяют значения, которые устанавливаются на внешних выводах на каком-то определенном шаге исполнения программы или в какой-то момент времени. Значения, ножки и шаги определяются в текстовом файле (*Stimulus File*). Этот файл нужно открыть, используя меню **Debug->Simulator Stimulus**, чтобы программе стали доступны определенные пользователем задания.

Воздействие некоторой тактовой частоты можно задать с помощью **Clock Stimulus**. При этом определяется частота и скважность по отношению к тактовой частоте микроконтроллера.

В случае работы с устройствами, подобными АЦП, более удобным может оказаться изменение состояния регистров (**Register Stimulus**). Значения, занесенные в файл, подставляются в эмулируемые регистры контроллера при достижении выбранных точек программы.

## Расширения файлов, используемые MPLab

Заданные по умолчанию расширения файлов, используемых MPLab, перечислены ниже.

<b>*.COD</b>	Содержит символьическую информацию и объектный код плюс другие данные, генерированные MPASM или MPC. Содержит большинство информации, связанной проектом.
<b>*.PJT</b>	Исходный файл C.
<b>*.C</b>	Include файл C.
<b>*.H</b>	Исходный файл ассемблера.
<b>*.ASM</b>	Файл для включения ассемблером.
<b>*.INC</b>	Исполняемый PIC16/17 в шестнадцатеричном формате.
<b>*.HEX</b>	Файлы конфигураций/установок.
<b>*.CFG</b>	Трассировочный файл.
<b>*.TRC</b>	Файл описания панелей инструментов.
<b>*.TBR</b>	Файл абсолютного листинга (распечатки), генерированный ассемблером/компилятором.
<b>*.LST</b>	Файл описания ошибок, генерированный во время работы ассемблера/компилятора.
<b>*.ERR</b>	Файл конфигурации Watch-окна.
<b>*.WAT</b>	Файл шаблона.
<b>*.TPL</b>	Файл конфигурации «горячих» клавиш MPLab.
<b>*.KEY</b>	Файл трассировки условных остановов.
<b>*.TB</b>	Файл воздействий на внешние выводы (Stimulus).
<b>*.STI</b>	Файл воздействий на регистры (Stimulus).
<b>*.REG</b>	

## Основы работы с ассемблером MPAsm

Microchip Technology предоставляет выбор способов использования кросс-ассемблера MPAsm. Эту программу можно использовать как отдельно, встраивая ее в привычные пользователю интегрированные среды разработки или запуская ее из командной строки MS-DOS или Windows, так и совместно со средой разработки, предоставляемой самой фирмой-изготовителем микроконтроллеров MPLab (см. следующую главу). Описание системных требований также можно найти в следующей главе.

Мы будем рассматривать использование кросс-ассемблера (или просто ассемблера) только из интегрированной среды разработки MPLab, поэтому детали, относящиеся к особенностям иного способа запуска MPAsm, будут опущены. Кроме того, для решения наших задач мы не будем использовать расширенные возможности MPLink и MPLib.

В целом MPAsm ничем не отличается от любого другого ассемблера, являясь фактически построчным компилятором. Это означает, что информационной единицей в написанной пользователем программе, обрабатываемой ассемблером, является строка исходного текста. Компилятор последовательно транслирует строки программы, начиная с первой, и заканчивает этот процесс по достижении директивы END. Обычно ассемблер производит два (или три) прохода, но для пользователя знания этих тонкостей может и не потребоваться.

Максимальное число символов в строке 255.

Исходный текст можно создавать, используя любой редактор ASCII текста. Мы для этих целей будем использовать редактор, встроенный в интегрированную среду MPLab.

Ниже приведен пример текста программы.

```
; Sample MPAsm Source Code. It is for illustration only.

; list      p=16C54, r=HEX
org      0x1ff ; Reset Vector
goto    Start ; Go back to the beginning
org      0x000 ; The main line code starts here
Start
    movlw   0xa0 ; Perform some PIC16/17 code
    movlw   0xb0 ;
    goto    Start ; do it forever...
end
```

Каждая строка исходного текста может содержать до 4-х полей информации различного типа. Этими типами являются:

Метка	Label
Мнемоника	Mnemonic
Операнд(ы)	Operand(s)
Комментарий	Comment

Соблюдение порядка следования и положения в строке является обязательным условием.

Метки должны начинаться в первой колонке. Признаком конца имени метки может быть двоеточие <:>, пробел, табуляция или перевод строки. Метки должны начинаться с буквы или подчерка <\_> и могут содержать буквы, цифры, знак подчерка и знак вопроса. Метка не должна превышать 32 символа. По умолчанию имена меток чувствительны к верхнему

нижнему регистру. Если для определения метки используется двоеточие на конце, то оно не относится к имени метки.

**Мнемоники** - это мнемоники команд микроконтроллера, директивы ассемблера или макропрограммы. Они не должны начинаться в первой колонке. Если в той же строке содержится метка, то мнемоника должна быть отделена от метки хотя бы одним пробелом или табуляцией.

**Операнды** всегда следуют за мнемоникой и должны быть также отделены от нее хотя бы одним пробелом или табуляцией. В списке операнды разделяются между собой запятыми.

**Комментарии** могут начинаться в любом месте строки и после любого предыдущего текста. Нужно только помнить, что все, что написано после признака начала комментария <,>, является комментарием и игнорируется ассемблером.

## Файлы, используемые ассемблером

По умолчанию ассемблер использует следующие расширения для имен файлов:

.ASM	Расширение для имени исходного файла MPAsm: <source_name>.ASM
.OBJ	Расширение для имени перемещаемого объектного файла MPAsm: <source_name>.OBJ
.LST	Расширение для имени выходного файла листинга программы MPAsm или MPLink: <source_name>.LST
.ERR	Расширение для имени выходного файла описания ошибок при компиляции: <source_name>.ERR
.HEX	Расширение для имени выходного HEX-файла: <source_name>.HEX
.COD	Расширение для имени выходного файла с символьной и отладочной информацией: <source_name>.COD
.HXL	Расширение для имени выходных файлов, содержащих отдельно младшие и старшие байты: <source_name>.HXL, <source_name>.HXH

Выходной HEX-файл может иметь несколько разных форматов. За более подробной информацией следует обратиться к фирменному описанию, например [5].

## Директивы ассемблера

В этом разделе описываются директивы ассемблера, с помощью которых можно управлять процессом компиляции исходного файла, параметрами выходных файлов, резервированием памяти под данные и программу и т. п.

В ассемблере предусмотрено 4 основных типа директив.

**Data Directives** - директивы для данных. С их помощью осуществляется резервирование памяти и присвоение символьических имен для адресов и данных.

**Listing Directives** - директивы управления созданием файла листинга компилируемой программы. С их помощью можно указать заголовки, параметры страниц выходного файла и другие опции, связанные форматированием текста.

**Control Directives** - эти директивы служат для написания секций условного ассемблирования. Т. е. таких участков программы, окончательный вид которых будет формироваться на этапе компиляции и зависит от условий, заданных в самом тексте программы.

**Macro Directives** - эти директивы управляют процессом трансляции программы внутри макроопределений.

В этом пособии мы рассмотрим большинство директив данных, а также некоторые директивы управления листингом и условного ассемблирования. Язык макроопределений здесь не описывается, а для желающих с ним познакомиться рекомендуется обратиться к фирменному руководству или воспользоваться средством HELP самого ассемблера.

Далее приводятся наиболее употребительные команды ассемблера. Еще раз отметим, что это далеко не полный список. И, кроме того, для правильного использования синтаксиса команд микроконтроллера рекомендуется обращаться к описанию системы команд микроконтроллера, для которого пишется программа.

## CONFIG

Установка программируемых битов конфигурации

**Синтаксис** `_config <expr>`

**Описание:**

Устанавливает биты программирования конфигурации процессора в соответствии со значением величины, описанной в `<expr>`. Описание конфигурационных битов содержится в соответствующих описаниях каждого типа микроконтроллеров.

До использования этой директивы необходимо с помощью директивы LIST или PROCESSOR указать тип процессора, для которого пишется программа.

**Пример:**

```
List      p=17c42, f=INHX32
_config  H'FFFF'          ; Default configuration bits
```

**См. также:** LIST PROCESSOR \_IDLOCS

## CONSTANT

Определение символьной константы

**Синтаксис** `constant <label>=<expr> [...,<label>=<expr>]`

**Описание:**

Создает символьную переменную для использования в выражениях ассемблера. Константы не могут быть переустановлены после того, как были инициализированы, и выражения должны быть полностью определены к моменту присвоения. Это принципиальное отличие между символьическим именем, объявляемым как CONSTANT и именами, определяемыми директивами VARIABLE или SET. В остальных случаях переменные и константы могут заменять друг друга в выражениях.

