ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Г.Р.Алпатов

Применение PIC-контроллеров в измерительной технике

Учебно-методическое пособие

Ростов-на-Дону

2008

Алпатов Г.Р. Применение PIC-контроллеров в измерительной технике. Ростов-на-Дону, 2008

Учебно-методическое пособие предназначено для студентов, обучающихся по направлению подготовки 200100 Приборостроение. В нем рассматриваются аппаратное устройство микроконтроллеров фирмы МикроЧип (MicroChip), основы программирования их на языке ассемблер и языках высокого уровня. Приводятся примеры программ, описание стенда для их отладки и описание среды разработки MpLab.

ОГЛАВЛЕНИЕ

Введение

Развитие микроэлектроники и широкое применение ее изделий в промышленном производстве, в быту, в устройствах и системах управления самыми разнообразными объектами и процессами является в настоящее время одним из основных направлений научно-технического прогресса.

Использование микроконтроллеров (однокристальных ЭВМ) в изделиях не только приводит к повышению технико-экономических показателей (точности, стоимости, надежности, потребляемой мощности, габаритных размеров), но и позволяет сократить время разработки изделий и делает их модифицируемыми, адаптивными.

Широко распространены микроконтроллеры семейства MCS-51 фирмы INTEL, SIGNAL, PIC-контроллеры фирмы Microchip.

Выше перечисленные семейства микроконтроллеров значительно отличаются друг от друга системой команд, наличием периферийных устройств, вычислительной мощностью, быстродействием, удобством применения. В этом ряду занимают высокое место для устройств среднего класса требований микроконтроллеры фирмы MicroChip, среди которых можно подобрать нужный тип практически для любой задачи. Эти качества PIC-контроллеров позволили фирме Microchip стать одной из лидирующих компаний в своей области. Так если в 1990 году по объему продаж она была на 20-ом месте, в 1993-м – на 8-ом, то в 1997 году, несмотря на сильнейшую конкуренцию, на 2-ом месте [1]. Это положение в значительной степени сохраняется и в настоящее время.

Все это определяет необходимость изучения микроконтроллеров, в частности микроконтроллеров фирмы MicroChip на ФВТ, для которых на факультете имеется лабораторная установка, позволяющая получить практические навыки в программировании микроконтроллеров для измерения напряжения, тока, частоты, периода, мощности, температуры, компонентов RLC, времени, электроэнергии, формы сигнала. Слушатели курсов знакомятся также с популярными интерфейсами: RS232, RS435, I2C, 1-WIRE, SPI.

При работе с методическим пособием подразумевается знакомство с курсами «Измерительная техника», «Вычислительная техника», «Работа на компьютере IBM PC».

Итак, Вы в первый раз пришли на лабораторные занятия по курсу «Применение PIC-контроллеров в измерительной технике». С чего же начать?

1. Включите питание управляющей ЭВМ и стенда для программирования PIC-контроллеров.

2. Запустите приложение «MpLab574.exe» и откройте свой первый проект, который ранее создан в системе [3]. Для этого:

3. Выберите пункт меню «Project» вверху окна, далее - «Open Project», откройте папку «…\lab_rab00» и в ней откройте проект «lab_rab00.prg». Если все сделано правильно и указанный проект существует, должны открыться четыре окна проекта: «…\lab_rab00.asm», «File Register Window», «Special File Register Window» и «Stack Window».

4. Выполните компиляцию проекта. Для этого выберите пункт меню «Project», затем «Make Project». Если в проекте нет ошибок, то во время компиляции на короткое время появляются окошки, отражающие зеленым цветом нормальный ход компиляции или красным цветом, если в проекте присутствуют ошибки. Для успешной компиляции все ошибки должны быть устранены.

5. После успешной компиляции в папке «…\lab_rab00» должен появиться файл «lab_rab00.hex», предназначенный для программирования микроконтроллера стенда. Чтобы запрограммировать PIC-контроллер нужно закрыть приложение «MpLab574.exe» и открыть программное обеспечение программатора стенда, которое называется «IcProg.exe». Затем в меню «Файл», подменю «Открыть файл» открыть файл «…\lab_rab00\lab_rab00.hex». В окнах программатора должна появиться информация проекта. Далее нужно выбрать в меню «Команды» подменю «Программировать Все». PIC-контроллер должен запрограммироваться за несколько секунд, при этом ПО программатора должно визуально показывать процесс программирования и сообщить результаты программирования – успешно или с ошибками.

6. В случае успешного программирования необходимо выйти из ПО программатора и нажать кнопку «Сброс» на лицевой панели стенда. Программа, записанная в PIC-контроллере должна выполняться и выполнить свои действия, а именно вывести на ЖКИ индикатор сообщение «Hello, Word!».