**Пример:**

```
Variable RecLength=64           ; Set Default RecLength
Constant BufLength=512          ; Init BufLength
                                ; RecLength may
                                ; be reset later
                                ; in RecLength=128
                                ;
                                ;
                                ;
constant MaxMem=RecLength+BufLength ; CalcMaxMem
```

**См. также:** SET VARIABLE

## DATA

Создание численных или текстовых данных

**Синтаксис** [label] data <expr>[,<expr>,...<expr>]  
[label] data "<text\_string>"["<text\_string>"]...

**Описание:**

Размещает одно или больше слов в программной памяти. Данные могут быть константами, перемещаемыми или внешними метками, или выражениями. Данные могут состоять из символов ASCII, текста, заключенного в одинарные кавычки для одного символа или двойные кавычки для нескольких. Один символ помещается в правую (младшую) часть слова, в то время как несколько символов упаковываются по два в одно слово, где старший байт соответствует первому символу. Если вводится нечетное число символов, последний байт будет нулевым.

Повторяющееся выражение последовательно заполняет байты программной памяти, пока не закончится список. Подразумевается четное число выражений, в противном случае последний байт будет заполнен нулем.

**Пример:**

```
Data reloc_label+10 ; constants
Data 1,2,ext_label ; constants, externals
Data "testing 1,2,3" ; text string
Data 'N' ; single character
Data start_of_program ; relocatable label
```

См. также: DATA DB DE DT DW

## DB

Определение одного байта данных

**Синтаксис** [label] db <expr>[,<expr>,...<expr>]

**Описание:**

Резервирует слова в программной памяти, в которые упаковывается байт информации. Повторяющиеся выражения последовательно заполняют байты программной памяти, пока не закончится список. Подразумевается четное число выражений, в противном случае последний байт будет заполнен нулем.

**Пример:**

```
Db 't', 0x0f, 'e', 0x0f, 's', 0x0f, 't', '\n'
```

См. также: DATA DE DT DW

## DE

Определение байта данных для EEPROM

**Синтаксис** [label] de <expr>[,<expr>,...<expr>]

**Описание:**

Резервирует слова памяти с 8-битовыми данными. Каждое выражение должно определять 8-битовое значение. Старшие биты обнуляются. Каждый символ заносится в отдельную ячейку.

Хотя эта директива предназначена для инициализации данных EEPROM в микросхемах PIC16C8x, она может быть использована в любом месте программы для любого процессора.

**Пример**

```
Org H'2100' ; Initialize EEPROM Data
De "My Program, v1.0", 0
```

См. также: DATA DB DT DW

## DT

Определение таблицы

**Синтаксис** [label] dt <expr>[,<expr>,...<expr>]

**Описание:**

Генерирует последовательность команд RETLW, по одной для каждого выражения. Каждое выражение должно определять 8-битовое значение. Каждый байт заносится в отдельную ячейку программной памяти как команда RETLW.

**Пример**

```
dt "A Message", 0
dt FirstValue, SecondValue, EndOfValues
```

См. также: DATA DB DE DW

## DW

Определение одного слова данных

**Синтаксис** [label] dw <expr>[,<expr>,...<expr>]

**Описание:**

Резервирует слова в программной памяти, в которые записывается соответствующая информация. Повторяющиеся выражения последовательно заполняют слова программной памяти, пока не закончится список. Выражения могут быть последовательностями символов и располагаются в памяти так же, как описано для инструкции DATA.

**Пример**

```
dw 39, "diagnostic 39", (d_list*2+d_offset)
dw diagbase-1
```

См. также: DATA DB

## END

Конец программы

**Синтаксис** end

**Описание:**

Отмечает конец текста программы

**Пример**

```
start ; executable code
.
.
.
end ; end of instructions
```

См. также:

## EQU

Определение ассемблерной константы

**Синтаксис** [label] equ <expr>

**Описание:**

Значение выражения присваивается метке

**Пример**

```
four equ 4 ; assigned the numeric value of
; to label four
```

См. также: SET

## INCLUDE

Вставить дополнительный исходный файл

**Синтаксис**    include <<include\_file>>  
              include "<include\_file>"

**Описание:**

Указанный файл считывается как текст программы. При достижении конца файла этот текст встраивается в исходную программу. Допустимо до шести уровней вложения этой директивы. Если указан полный путь, то только он и будет использоваться для поиска. В противном случае порядок поиска таков: текущая директория, директория для исходных файлов, директория, где находится сама программа MPAsm.

**Пример**

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

**См. также:**

## LIST

Управление созданием листинга программы

**Синтаксис**    list [<list\_option>, ..., <list\_option>]

**Описание:**

Если эта инструкция использована без operandов, то ее действие сводится к тому, что она разрешает создание выходного листинга, если где-то перед этим он был запрещен. В других случаях директива, в соответствии с operandами, используется для управления процессом ассемблирования или форматом файла листинга.

**Пример**

```
list p=17c42, f=INHX32, r=DEC
```

**См. также:** NOLIST PROCESSOR RADIX

Список параметров директивы List

Параметры	По умолчанию	Описание
b=nnn	8	Установить шаг табуляции
c=nnn	132	Установить число колонок текста
f=<format>	INHX8M	Установить один из следующих форматов выходных HEX- файлов - INHX32, INHX8M или INHX8S
Mm=ON OFF	On	Выводить карту памяти в файл листинга
n=nnn	60	Установить число строк в странице
p=<type>	None	Установить тип процессора, например PIC16C54
r=<radix>	hex	Установить radix по умолчанию: hex, dec, oct
st=ON OFF	On	Выводить таблицу символических имен в файл листинга
t=ON OFF	Off	Обрезать строки листинга программы (иначе - перенести)

**Примечание:** Все параметры директивы LIST воспринимаются как десятичные операнды.

## NOLIST

Запрещение записи в выходной файл листинга

**Синтаксис**    NOLIST

**Описание:**

Запрещает создание выходного листинга.

**См. также:** LIST

## ORG

Устанавливает начальный адрес

**Синтаксис**    [<label>] org <expr>

**Описание:**

Присваивает значение, заданное выражением, счетчику команд трансляции. Если определена метка, то ей присваивается то же значение. Если в исходном тексте нет директивы ORG, то генерация кода начинается с нулевого адреса. Когда создается объектный файл, использование метки обязательно.

**Пример**

```
int_1 org 0x20 ; Vector 20 code goes here  
int_2 org int_1+0x10 ; Vector 30 code goes here
```

**См. также:** RES FILL

## PAGE

Вставляет признак начала новой страницы

**Синтаксис**    page

**Описание:**

В выходной файл листинга программы вставляется признак начала новой страницы.

**См. также:** LIST TITLE SUBTITLE

## PROCESSOR

Устанавливает тип процессора

**Синтаксис**    processor <processor\_type>

**Описание:**

Устанавливает тип используемого процессора.

**Пример**

```
processor 16C54
```

**См. также:** LIST

## RADIX

Устанавливает систему исчисления по умолчанию

**Синтаксис** radix <default\_radix>

**Описание:**

Устанавливает систему исчисления по умолчанию. Если директива нигде не использована, то по умолчанию используется HEX. Допустимые параметры директивы hex, dec, oct.

**Пример**

```
radix dec
```

См. также: LIST

## RES

Резервирование памяти

**Синтаксис** <label> res <mem\_units>

**Описание:**

Счетчик команд получает значение, увеличенное от текущего значения на величину, равную <mem\_units>. Обратите внимание, что метке будет присвоено значение адреса, а не константы или переменной.

**Пример**

```
Buffer res 64 ; reserve 64 words of storage
```

См. также: ORG FILL

## SET

Определение ассемблерной переменной

**Синтаксис** <label> set <expr>

**Описание:**

Значение выражения присваивается метке. Директива SET эквивалентна EQU, за исключением того, что метку, определенную с помощью директивы SET, можно переопределить другой такой же командой.

**Пример**

```
Area set 0
Width set 0x12
Length set 0x14
Area set length * width
Length set length + 1
```

См. также: EQU VARIABLE

## SUBTITLE

Определяет подзаголовок программы

**Синтаксис** subtitle "<sub\_text>"

**Описание:**

<sub\_text> это ASCII текст, заключенный в двойные кавычки, длиной не более 60 символов. Эта директива определяет вторую строку заголовка в выходном файле листинга.

**Пример**

```
subtitle "diagnostic section"
```

См. также: TITLE

## TITLE

Определяет заголовок программы

**Синтаксис** title "<title\_text>"

**Описание:**

<title\_text> это ASCII текст, заключенный в двойные кавычки, длиной не более 60 символов. Эта директива определяет верхнюю строку заголовка на каждой странице выходного файла листинга.

**Пример**

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

См. также: LIST SUBTITLE

## VARIABLE

Определение символьной переменной

**Синтаксис** variable <label>[=<expr>,...,<label>[=<expr>]]

**Описание:**

Создает символьную переменную для использования в выражениях ассемблера. Переменные и константы могут заменять друг друга в выражениях.

Директива VARIABLE создает символьическое имя, которое функционально эквивалентно создаваемому директивой SET. Разница между ними состоит в том, что для директивы VARIABLE не требуется инициализация переменной во время ее определения. Необходимо отметить, что значение переменной не может быть изменено вместе с операндом. Необходимо использовать присвоение значения переменной, инкремент или декремент в отдельных строках программы.

**Пример**

См. пример к директиве CONSTANT

См. также: SET CONSTANT

## Синтаксис выражений и операций ассемблера

Выражения, используемые в строках исходного текста в качестве operandов, могут состоять из констант, символьических имен или любой комбинации тех и других, разделенных арифметическими операторами. Перед каждой константой или символьическим именем может стоять знак плюс или минус для индикации положительного или отрицательного выражения.

**Операторы** - это арифметические символы такие, как, например, знак «+» или «-». Они используются для написания выражений. Есть определенный порядок выполнения операторов в соответствии с присвоенным старшинством.

Последовательность выполнения операторов определяет, в каком порядке производится вычисление элементов выражения. Обычно действия выполняются слева направо и выражения, заключенные в скобки, всегда вычисляются первыми.

**Radix** задает основание системы исчисления, которое используется при вычислении выражений. По умолчанию используется шестнадцатеричный Radix (основание 16). Radix по умолчанию можно изменить директивой ассемблера. Можно также изменять Radix с помощью операторов управления системой исчисления.

**Примечание:** Все выражения вычисляются как целочисленные и 32-битовые.

#### Текстовые последовательности

Таковыми являются последовательности, состоящие из любых допустимых символов ASCII (с кодами от 0 до 127) и заключенные в двойные кавычки. Они могут быть любой длины, которую только можно вместить в строке текста из 255 символов. Когда встречаются закрывающие кавычки, последовательность считается законченной. Если закрывающих кавычек нет, то последовательность заканчивается признаком конца строки. Не предусмотрено никаких средств переноса последовательности на следующую строку, в этом и нет необходимости, поскольку можно просто использовать еще одну директиву DW на следующей строке программы.

Директива DW размещает содержимое последовательности в идущие по порядку слова. Если в последовательности содержится нечетное количество символов, то директивы DW и DATA дополняют последнее слово нулевым байтом.

Если в поле операнда используется текстовая константа, то она должна состоять только из одного символа, иначе будет зафиксирована ошибка.

В приведенных ниже примерах обратите внимание на разницу в сгенерированных кодах для разных директив задания текстовых последовательностей.

```

7465 7374 696E      dw "testing output string one\n"
6720 6F75 7470
7574 2073 7472
696E 6720 6F6E
650A

B061                 #define str "testing output string two"
                      movlw "a"

7465 7374 696E      data "testing first output string"
6720 6669 7273
7420 6F75 7470
7574 2073 7472
696E 6700

```

В ассемблере допустимо использовать ANSI 'C' Escape-последовательности для того, чтобы определить специальные символы управления. Они приведены в таблице.

#### ANSI 'C' Escape-последовательности

Escape-символ	Описание	Значение
\a	Bell (alert) character	07
\b	Backspace character	08
\f	Form feed character	0C
\n	New line character	0A
\r	Carriage return character	0D
\t	Horizontal tab character	09
\v	Vertical tab character	0B
\\\	Backslash	5C
\?	Question mark character	3F
\'	Single quote (apostrophe)	27
\\"	Double quote character	22
\000	Octal number (zero, Octal digit, Octal digit)	
\xHH	Hexadecimal number	

#### Числовые константы и RADIX

В MPASM поддерживаются следующие типы систем исчисления: шестнадцатеричная, десятичная, восьмеричная, двоичная и коды символов. Система исчисления по умолчанию - шестнадцатеричная. Система исчисления по умолчанию определяет, какое значение будет присвоено константе в выходном объектном файле в том случае, если тип системы исчисления не задан явно при описании самой константы.

Если в определении константы нет знаков (+ или -) то считается, что она положительная.

Далее в таблицах представлены способы задания системы исчисления и арифметические операторы и порядок выполнения действий.

**Примечание:** В момент непосредственного вычисления выражения, определяющего константу, используется 32 бита без знака. Если полученное значение не помещается в предназначеннное для него поле, то будет выдано предупреждение.

## Определение системы исчисления

Тип	Синтаксис	Пример
Десятичная	D'<digits>'	D'100'
Шестнадцатеричная	H'<hex digits>'	H'9f'
Восьмеричная	O'<octal digits>'	O'777'
Двоичная	B'<binary digits>'	B'00111001'
Код символа	'<character>' A'<character>'	'C' A'C'

## Арифметические операторы и порядок выполнения действий

Оператор	Пример
( Left Parenthesis	l + (d * 4)
) Right Parenthesis	(Length + 1) * 256
! Item NOT (logical complement)	if ! (a == b)
- Negation (2's complement)	-1 * Length
high Return high byte	movlw high CTR Table
low Return low byte	movlw low CTR Table
*	a = b * c
/ Divide	a = b / c
% Modulus	entry len = tot len % 16
+	tot len = entry len * 8 + 1
- Subtract	entry len = (tot - 1) / 8
<< Left shift	flags = flags << 1
>> Right shift	flags = flags >> 1
>= Greater or equal	if entry idx >= num entries
> Greater than	if entry idx > num entries
< Less than	if entry idx < num entries
<= Less or equal	if entry idx <= num entries
== Equal to	if entry idx == num entries
!= Not equal to	if entry idx != num entries
& Bitwise AND	flags = flags & ERROR BIT
^ Bitwise exclusive OR	flags = flags ^ ERROR BIT
Bitwise inclusive OR	flags = flags   ERROR BIT
~ Complement	flags = ~flags
&& Logical AND	if (len == 512) && (b == c)
Logical OR	if (len == 512)    (b == c)
= Set equal to	entry index = 0
+= Add to, set equal	entry index += 1
-= Subtract, set equal	entry index -= 1
*= Multiply, set equal	entry index *= entry length
/= Divide, set equal	entry total /= entry length
%= Modulus, set equal	entry index %= 8
<<= Left shift, set equal	flags <<= 3
>>= Right shift, set equal	flags >>= 3
&= AND, set equal	flags &= ERROR FLAG
!= Inclusive OR, set equal	flags  = ERROR FLAG
^= Exclusive OR, set equal	flags ^= ERROR FLAG
\$ Return program counter	goto \$ + 3

## Работа с программатором

Программатор предназначен для записи отлаженной программы в микроконтроллер. Можно, конечно, записывать и не до конца проверенную программу, если такая необходимость возникает, но лучше все же стараться этого избегать, поскольку микросхемы микроконтроллеров имеют ограниченное число циклов перезаписи.

Для работы с программатором его необходимо соединить с параллельным портом компьютера. Этот порт обычно используется для принтера. После этого надо подать питание от какого-либо источника постоянного напряжения величиной от 15 до 20 В и запустить программу, обслуживающую программатор.

Рекомендуется перед использованием программатора закрыть все приложения, поскольку не исключена возможность обращения какой-либо задачи к параллельному порту, что может привести к неправильной работе программатора и даже повреждению программируемой микросхемы. Поэтому не запускайте ничего лишнего в то время, когда активно используете программатор.

После подачи питания считайте HEX-файл в буфер программатора и только после этого можете соединяться кабелем с платой, на которой стоит программируемый микроконтроллер, или вставить микроконтроллер в панельку для программирования.

## Программа управления программатором

Программа предоставляет удобный интерфейс для записи необходимой информации в микроконтроллер. Все действия можно выполнять с помощью меню или кнопок на панели инструментов. С помощью этой программы можно загрузить HEX-файл в память и просмотреть его содержимое. В этот же буфер можно считать информацию из ранее запрограммированного микроконтроллера и сохранить этот код в файле.