Мы видим, что PIC-контроллер запрограммирован и работает. Но что это за таинственные обозначения w, tmr0, option_reg, pcl, pclath и все остальные в правой верхней части экрана? Каков смысл команд clrf, bsf, bcf, movlw и других команд ассемблера? Как вообще научиться программированию PIC-контроллеров на языке ассемблер и других языках?

Ответы на эти и другие вопросы даст Вам изучение методического пособия, которое Вы держите в руках.

МОДУЛЬ 1. АППАРАТНОЕ УСТРОЙСТВО PIC-КОНТРОЛЛЕРОВ

Комплексная цель изучения модуля состоит в знакомлении с аппаратным устройством PIC-контроллеров и законченных изделий на их основе, с системой команд на примере конкретного изделия.

1.1. Особенности архитектуры PIC-контроллера

С точки зрения пользователя, необходимыми компонентами каждого микроконтроллера являются:

- центральный процессор;
- память данных и память программ;
- тактовый генератор и таймер;
- порты ввода-вывода;
- схема сброса.

Могут также быть:

- схема прерываний;
- аппаратные модули, работающие без участия центрального процессора.

Память данных и память программ во всех микроконтроллерах PIC разделены (Гарвардская архитектура). Присутствует шина данных, которая имеет разрядность 8 бит и разделена с шиной адреса, которая соединяет центральный процессор с памятью программ. В результате процессор в состоянии одновременно выполнять доступ к данным и к словам команд. Также эта архитектура допускает, чтобы разрядность ячеек в памяти программ не зависела от разрядности ячеек в памяти данных. Кроме этого одновременно с выполнением команды центральный процессор выбирает из памяти программ следующую команду. Этим обработка ускоряется почти в два раза при одной и той же тактовой частоте. Только в том случае, когда обрабатываемая в данный момент команда является командой перехода, подготовленная команда отбрасывается и выбирается команда по новому адресу перехода.

Разрядность команд: 12, 14 и 16 бит. В каждой команде требуется только одно обращение к шине адреса. Для этого требуется хорошо продуманный, сокращенный набор команд (примерно 35 команд). Разрядность памяти программ в первом поколении микроконтроллеров PIC составляет 12 бит. Эти устройства, называемые «базовой серией» (base-line) – быстрые, хотя с современной точки зрения, довольно «спартанские». Среднее подсемейство (mid-range) с разрядностью 14 бит представляет уже заметно больше комфорта. Удлинение слова команды используется преимущественно в пользу расширения диапазона адресов, поэтому сам набор команд в сравнении с базовой серией микроконтроллеров PIC не изменился. Именно из этой серии используются PIC-

контроллеры в учебном стенде (PIC16F870[4], PIC16F873[5] или PIC16F876). Наиболее современное старшее подсемейство (enhanced) имеет 16-ти разрядные слова команд, которые используются не только для дальнейшего расширения адресного пространства, но также и для увеличения набора команд. Общее количество типов микроконтроллеров PIC более двухсот[7].

Остановимся подробнее на серии PIC16F870, PIC16F873, PIC16F876. Цена в розницу PIC16F870-5$, PIC16F873-8,5$ (для сравнения PIC16F84-4,5$). Основные параметры этих контроллеров приведены в таблице 1, а назначение выводов - в таблице 2.

Таблица 1. Параметры контроллеров серии PIC16F870, 3, 6.

Параметр	PIC16F870	PIC16F873	PIC16F876	PIC18F252	Примечание
Макс. тактовая частота, МГц	20	20	20	40	
Сброс (задержка сброса)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	
Флэш-память программ	2K×14	4K×14	8K×14	16K×16	
Кол-во банков памяти программ	1	2	4		
Память данных	128	192	368	1536	
Кол-во банков памяти данных	1	2	4		
EEROM	64	128	256	256	
Модуль таймеров	1-16 bit 2-8 bit WDT	3-16 bit 1-8 bit WDT	3-16 bit 1-8 bit WDT	3-16 bit 1-8 bit WDT	
Векторы прерываний	1	1	1	2	
Кол-во АЦП	5(10 bit)	5(10 bit)	5(10 bit)	5(10 bit)	
Модуль ШИМ, 10 бит	1	1	1	2	
Serial I/O	USART	USART, MI2C/SPI	USART, MI2C/SPI	USART, MI2C/SPI	

Таблица 2. Назначение выводов контроллеров серии PIC16F870, 3, 6.