Можно проверить, надо ли очистить содержимое программной памяти в микросхеме. Если это микроконтроллер с ультрафиолетовым стиранием информации, то для стирания потребуется специальная лампа. Для электрически перепрограммируемых вариантов (PIC16C8X) нужно только нажать на панели инструментов кнопку ERASE.

По нажатии соответствующей кнопки начинается программирование, а после его окончания можно сравнить информацию в программном буфере и написанную в микроконтроллер.

Не забудьте, что программе неоткуда взять сведения о типе программируемого контроллера (а их несколько десятков), поэтому важным этапом работы с программой является установка нужного вам типа микроконтроллера на специально отведенной для этого панели. Если забыть это сделать, то в результате можно получить совершенно нерабочий экземпляр микросхемы. Это связано с тем, что разные семейства микроконтроллеров имеют разное расположение и назначение битов конфигурации, которые «пропшиваются» одновременно с записью программы. Для некоторых типов микроконтроллеров такая ошибка может привести к летальному исходу, потому что запрограммированный бит защиты микросхемы от считывания не у всех типов микросхем может быть восстановлен после стирания. Ну а про микросхемы, предназначенные для однократного программирования, и говорить нечего.

Полезно после загрузки HEX-файла проконтролировать биты конфигурации, которые всегда отображаются. Эти биты должны быть заданы в исходной программе, но при необходимости их можно изменить и во время программирования.

## Задания для самостоятельной работы

Задания ориентированы на использование микроконтроллера PIC16C84 (PIC16F84). Однако большинство работ можно проделать и с другими аналогичными контроллерами.

Первая группа заданий, названных упражнениями, посвящена изучению системы команд микроконтроллера и среды разработки программ MPLab. Для их выполнения требуется только персональный компьютер с установленной и настроенной программой MPLab.

Во второй части используются платы-заготовки, на которых смонтированы микроконтроллеры и некоторое обрамление. Эти задания предполагают достаточно длительную самостоятельную работу, включающую распределение ресурсов микроконтроллера для решения поставленной задачи.

Для загрузки программ, предварительно проверенных в среде MPLab, используется программатор и специальная программа для его поддержки. Результатом работы должно быть нормальное функционирование запрограммированного микроконтроллера и всего устройства в целом.

## Вопросы и задания для первого знакомства

В этом разделе представлены упражнения для быстрого освоения системы команд микроконтроллера PIC16C84 (PIC16F84) и оболочки MPLab. Все эти задания выполняются только в среде MPLab и не требуют их проверки на реальных микроконтроллерах. Проверка программ производится с использованием программного эмулятора, встроенного в MPLab.

Возможности интегрированной среды MPLab позволяют не только освоиться с системой команд используемого микроконтроллера, но и контролировать правильность программирования всех внешних устройств и учитывать особенности архитектуры. Практически все тонкости архитектуры контролируются программным эмулятором, поэтому, к примеру, невозможно управлять каким-либо выходом микросхемы, не запрограммировав предварительно соответствующую ножку как выход.

Последовательность выполнения упражнений (кроме первого) такая: определяется алгоритм, который более всего подходит для решения поставленной задачи, на ассемблере пишется программа, компилируется и проверяется с помощью программного эмулятора, встроенного в MPLab.

### **Упражнение 1.**

Реассемблировать, понять и описать, что делает нижеприведенный код программы:

```
3020 0084 0180 0A84 1E04 2802
```

Записать текст программы на ассемблере и проверить, как он будет выполняться программным эмулятором MPLab.

Для этого необходимо воспользоваться определениями файла P16F84.inc, включив этот файл в свою программу директивой INCLUDE, и использовать имена регистров, метки и символические имена.

### **Упражнение 2.**

Составить программу подсчета числа единиц в байте.

Исходный байт (SRC\_B) находится по адресу 20h, результат сохранить в 21h (DST\_B).

Есть несколько вариантов реализации этой программы. Попробуйте придумать хотя бы два.

### **Упражнение 3.**

В пять последовательно расположенных ячеек памяти файлового регистра, начиная с адреса 21h (ARR\_B), запишите любые числа. Необходимо подсчитать число байт в этих ячейках памяти, у которых младший бит (D0) равен 1. Результат поместить в 28h (DST\_B).

Определить число ячеек, больших 80h. Результат поместить в 29h (DST\_B1).

### **Упражнение 4.**

Сложить два 16-разрядных числа, находящихся в ячейках: первое (SRC1\_L, SRC1\_H) в 20h, 21h, второе (SRC2\_L, SRC2\_H) в 22h, 23h. Результат (DST\_L, DST\_H) поместить в ячейки 28h, 29h.

### **Упражнение 5.**

Из 16-разрядного числа в (SRC1\_L, SRC1\_H) 20h, 21h вычесть 16-разрядное число (SRC2\_L, SRC2\_H) в 22h, 23h, результат (DST\_L, DST\_H) поместить в 28h, 29h.

### **Упражнение 6.**

Поменять местами 16-разрядные операнды: находящиеся в 20h, 21h (SRC1\_L, SRC1\_H) - первые, находящиеся в 22h, 23h (SRC2\_L, SRC2\_H) - вторые.

### **Упражнение 7.**

Изменить на инверсное (обратное) значения всех битов в четырех последовательно расположенных ячейках памяти, начиная с адреса 20h (SRC\_B).

### **Упражнение 8.**

Установить в 1 четыре младших бита в пяти последовательно расположенных байтах памяти, начиная с адреса 20h (SRC\_B).

Установить в 0 четыре старших бита в пяти последовательно расположенных ячейках памяти, начиная с адреса 28h (SRC\_B1).

### **Упражнение 9.**

Содержимое ячейки 20h (SRC\_B) умножить на 8, результат поместить в ячейки 28h, 29h (DST\_L, DST\_H).

Содержимое ячеек 20h, 21h (SRC\_L, SRC\_H) разделить на 16, результат занести в 2Ah, 2Bh (DST\_L, DST\_H). Остаток не учитывать.

### **Упражнение 10.**

В ячейке (SRC\_B) с адресом 20h изменить расположение бит на зеркальное, т. е. D0<-->D7, D1<-->D6, D2<-->D5, D3<-->D4.

### **Упражнение 11.**

Используя меню программного эмулятора в MPLab **Debug->Simulator Stimulus->Asynchronous Stimulus**, назначьте на одну из кнопок бит D0 порта A. Присвойте этой кнопке способ воздействия **Toggle**.

Составьте программу, которая, проверяя в цикле состояние этого бита, будет инкрементировать регистр памяти каждый раз, когда на этом входе состояние изменяется с низкого уровня на высокий.

Результат поместить в 20h (DST\_B).

### **Упражнение 12.**

Сделайте установку типа воздействия **Toggle** для битов D0 и D1 порта А. Напишите программу, в которой запрограммируйте на выход выводы порта В, соответствующие битам D0 и D1. Далее программа должна синхронно изменять состояния выходов порта В в соответствии с состоянием входов порта А, которое задается переключением кнопок **Asynchronous Stimulus**.

### Упражнение 13.

Напишите программу, в которой инициализируется таймер и предварительный делитель. Проверьте работу таймера с помощью эмулятора.

Дополните программу точкой входа по прерыванию от таймера и подпрограммой обработки прерывания, которая, к примеру, инкрементирует один из регистров (DST\_B по адресу 20h). Разрешите прерывания от таймера. Основная часть программы может исполнять какой-либо пустой цикл. Проверьте изменение состояния таймера и обработку прерывания с помощью программного эмулятора.

### Задания для проектирования реальных устройств

Все приводимые ниже задания основываются на использовании одной и той же учебной платы, которая содержит в себе микропроцессор PIC16C84 или PIC16F84. На плате также расположены восемь 7-сегментных индикаторов и два регистра с ключами, которые обеспечивают управление этим индикатором. Один из входов микроконтроллера подключен к буферному транзистору и с его помощью можно подавать на процессор сигналы с уровнями, не согласованными с напряжением питания. С помощью двух DIP-переключателей можно изменять направление передачи двух сигналов, которые также используются для связи микроконтроллера с внешним миром, но уровни сигналов на этих входах (или выходах) должны соответствовать выбранному напряжению питания. Для подключения питания используется 9-контактный разъем. Для программирования (записи новой программы в микроконтроллер) к этому же разъему вместо питания подключается программатор. В другом разъеме (15-контактном) изначально не задействован ни один контакт, и его можно использовать по своему усмотрению для подключения к микроконтроллеру внешних сигналов.