Обозначение	№	Тип	Тип буфера	Назначение
OSC1/CKLIN	9	I	ST/CMOS	Вход для подключения кварцевого резонатора, либо RC-цепочки, либо вход для внешнего тактового генератора.
OSC2/CKLOUT	10	O		Выход для подключения кварцевого резонатора в режиме работы с кварцем, в режиме RC-генератора на выходе присутствуют импульсы с частотой 1/4 от OSC1.
-MCLR/Vpp	1	I/P	ST	Вход сброса микроконтроллера или вход напряжения программирования. Сброс микроконтроллера происходит при низком логическом уровне на входе.
				Двунаправленный порт ввода/вывода PORTA.
RA0/AN0	2	I/O	TTL	RA0 может быть настроен как аналоговый канал 0.
RA1/AN1	3	I/O	TTL	RA1 может быть настроен как аналоговый канал 1.

RA2/AN2/Vref-	4	I/O	TTL	RA2 может быть настроен как аналоговый канал 2 или вход отрицательного опорного напряжения.
RA3/AN3/Vref+	5	I/O	TTL	RA3 может быть настроен как аналоговый канал 3 или вход положительного опорного напряжения.
RA4/T0CKI	6	I/O	ST	RA4 может использоваться в качестве входа внешнего тактового сигнала для TMR0. Выход с открытым стоком.
RA5/-SS/AN4	7	I/O	TTL	RA5 может быть настроен как аналоговый канал 4 или вход выбора микросхемы в режиме ведомого SPI.
				Двунаправленный порт ввода/вывода PORTB. PORTB имеет программно подключаемые подтягивающие резисторы на входах.
RB0	21	I/O	TTL/ST	RB0 может использоваться в качестве входа внешних прерываний.
RB1	22	I/O	TTL	Нет особенностей.
RB2	23	I/O	TTL	Нет особенностей.
RB3	24	I/O	TTL	RB3 может использоваться в качестве входа для режима низковольтного программирования.
RB4	25	I/O	TTL	Прерывания по изменению уровня входного сигнала.
RB5	26	I/O	TTL	Прерывания по изменению уровня входного сигнала.
RB6	27	I/O	TTL/ST	Прерывания по изменению уровня входного сигнала или вывод для режима внутрисхемной отладки ICD. Тактовый вход в режиме программирования.
RB7	28	I/O	TTL/ST	Прерывания по изменению уровня входного сигнала или вывод для режима внутрисхемной отладки ICD. Вывод данных в режиме программирования.
				Двунаправленный порт ввода/вывода PORTC.
RC0	11	I/O	ST	RC0 может использоваться в качестве выхода генератора TMR1 или входа внешнего тактового сигнала для TMR1.
RC1	12	I/O	ST	RC1 может использоваться в качестве входа генератора для TMR1 или вывода модуля CCP2.
RC2	13	I/O	ST	RC2 может использоваться в качестве вывода модуля CCP1.
RC3	14	I/O	ST	RC3 может использоваться в качестве входа/выхода тактового сигнала в режиме SPI и I2C.
RC4	15	I/O	ST	RC4 может использоваться в качестве входа данных в режиме SPI или входа/выхода данных в режиме I2C.
RC5	16	I/O	ST	RC5 может использоваться в качестве выхода данных в режиме SPI.
RC6	17	I/O	ST	RC6 может использоваться в качестве вывода передатчика USART в асинхронном режиме или вывода синхронизации USART в синхронном режиме.
RC7	18	I/O	ST	RC7 может использоваться в качестве вывода приемника USART в асинхронном режиме или вывода данных USART в синхронном режиме.
Vss	8, 19	P	-	Общий вывод для внутренней логики и портов ввода/вывода.
Vdd	20	P	-	Положительное напряжение питания для внутренней логики и портов ввода/вывода.

Обозначения: I=вход, O=выход, I/O=вход/выход, P=питание, - = не используется, TTL=входной буфер, ST=вход с триггером Шмидта.

1.2. Аппаратные модули PIC-контроллеров

Рассмотрим аппаратное устройство микроконтроллеров PIC по отдельным узлам, сначала рассмотрим типичный порт ввода-вывода (рисунок 1), а узел АЦП, таймера TMR1 рассмотрим позднее. Узлы, которые не применяются в этом пособии предлагается изучить самостоятельно по технической документации.

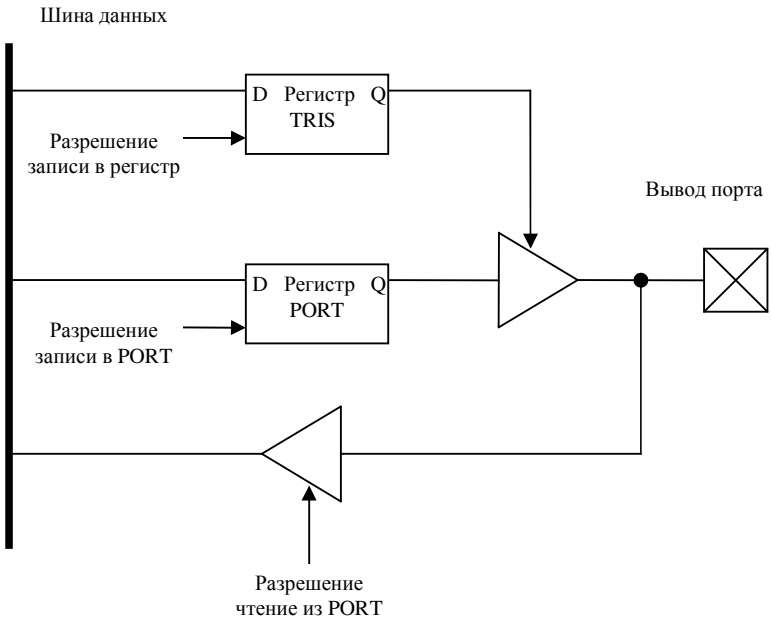


Рисунок 1. Упрощенная схема порта ввода-вывода

Запись единицы в управляющий регистр (TRIS) отключает выходной буфер, поэтому можно читать информацию на выводе порта. Запись нуля в управляющий регистр (TRIS) включает выходной буфер, который передает данные из регистра порта (PORT) на вывод. Здесь есть некоторые тонкости: некоторые команды внутренне выполняются как чтение+запись. Например,команды BCF и BSF считывают порт целиком, модифицируют один бит и выводят результат обратно. Здесь необходима осторожность. Например, команда BSF для бита 5 регистра f6 (порт B) сначала считает все восемь бит. Затем выполняются действия над битом 5 и новое значение байта целиком записывается в выходные защелки. Если другой бит регистра f6 используется в качестве двунаправленного ввода/вывода (скажем бит 0) и в данный момент он определен как входной, входной сигнал на этой ножке будет считан и записан обратно в выходную защелку этой-же ножки, затирая ее предыдущее состояние. До тех пор пока эта ножка остается в режиме ввода, никаких проблем не возникает. Однако, если позднее линия 0 переключится в режим вывода, ее состояние будет неопределенным.

Мы специально подробно остановились на этой тонкости в работе PIC-контроллеров чтобы подчеркнуть необходимость тщательного изучения технической документации.

PIC16F870 имеет пять каналов аналого-цифрового преобразователя (AD). AD преобразование 10-ти разрядное, осуществляется методом последовательного приближения, на время преобразования уровень входного сигнала удерживается устройством выборки-хранения (рисунок 2). Источник опорного напряжения задается программно. Внутренним источником является положительное напряжение питания устройства (Vdd), внешний источник подключается к контакту RA3/AN3/Vref. Модуль AD состоит из программно доступных регистров:

- результата ADRESH, ADRESL;
- управления 0 ADCON0;
- управления 1 ADCON1.

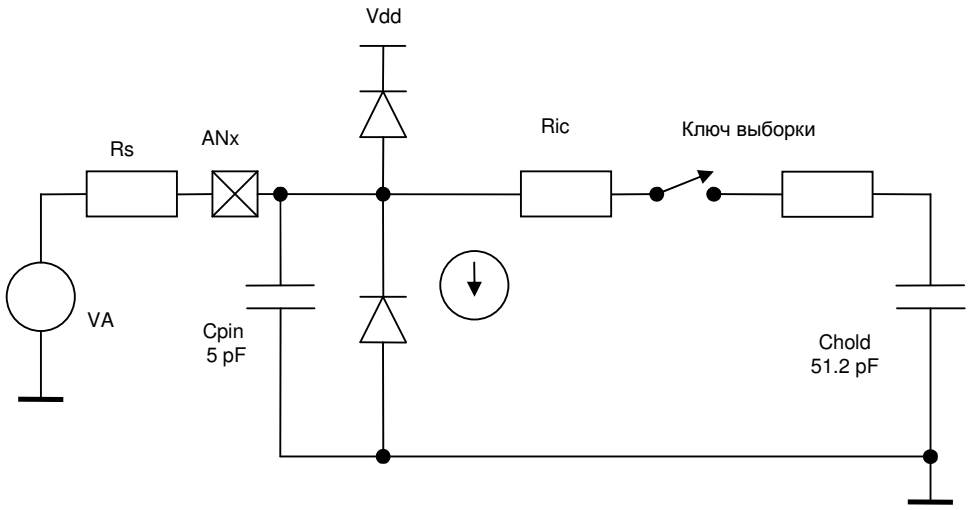


Рисунок 2. Аналоговая модель устройства выборки-хранения

Для того чтобы обеспечить точность AD преобразователя, необходимо полностью зарядить конденсатор хранения (Chold) напряжением входного сигнала. Внутреннее сопротивление источника сигнала (Rs) и сопротивление ключа, осуществляющего выборку, определяют время зарядки конденсатора Chold [8]. Рекомендуется использовать источники сигнала с Rs не более 10 кОм.