Выполнение заданий предусматривает разработку алгоритмов работы устройства, схему внешних соединений учебной платы с другими устройствами (если это необходимо), разработку и отладку программы с использованием программного эмулятора и окончательное программирование микроконтроллера, используя программатор. Последние два действия приходится, как правило, повторять неоднократно, поскольку даже отладка программы с помощью эмулятора еще не гарантирует адекватную работу реального микроконтроллера.

### Задание 1. Часы.

Что такое часы, все знают, поэтому лишь порекомендуем формат отображения. Поскольку имеется 8 индикаторов, то можно задействовать их таким образом:

ЧЧ-ММ-СС

Буквы обозначают значение часов, минут и секунд, а тире - это тире.

Для начала можно написать программу наподобие секундомера (таймера). Затем преобразовать ее таким образом, чтобы можно было производить начальную установку и обратный счет. Для этого следует через внешний разъем присоединить пару кнопок. Придумайте алгоритм их экономного использования.

И наконец, имея эти заготовки, нетрудно все преобразовать в настоящее устройство для установки и индикации времени.

### Задание 2. Индикатор с приемом информации по RS-232.

Такой индикатор может потребоваться для отображения информации, посылаемой из другого устройства, например вольтметра. В нашем случае индикаторное устройство будет работать только на прием.

Для выполнения этого задания необходимы минимальные сведения о том, как устроен интерфейс последовательной связи RS-232, какие сигналы используются для связи и какие временные соотношения требуется соблюдать, чтобы обеспечить корректную работу интерфейса. В качестве одного из источников информации об этом интерфейсе можно назвать [2].

Устройство должно принимать информацию из последовательного порта IBM PC и отображать ее на индикаторе. Поскольку возможности отображения ограничены только цифрами (0...9), то необходимо контролировать поступающую информацию на предмет ее корректности. Некоторая модификация программы позволит отображать и символы шестнадцатиричных цифр (A, B, C, D, E, F), а возможно, и еще какие-нибудь другие символы.

Для передачи символов с компьютера можно воспользоваться программой эмуляции терминала, установив, к примеру, скорость передачи 9600 бод, 8 бит для передачи без контроля четности, два стоповых бита.

Передаваемая информация должна рассматриваться как символы в стандартной кодировке ASCII. Отображать лучше всего в шестнадцатиричном формате.

Рекомендуется предусмотреть специальные символы, инициирующие стирание всей информации, признак начала отображаемой последовательности, эффект мигания и т. п.

Сигнал с линии данных интерфейса RS-232 (его название TxD) лучше всего подавать на вход транзистора (см. схему в приложении). Он будет работать как преобразователь уровня входного сигнала.

### Задание 3. Частотомер

Устройство должно измерять временные параметры сигнала, такие как частота, период следования, уметь подсчитывать поступившее на этот вход число импульсов и отображать эту информацию на индикаторе.

Сигнал лучше всего подавать на вход транзистора (см. схему в приложении), тогда можно в разумных пределах почти не заботиться о его величине.

Какой из параметров сигнала должен измеряться в данный момент, можно задавать с помощью внешних кнопок, подключаемых через разъем к микроконтроллеру. Придумайте, как эту информацию отобразить одновременно с основной.

Алгоритмы измерения частоты (или периода) будут зависеть от соотношения частоты измеряемого сигнала и скорости, с которой работает микроконтроллер. Можно выделить два основных варианта.

В первом рассматривается очень высокая входная частота. Тогда микроконтроллер может просто работать как счетчик импульсов. Для этого удобно использовать внутренний таймер с программируемым коэффициентом деления. В этом случае можно говорить об измерении частоты сигнала.

Во втором варианте алгоритма контроллер должен распознавать моменты перехода сигнала ноль и измерять время между ними, то есть измеряется период. Для счета времени можно использовать, например, счет количества циклов. Поскольку процессор работает с известной частотой, то время выполнения каждого такого цикла легко определить.

Наиболее важный вопрос, на который необходимо ответить, заключается в том, как “сщить” эти два диапазона. Для этого надо выбрать некоторую пороговую частоту, выше которой устройство производит измерения, используя первый вариант, а ниже – второй. Правильно выбрать эту границу можно только исходя из критерия равных погрешностей для обоих вариантов измерения на границе диапазонов.

Оцените возможные диапазоны использования устройства при измерении всех вышеперечисленных параметров и точность, с которой эти измерения проводятся.

Для правильного отображения результата измерения необходимо, исходя из прогнозируемой вами точности, определить требуемое число дво-

ичных разрядов, а на основании этого и число отображаемых десятичных знаков.

Для выполнения арифметических преобразований двоичных чисел разрядностью более восьми вам необходимо разработать подпрограммы работы с такими числами, основываясь на базовых арифметических командах.

Кроме этого, вам потребуется программа преобразования двоичного представления в десятичное. Один из вариантов реализации такого алгоритма заключается в следующем.

Из двоичного числа вычитается число, соответствующее единице в старшем десятичном разряде (т. е. для 4-значного десятичного числа надо вычесть  $1000_{10}$ ). Если получится положительный результат, то увеличивается на единицу счетчик соответствующего разряда. Если результат отрицательный, то восстанавливается последнее положительное значение искомого числа и вычисления продолжаются вычитанием младшей единицы десятичного разряда (в нашем примере  $100_{10}$ ). Результат подсчета сохраняется уже в другом регистре. Далее подсчитывается следующий по значимости десятичный разряд и т. д., пока не дойдет до 1.

Для отображения используются получившиеся значения десятичных разрядов.

## Список литературы

1. Однокристальные микроконтроллеры Microchip: PIC16C8X / Пер. с англ. / Под ред. А. Н. Владимира. Рига: Ogmix, 1996.
2. Искусство схемотехники: В 3-х томах. / Пер. с англ. Хоровиц П., Хилл У. 4-е изд., перераб. и доп. М.: Мир, 1993.
3. PIC16C84 8-bit CMOS EEPROM Microcontroller. Microchip document. DS30445C DataSheet, 1997.
4. MPLab User's Guide. Microchip documents, DS51025A DataSheet, 1996.
5. MPSim User's Guide. Microchip documents, DS30027I DataSheet, 1995.

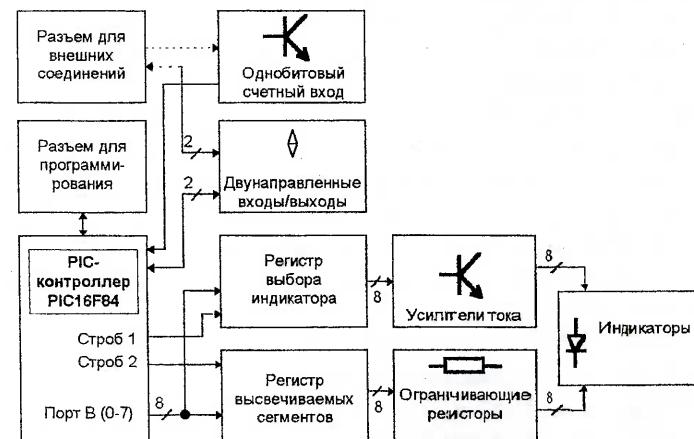
## Приложение 1. Описание и схемы учебной платы

### Назначение и состав устройства

Как уже отмечалось, плата используется в лабораторном практикуме по изучению архитектуры и принципов программирования микроконтроллеров. Работа с такой учебной (макетной) платой поможет получить навыки по практическому использованию микроконтроллеров в конкретных устройствах, программированию и отладке реальных программ.

Поскольку есть возможность разместить на плате дополнительные компоненты, то можно использовать ее для разработки других устройств, даже очень сильно отличающихся от рассматриваемого в этом описании. Более того, вместо микроконтроллера PIC16C8x фирмы MicroChip Technology вполне допустимо использовать и другие 18-выводные контроллеры этой же фирмы. При этом предварительно полезно провести анализ функционального назначения выводов микроконтроллера, чтобы максимально эффективно использовать как возможности платы, так и используемой микросхемы контроллера.

На рисунке приведена блок-схема устройства, которая поясняет основные принципы работы. В конце приложения приведена полная принципиальная схема.



Кроме самого микроконтроллера, на плате расположены два вспомогательных регистра 1554ИР23, один из которых служит для выбора индикатора, который активен в данный момент, а во второй записывается информация, соответствующая набору подсвечиваемых сегментов. Запись производится раздельно по сигналам **Строб 1** и **Строб 2** соответственно.