Модуль таймера TMR1 - это 16 разрядный таймер/счетчик, состоящий из двух 8 разрядных регистров TMR1H и TMR1L, которые доступны по чтению и по записи (рисунок 3). Пара регистров TMR1H+TMR1L инкрементируется от 0000H до FFFFH и при переполнении переходит в 0000H. Прывание TMR1 генерируется при переполнении. Для применения на нашем стенде важно, что он может работать работать в режиме 16-ти разрядного асинхронного счетчика.

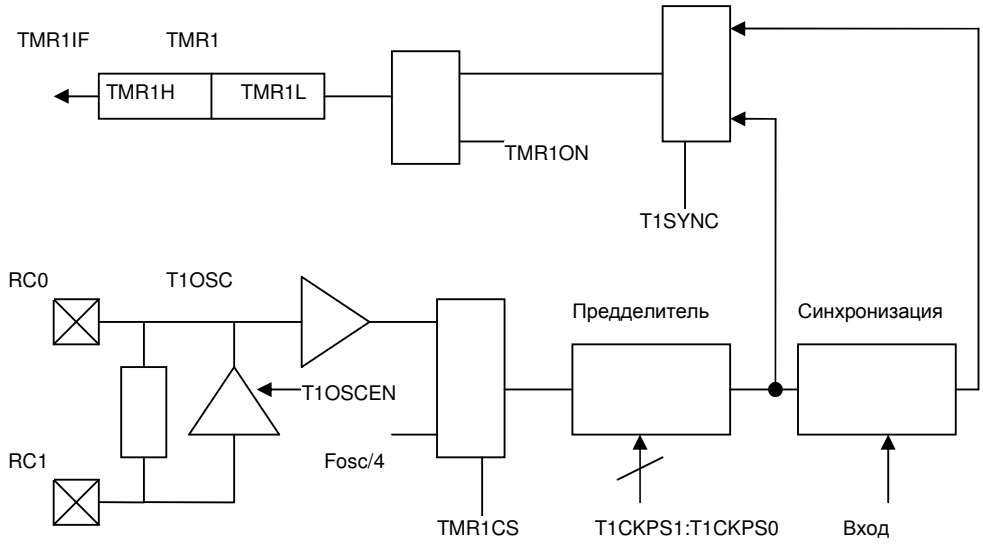


Рисунок 3. Структурная схема таймера TMR1

Регистры TMR1H и TMR1L можно читать во время, когда таймер работает в асинхронном режиме от внешнего генератора (обеспечивается аппаратными средствами [4]. Для записи рекомендуется просто остановить таймер и записать желаемое значение.

1.3. Схемотехника применения PIC-контроллеров

Что бы изделие на основе PIC-контроллеров правильно работало, нужно соблюдать некоторые правила: обеспечение качественным питанием, уменьшение электромагнитные помехи на его выводах, не допускать перегрузкок и т.д. [9]. Остановимся подробнее на некоторых моментах. Необходимо ставить керамический конденсатор 0,1 мкФ вблизи выводов питания (рисунок 4), добавлять последовательно резистор для защиты от перегрузки резонаторов с низким уровнем возбуждения (рисунок 5).

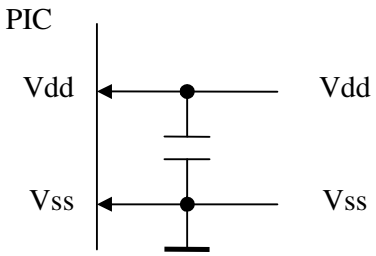


Рисунок 4. Керамический конденсатор вблизи выводов питания.

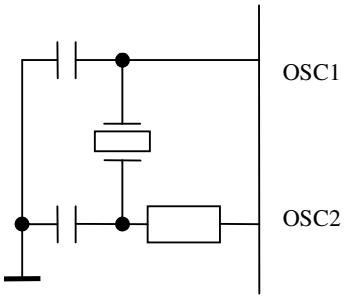


Рисунок 5. Подключение кварцевого резонатора.

На рисунке 6 приведена типовая схема внешнего сброса. Данная схема необходима, если скорость нарастания напряжения питания недостаточна (менее 0,05 В/мс). Диод используется для разряда конденсатора при выключении питания. Резистор может иметь значение от100 до 1000 Ом и предназначен для ограничения тока по выводу MCLR’ при перезаряде конденсатора.

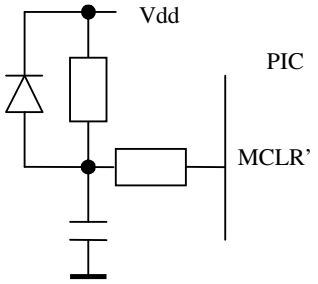


Рисунок 6. Схема внешнего сброса.

На рисунке 7 показано подключение буферногоусилителя для аналоговых измерений. Операционный уилитель нужно брать с однополярным питанием и работой входа/выхода от потенциала земли до потенциала питания.

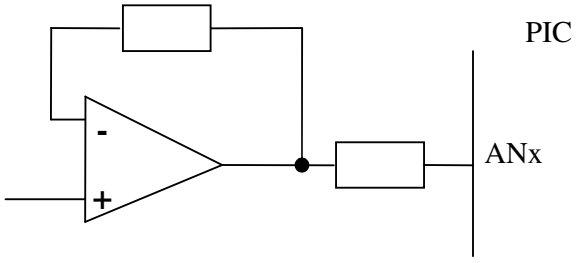


Рисунок 7. Подключение буферного усилителя.

1.4. Команды микроконтроллеров PIC

Каждая команда состоит из двух компонентов: кода операции и аргументов. Аргументы могут быть константами, адресами или прочими параметрами. Оба компонента заключены в одном слове команды, разряды которого используются оптимально, поскольку отсутствует выделение строго определенного количества разрядов под код операции и аргументы. Так, в некоторых командах аргументы очень длинные, в то время как в других командах их вообще нет. В качестве примера рассмотрим команды, выполняющие логические и арифметические операции с аргументом данных и рабочим регистром W. Этот класс команд имеет следующую структуру:

12 разрядов	OOOOO,D,FFFF
14 разрядов	OOOOO,D,FFFFFF
16 разрядов	OOOOOO,D,FFFFFFFF

При этом «O» обозначает код операции, «D» - селектор регистра назначения, а «F» - разряды указателя на аргумент данных.

Очень удобно в этих командах то, что пунктом назначения для результата операции может быть, на выбор, регистр W или указанный в аргументе файловый регистр. В языке ассемблера, поддерживаемом компанией Microchip, такая команда записывается следующим образом:

```
ADDWF      WERT,W
```

Этой командой содержимое ячейки памяти с именем WERT суммируется с рабочим регистром W и результат помещается в рабочий регистр W.

Всего в системе команд среднего подсемейства PIC-контроллеров 35 кодов операции, есть все что требуется для программирования:

Таблица 3. Команды с файловым регистром F в качестве аргумента

Команды+операнды	Операция	Флаги состояния	Примечания
ADDWF F,D	Add (W) and (F) (D)=(W) + (F)	C, DC, Z	Здесь примеры применения
ANDWF F,D	AND W with F (D)=(W) AND (F)	Z	
CLRF F	Clear (F) (F)=0	Z	
CLRW	Clear (W) (W)=0	Z	
COMF F,D	Complement (F) (D)=NOT (F)	Z	
DECF F,D	Decrement (F) (D)=(F) - 1	Z	
DECFSZ F,D	Decrement (F), Skip if 0 (D)=(F) - 1, пропускаем следующую команду, если Z=1	не изменяются	
INCF F,D	Increment (F)	Z	

	(D)=(F) + 1		
INCFSZ F,D	Increment (F), Skip if 0 (D)=(F) + 1, пропускаем следующую команду, если Z=1	не изменяются	
IORWF F,D	Inclusive OR (W) with (F) (D)=(W) OR (F)	Z	
MOVF F,D	Move f (D)=(F)	Z	
MOVWF F,D	Move W to f (F)=(W)	не изменяются	
NOP	No Operation Нет операции	не изменяются	
RLF F,D	Rotate Left (F) through Carry Циклический сдвиг влево через C	C	
RRF F,D	Rotate Right (F) through Carry Циклический сдвиг вправо через C	C	
SUBWF F,D	Subtract (W) from (F) (D)=(F) - (W)	C, DC, Z	
SWAPF F,D	Swap nibbles in (F) Переставить полубайты в (F) (D)=(F)	не изменяются	
XORWF F,D	Exclusive OR W with f (D)=(W) XOR (F)	Z	