К выходу регистра выбора индикатора подключены транзисторные усилители, а выход регистра сегментов соединяется с индикаторами через ограничительные резисторы, которыми задается ток подсвечиваемого сегмента. Таким образом, чтобы высветить сегмент, надо его анод через ключ скоммутировать с положительным напряжением питания, а на катоде (т. е. на соответствующем выходе регистра сегментов) установить низкий уровень напряжения (логический 0). Устройство может обслуживать до восьми индикаторов.

Однобитовый счетный вход (простейший транзисторный преобразователь уровня) соединен с входом микроконтроллера, который можно запрограммировать как таймер (или счетчик). При необходимости его можно вывести на одну из ножек внешнего разъема и использовать для обработки внешних сигналов. На него можно подавать напряжение в достаточно широком диапазоне (по крайней мере от 2 до 20 В). Ограничения определяются тем, что входной транзистор, с одной стороны, должен надежно открываться и закрываться, а с другой - слишком большое напряжение может его повредить.

Два сигнала можно использовать и как входы, и как выходы. Надо только помнить, что уровни напряжений на них должны соответствовать логическим уровням сигналов, а направление передачи входного буфера 1554ЛП18, которое задается микропереключателями на плате, не должно противоречить запрограммированному направлению передачи соответствующих входов микроконтроллера.

Питание платы осуществляется через второй разъем. Основное питание 5 В подается на оба контакта разъема. Если на плате смонтированы дополнительные элементы, требующие питания +12 В или -12 В, то его можно подать через предусмотренные для этого дополнительные контакты.

Этот же разъем служит и для подключения программатора. Причем основное питание должно быть отключено. Для программирования используются следующие 5 сигналов:

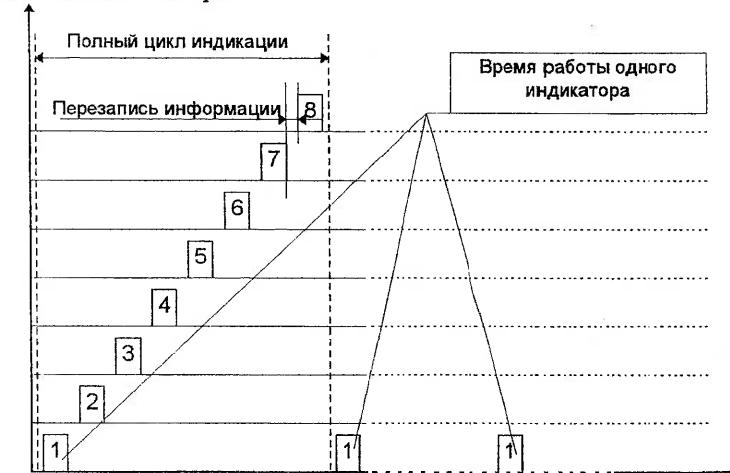
- Progr. Data – вход/выход передачи данных;
- Progr. CLC – вход синхронизации данных;
- MCLR/Vpp – вход подачи напряжения программирования и сброса;
- Progr. Vdd – вход подачи программирующего напряжения;
- GND – общий.

При подаче напряжения 12 В на вход MCLR/Vpp контроллер переходит в режим записи и происходит сброс внутренних регистров. Данные для программирования в последовательном виде передаются по линии Progr. Data и сопровождаются сигналом синхронизации Progr. CLC.

В схеме предусмотрены специальные меры для того, чтобы можно было программировать микроконтроллер, не вынимая его из платы. Поскольку величина сигнала MCLR может представлять опасность для остальной части схемы, то для защиты используется диод VD1, который предотвращает прохождение высокого напряжения на нее. Резисторы R2 и R3 также ограничивают воздействие подключенных к выводам программирования микросхем на процесс программирования. Необходимо обратить внимание, что при указанных на схеме номиналах в качестве регистров можно использовать только КМОП-микросхемы, поскольку только они обеспечивают небольшой ток и малое падение напряжения на указанных резисторах.

### Динамическая индикация

Основная идея динамической индикации сродни развертке в телевизоре. Это означает, что цикл полного отображения развернут во времени, и в каждый конкретный момент либо отображается только один выбранный индикатор, либо вообще ни одного. На рисунке показана временная диаграмма одного полного цикла отображения информации с использованием всех восьми индикаторов.



Интервал, когда все сегменты погашены, необходим для того, чтобы можно было изменить информацию о засвечиваемых сегментах (для сле-

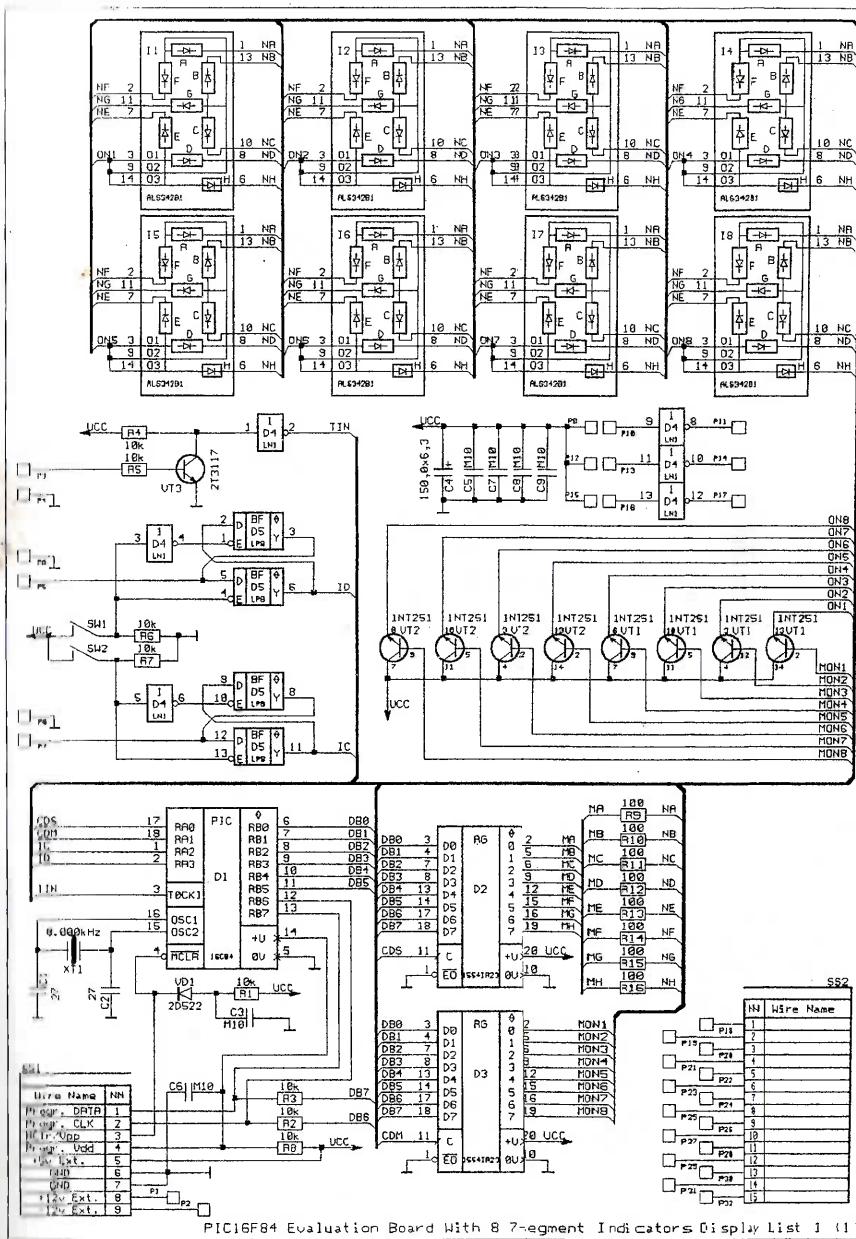
дующего индикатора) не вызывая паразитной засветки. Если этого не делать, то будет наблюдаться небольшое свечение сегментов, которые на самом деле не должны использоваться.