Таблица 4. Команды с адресами разрядов в качестве аргументов; **B=0…7**

Команды+операнды	Операция	Флаги состояния	Примечания
BCF F,B	Bit Clear (F) Очистить бит B в (F)	не изменяются	
BSF F,B	Bit Set (F) Установить бит B в (F)	не изменяются	
BTFSC F,B	Bit Test (F), Skip if Clear Пропустить следующую команду, если бит B=0 в (F)	не изменяются	
BTFSS F,B	Bit Test (F), Skip if Set Пропустить следующую команду, если бит B=1 в (F)	не изменяются	

Таблица 5. Команды с константами, команды перехода и другие

Команды+операнды	Операция	Флаги состояния	Примечания

ADDLW K	Add literal and W (W)=(W)+K	C, DC, Z	
ANDLW K	AND literal with W (W)=(W) AND K	Z	
CALL ADDR	Call subroutine Вызов подпрограммы	не изменяются	
CLRWDT	Clear Watchdog Timer Сброс сторожевого таймера	TO, PD	
GOTO ADDR	Go to address Переход по адресу	не изменяются	
IORLW K	Inclusive OR literal with W (W)=(W) OR K	Z	
MOVLW K	Move literal to W (W)=K	не изменяются	
RETFIE	Return from interrupt Выход из подпрограммы обработки прерывания	GIE	
RETLW K	Return with literal in W Возврат с (W)=K	не изменяются	
RETURN	Return from Subroutine Возврат	не изменяются	
SLEEP	Go into standby mode Переход в спящий режим	TO, PD	
SUBLW K	Subtract W from literal (W)=K – (W)	C, DC, Z	
XORLW K	Exclusive OR literal with W (W)=(W) XOR K	Z	

Примечание: скобки означают содержимое того регистра, имя которого заключено в скобки

Команды очень хорошо документированы в фирменных технических описаниях, где можно найти даже коды операций.

В описании команд следует всегда обращать внимание на колонку, в которой отмечены флаги, изменяемые в результате выполнения той или иной команды. Флаги находятся в регистре состояния STATUS.

ПРОЕКТНОЕ ЗАДАНИЕ К МОДУЛЮ 1

1. Через токоограничевающий резистор к одному из выводов PIC-контроллера подключен оптрон. Рабочий ток оптрона 10 мА, при этом падение напряжения составляет 1,8 В. Определить значение резистора R.

2. Необходимо подключить память для расширения ОЗУ типа SPI. К каким выводам PIC-контроллера это лучше сделать. Нарисуйте схему подключения.

ТЕСТОВОЕ ЗАДАНИЕ К МОДУЛЮ 1

Тестовое задание выполняется в течении 5 минут и содержит 7 вопросов (правильный ответ помечается любым значком).

Результаты теста оцениваются по следующей шкале:

7 или 6 правильных ответов – оценка «5»,

5 правильных ответов – оценка «4»,

4 правильных ответа – оценка «3»,

менее 4 правильных ответов – оценка «2».

1. Количество выводов PIC-контроллера PIC16F870 в корпусе DIP:
   а) 18
   б) 24
   в) 28
   г) 30
2. Количество разрядов шины данных PIC-контроллера PIC16F870:
   а) 4
   б) 8
   в) 16
   г) 32
3. На какой вывод PIC-контроллера типа PIC16F870 подается сигнал СБРОС:
   а) 1
   б) 2
   в) 3
   г) 4
4. Какая из нижеследующих команд производит сложение рабочего регистра с константой:
   а) ADDWF TEMP,W
   б) ADDLW TEMP
   в) INCF TEMP,F
   г) ADDWF TEMP,F
5. Какая из нижеследующих команд перешлет данные из переменной TEMP в регистр W, не изменяя флаги в регистре STATUS:
   а) SWAPF TEMP,W
   б) MOVF TEMP,W
   в) INCF TEMP,W
   г) COMF TEMP,W

6. Какова глубина стека PIC-контроллера PIC16F870:
   а) 2
   б) 4
   в) 8
   г) 16
7. Какой вывод PIC-контроллера PIC16F870 требует подключения внешнего резистора к шине питания, чтобы получить на нем уровень логической 1:
   а) RA1
   б) RA2
   в) RA3
   г) RA4

МОДУЛЬ 2. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР

Комплексная цель изучения модуля состоит в знакомлении с аппаратным устройством PIC-контроллеров и законченных изделий на их основе, с системой команд на примере конкретного изделия.