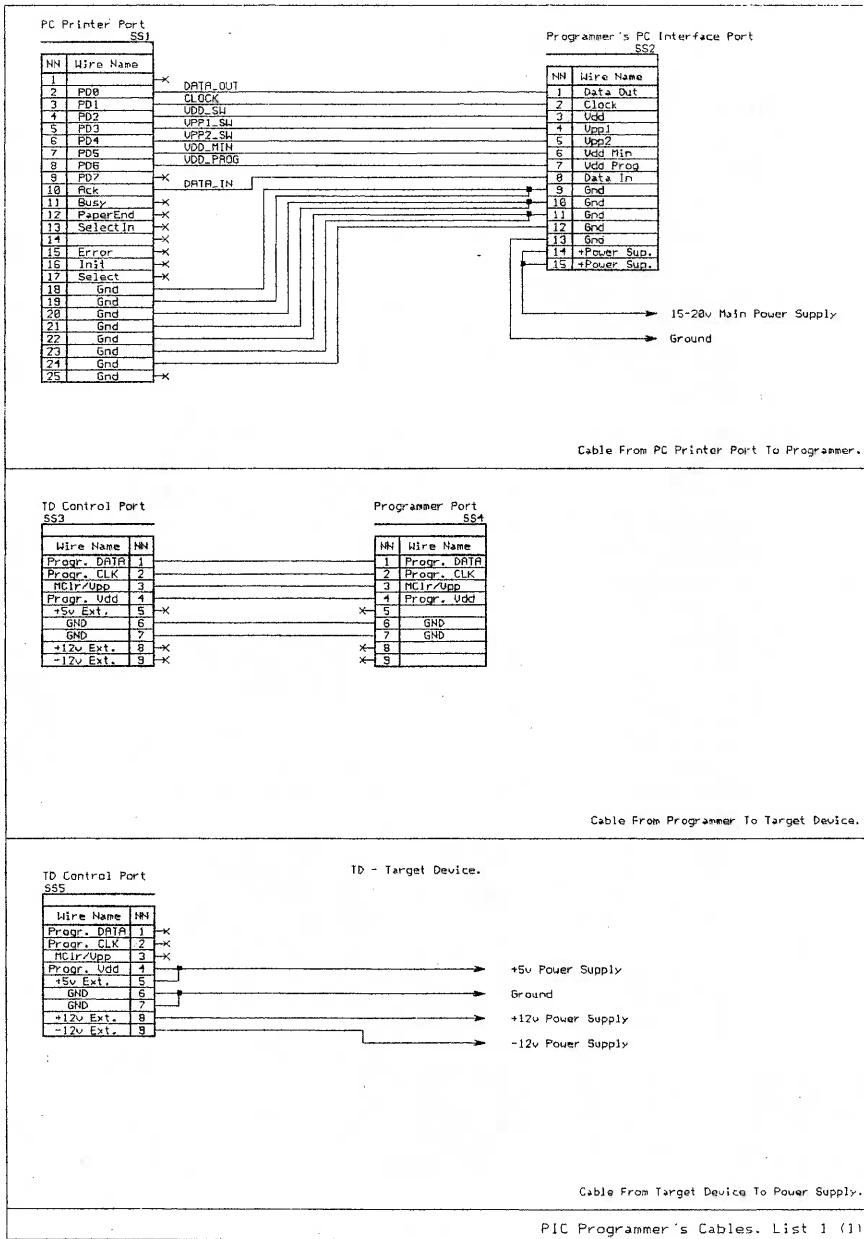
Важным параметром для обслуживания такого индикатора будет время полного отображения (регенерации). Из-за особенностей человеческого зрения это время не должно быть больше 20-30 мс (это как бы частота кадров телевизионной развертки). Поскольку предполагается использовать восемь индикаторов, то время высвечивания отдельного индикатора будет в восемь раз меньшей величиной.

С другой стороны, минимальное время регенерации ограничено производительностью микропроцессора. Если все выполняемые им действия сведутся только к обслуживанию индикатора, то весьма сомнительно, что такое устройство может быть чем-нибудь полезным.

Далее в Приложении 1 приводится принципиальная схема учебной платы и распайка соединительных кабелей.

В Приложении 2 приведен ассемблерный текст базовой программы, которую можно использовать как основу для написания других, расширенных приложений, где необходимо использование индикатора. Все особенности программирования описаны в комментариях к программе.





Распайка соединительных кабелей

## Приложение 2. Базовая программа обслуживания индикации

```

; Watch-84.asm
;
; Это демонстрационная программа, которая может использоваться как
; заготовка для написания других программ различного назначения.
;
; Программа отображает содержимое знакогенератора на линейке семи-
; сегментных индикаторов. В состав программы входит подпрограмма
; отображения буфера, которая инициируется примерно один раз в 10 мс. Кроме
; того, показано использование таблицы знакогенератора и способ доступа к
; этой таблице. Полезными могут оказаться и примеры программирования портов
; микроконтроллера и прерываний.
;
; Процедура прерываний от таймера декрементирует до единицы 4
; программных счетчика (T1Cnt0-T1Cnt3), которые можно использовать для
; задания квантов времени в программе.
;
; Для счета времени инициирования отображения используется T1Cnt0.
;
; Момент начала отображения определяется по достижению им единицы. На
; индикаторах одновременно отображается только часть таблицы знакогенерато-
; ра. С интервалом примерно 2.5 сек эта информация сдвигается, и таким
; образом сканируется вся таблица.
;
```

```

LIST
_CONFIG _PWRTE_OFF & _CR_OFF & _WDT_OFF & _XT_OSC
INCLUDE "p16f84.inc"
INCLUDE "Constants.inc"
;
```

```

; Биты Порта А:                                ; Биты стробирования регистра сегментов
CDS_St Equ 0                                     ; Бит стробирования регистра сегментов
CDM_St Equ 1                                     ; Бит стробирования регистра индикаторов
T_In Equ 4                                       ; Бит входа транзисторного ключа
                                                ; Системные константы:
CommaS Equ 16                                     ; Код запятой в таблице семисегментного знакогенератора
Desh_S Equ 17                                     ; Код тире в таблице семисегментного знакогенератора
RefTim_Equ 10+1                                   ; ПерIOD между пересчетом индикаторов
C_TimL_Equ 255                                    ; Время выдержки отображаемой информации (мл. байт)
C_TimH_Equ 8                                      ; Время выдержки отображаемой информации (ср. байт)

;-----; General Purpose Registers
;-----;

; Регистры обслуживания Индикации:
Ind1 Equ 0x0C                                     ; First Reg
Ind2 Equ 0x0D                                     ; 2 Reg
Ind3 Equ 0x0E                                     ; 3 Reg
Ind4 Equ 0x0F                                     ; 4 Reg
Ind5 Equ 0x10                                     ; 5 Reg
Ind6 Equ 0x11                                     ; 6 Reg
Ind7 Equ 0x12                                     ; 7 Reg
Ind8 Equ 0x13                                     ; 8 Reg
IndShf Equ 0x14                                   ; Регистр для сдвига текущего отображаемого индикатора

;-----; Свободные регистры
;-----;

; Переменные Программы:                         ; Счетчик времени для выдержки отображаемой информации
CngTil_Equ 0x25                                   ; Счетчик времени для выдержки отображаемой информации
CngIn_Equ 0x26                                     ; Счетчик времени для выдержки отображаемой информации
CngPtr_Equ 0x27                                   ; Указатель для пересылки рабочего регистра
Cngnt_Equ 0x28                                     ; Счетчик байт для пересылки отображаемой информации
TiCntr0_Equ 0x29                                   ; Декрементируемый до 1 таймерный счетчик
TiCntr1_Equ 0x2A                                   ; Декрементируемый до 1 таймерный счетчик

;-----; Program Code
;-----;

Org 0                                         Start
;-----; Interrupt Vector
;-----;

IntDev Org 4
;-----; Сохраняем регистры W и STATUS:
MovWF W_Temp                                     ; Сохраняем текущий до 1 таймерный счетчик
SwapF STATUS,W                                     ; Для сохранения регистра STATUS
MovWF S_Temp                                     ; Для сохранения регистра ESR
;-----; Работаем с таймером:
DecFSZ TiCntr0,W                                ; Декрементируем до 1 счетчик 0:
MovWF TiCntr0                                     ; Для сохранения регистра TiCntr0
DecFSZ TiCntr1,W                                ; Декрементируем до 1 счетчик 1:
MovWF TiCntr1                                     ; Для сохранения регистра TiCntr1
DecFSZ TiCntr2,W                                ; Декрементируем до 1 счетчик 2:
MovWF TiCntr2                                     ; Для сохранения регистра TiCntr2
DecFSZ TiCntr3,W                                ; Декрементируем до 1 счетчик 3:
MovWF TiCntr3                                     ; Для сохранения регистра TiCntr3
BCF INTCON,T0IF                                  ; Восстанавливаем флаг прерывания от таймера
SwapF S_Temp,W                                     ; Сбрасываем регистры W и STATUS
MovWF STATUS

```

```
SwapF W(Temp, F  
SwapF W(Temp, W  
RetFIE
```

```
; Tables
```

```
; Таблица Знакогенератора:
```

```
SegmTB AddWF PCL, F  
RetIW b'11000000' ;0.  
RetIW b'11111001' ;1.  
RetIW b'10100100' ;2.  
RetIW b'10110000' ;3.  
RetIW b'10011001' ;4.  
RetIW b'10010010' ;5.  
RetIW b'10000010' ;6.  
RetIW b'11111000' ;7.  
RetIW b'10000000' ;8.  
RetIW b'10010000' ;9.  
RetIW b'10001000' ;A.  
RetIW b'10000011' ;B.  
RetIW b'11000110' ;C.  
RetIW b'10100001' ;D.  
RetIW b'10000110' ;E.  
RetIW b'10001110' ;F.  
RetIW b'01111111' ;  
RetIW b'10111111' ;- запахая.  
;--  
SegmTE
```

```
; SubRoutines
```

```
; Программируем Порт А:
```

```
SetupA BCF STATUS, RPO  
Movlw b'00001100'  
Movwf TRISA  
BCF STATUS, RPO  
Clrf PORTA  
Retlw 0  
; Программируем Порт В:
```

```
SetupB BCF STATUS, RPO  
Movlw b'00000000'  
Movwf TRISB  
BCF STATUS, RPO  
Clrf PORTB  
Retlw 0  
; Программируем Таймер:  
; D0-D2, предварительный делитель на 32 (PSA0-PSA2),  
; D3, подключенный к WDT (PSA),  
; D4, инкремент по фронту 1/0 (TOSE),  
; D5, счетчик по внутреннему Tosc/4 (TOCS),  
; D6-D7, GFWU и GFRU не используются (оба 1).  
SetupT BCF STATUS, RPO ; Select Bank 0  
Clrf TMRO  
Bsf STATUS, RPO ; Select Bank 1  
ClrwDT Movlw b'11011100'  
Movwf OPTION_REG ; Load OPTION  
BCF STATUS, RPO ; Select Bank 0  
Retlw 0  
; Программируем Систему Прерываний:
```

```
IntInit BCF INTCON, TOIE ; Сбрасываем флаг прерывания от таймера  
BSF INTCON, TOIE ; Разрешаем прерывание от таймера
```

```

BSF INTCON, GIE ; Разрешаем общее прерывание
Retlw 0

; Инициализация Декрементируемых Таймерных Счетчиков:

CntIni
Movlw 1
Movwf TiCnt0
Movwf TiCnt1
Movwf TiCnt2
Movwf TiCnt3
Retlw 0
; Строб записи (высоким уровнем) В регистр сегментов:

```

```

CDS_On BSF PORTA, CDS_St
Nop
BCF PORTA, CDS_St
Retlw 0
; Строб записи (высоким уровнем) В регистр выбора индикатора:
CDM_On BSF PORTA, CDM_St
Nop
BCF PORTA, CDM_St
Retlw 0
; Обновление исходной отображаемой информации:
Changi
; Сохраняем FSR:
Movfw FSR
Movwf F(Temp
; Инициализируем счетчик байт:
Clrf CngCnt
; Инициализируем указатель на начало передаваемой области:
Movlw Ind1
Movwf FSR
; Записываем 8 байт из знакогенератора в буфер отображения:
C_L1 Movfw CngCnt

```

```

; Сохраняем FSR:
Movfw FSR
Movwf F(Temp
; Инициализируем счетчик байт:
Clrf CngCnt
; Инициализируем указатель на начало передаваемой области:
Movlw Ind1
Movwf FSR
; Записываем 8 байт из знакогенератора в буфер отображения:
C_L1 Movfw CngCnt

```

```

; Инициализация таймера:
Call SegInitD
Movwf INDF
IncF ESR,F
IncF CngCnt,F
Movlw 8
Subwf CngCnt,W
Btfss STATUS,Z
Goto C_L1
; Иcrementируем или обнуляем указатель в знакогенераторе:
IncF CngPtr,F
Movlw SegmE-SegmB-8
Subwf CngPtr,W
Btfsc STATUS,Z
Clrf CngPtr
; Обнуляем указатель в таблицу
; Запускаем новый счет времени обновления:
Movlw C_TimL
Movwf CngTil
Movlw C_TimH
Movwf CngTih
; Восстановливаем FSR:
Movfw F(Temp
Movwf FSR
Retlw 0
; Инициализация указателей индикатора:
RefIni
; Иницируем указатель данных отображаемого индикатора:
Movlw Ind1
Movwf FSR
; Иницируем регистр отображаемого индикатора:
Movlw b 00000001
Movwf IndShf
Retlw 0
; Смена отображаемого индикатора:

```

```

Refrsh      ; Все гасим:
Clrw
MovWF PORTB
Call CDM_On
; Пишем подсвечиваемые семменты:
MovFW INDF
MovWF PORTB
Call CDS_On
; Пишем подсвечиваемый индикатор:
MovFW IndShf
MovWF PORTB
Call CDM_On
; Готовим отображение следующего индикатора:
IncF FSR, F
BCF STATUS, C
RLF IndShf, F
BTFS C STATUS, C
Call Refini, 5 ; Если все 8 отработали, то начинаем с Ind1
; Готовим следующий временной интервал:
MovIW RefTim
MovWF TiCntr0
Retlw 0
;
```

---

```

; Main Program
;
```

```

Start      Call SetupA ; Программируем порт А
Call SetupB ; Программируем порт В
Call CntIni ; Инициализируем лекрементные счетчики
Call ChangI ; SetInd ; Инициализируем временный интервал
;
```

```

Глава 7. Программирование на языке FCBM
Call SetupT ; Программируем таймер
Call Intini ; Инициализируем прерывания
Clrf CngPtr ; Обнуляем указатель в таблицу
Movlw C_TimL ; Запускаем счет времени обновления:
Movwf CngTil
Movlw C_TimH
Movwf CngTih
Loop      DecFSZ TiCntr0, W ; Надо обновлять индикацию?
Goto L1   call Refrsh ; Светим следующий индикатор
DecFSZ CngTil, F ; Надо обновлять информацию?
GoTo L1   DecFSZ CngTih, F ; Надо обновлять информацию?
Goto L1   Call ChangI ; Обновляем отображаемую информацию
L1       GoTo Loop ; Start
;
```

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРОВ PIC16C84 (PIC16F84) .....	8
Описание системы команд .....	11
КРАТКИЕ СВЕДЕНИЯ О СРЕДЕ РАЗРАБОТКИ ПРОГРАММ MPLAB.....	33
Обзор инструментария MPLab.....	33
Проект. Начало работы .....	34
Основные функции отладчика.....	35
Специальные окна.....	37
Функции внешних воздействий .....	38
Расширения файлов, используемые MPLab.....	38
ОСНОВЫ РАБОТЫ С АССЕМБЛЕРОМ MPASM .....	39
Файлы, используемые ассемблером .....	41
Директивы ассемблера .....	42
Синтаксис выражений и операций ассемблера .....	49
Текстовые последовательности .....	50
Числовые константы и RADIX .....	51
РАБОТА С ПРОГРАММАТОРОМ .....	53
Программа управления программатором.....	53
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ .....	55
Вопросы и задания для первого знакомства .....	55
Задания для проектирования реальных устройств .....	58
Задание 1. Часы.....	59
Задание 2. Индикатор с приемом информации по RS-232.....	59
Задание 3. Частотомер .....	60
СПИСОК ЛИТЕРАТУРЫ.....	62
ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ И СХЕМЫ УЧЕБНОЙ ПЛАТЫ .....	63
Назначение и состав устройства .....	63
Динамическая индикация.....	65
ПРИЛОЖЕНИЕ 2. БАЗОВАЯ ПРОГРАММА ОБСЛУЖИВАНИЯ ИНДИКАЦИИ.....	69

*Юрий Михайлович Прокопьев*

## АРХИТЕКТУРА И ПРОЕКТИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

**Практикум по проектированию микроконтроллеров  
на примере компонентов фирмы Microchip Technology**

Учебное пособие

Редактор Е. И. Лукацук

Подписано в печать 19.07.99 г.  
Формат 60×84 1/16. Офсетная печать.  
Уч.-изд. л. 5. Тираж 150 экз.  
Заказ № 330

Лицензия ЛР №021285 от 6 мая 1998 г.  
Издательский центр НГУ  
630090, Новосибирск-90, ул. Пирогова, 2.