2.1. Основы языка Ассемблер

Для написания программ PIC-контроллеров в методическом пособии применяется ассемблер MPASM фирмы Microchip, являющейся составной частью среды MPLAB. Язык ассемблер является машинно-ориентированным языком, учитывающим все тонкости и особенности микроконтроллеров и позволяет разрабатывать очень критичные ко времени или ресурсам программы, однако его применение требует большой точности, терпения и внимательного обращения с мелочами. Для облегчения программирования на языке ассемблер применяются подпрограммы, макросы, модульное программирование и другие приемы. Распечатка примера подпрограммы задержки приведен ниже:

```
;*****************************************************************
; Delay_time      = ((DELAY_value * 3) + 4) * Cycle_time
; DELAY_value     = (Delay_time - (4 * Cycle_time)) / (3 * Cycle_time)
;
; i.e. (@ 4MHz crystal)
; Delay_time      = ((32 * 3) + 4) * 1uSec
;           = 100uSec
; DELAY_value     = (500uSec - 4) / 3
;           = 165.33
;           = 165
;*****************************************************************
DELAY500
            MOVLW D'165'            ; +1        1 cycle
            MOVWF DELAY       ; +2        1 cycle
DELAY500_LOOP
            DECFSZ      DELAY,F           ; step 1    1 cycle
            GOTO        DELAY500_LOOP     ; step 2    2 cycles
DELAY500_END
            RETURN                        ; +3        2 cycles
;
X_DELAY500
            MOVWF X_DELAY           ; +1        1 cycle
X_DELAY500_LOOP
            CALL        DELAY500          ; step1     wait 500uSec
            DECFSZ      X_DELAY, F  ; step2     1 cycle
            GOTO        X_DELAY500_LOOP   ; step3     2 cycles
X_DELAY500_END
            RETURN                        ; +2        2 cycles
```

Эта подпрограмма использует две переменные – DELAY и X_DELAY. Ниже показано как эта подпрограмма вызывается если нужна задержка, например 30 миллисекунд:

```
            MOVLW       .30         ; 0,5ms*30=15ms
            CALL        X_DELAY500  ;
```

Макросы используются следующим образом

```
BANK1       macro                   ; Select Bank 1
            bsf       STATUS,RP0    ;
            bcf       STATUS,RP1    ;
            endmM
```

Этот макрос переключает память данных на банк 1.

Еще пример макроса, осуществляющего задержку:

```
Pause:      macro       DLYF
            movlw       (DLYF/5)-1
            movwf       DELAY
            call        DLY5N
            endm
```

Что бы этот макрос работал, в тексте программы должна встретится подпрограмма:

```
DLY5N
            nop
            nop
            decfsz            DELAY,F
            goto              DLY5N
            return
```

Основы языка ассемблер будем изучать на распечатках примеров программ.

Для начала уточним как работает PIC-контроллер с позиции программиста?

При включении питания PIC-контроллера первой будет исполнена команда, находящаяся в ПЗУ программ по адресу 0. После исполнения команды, содержимое счетчика команд автоматически увеличивается на единицу, и далее будет выполнена команда, расположенная в ПЗУ по адресу 1. Так будет продолжаться до тех пор, пока не исполнится команда перехода, которая принудительно запишет в программный счетчик какое-нибудь другое значение. Для отладки программ используются различные отладчики. Наиболее удачная версия бесплатно предоставляется фирмой MicroChip. Она входит в состав среды разработки MpLab.

2.2. Знакомство со средой программирования MpLab

Итак, приступим к созданию первой программы. Одновременно начнем разбирать основные возможности интегрированной среды разработки MPLAB, одной из удачных ее версий – V5.74. Напомним, что самую последнюю версию можно загрузить с сервера Microchip – www.microchip.com.

Для того, что бы написать программу, прежде всего необходимо создать проект. Проект представляет собой обычный текстовый файл с расширением .pjt, в котором хранятся все настройки, имеющие отношение к какой-то разрабатываемой программе. При желании этот файл можно редактировать или даже создавать в обычном текстовом редакторе, однако в этом нет необходимости, т.к. за Вас все сделает MPLAB. Создается проект командой «Project -> New Project». По выбору этой команды появится диалоговое окно сохранения файла, в котором необходимо задать имя проекта. Давайте зададим имя l_rab01.pjt. После нажатия на клавишу OK, будет выведено окно с названием «Edit Project», в котором задаются все настройки проекта. В строке «Target Filename» Вы увидите только что заданное имя проекта с расширением .hex .Под этим именем после ассемблирования программы будет создан файл с кодами для прошивки процессора. В строке «Include Path» указываются пути к файлам проекта, находящимся не в директории проекта. Эти пути должны быть написаны через точку с запятой. В нашем случае все файлы будут храниться в одном каталоге, поэтому этом поле можно оставить пустым. Аналогично в строке «Library Path» указывается путь к файлам библиотек, а в «Linker Script Path» путь к файлам скриптов линковщика. Эти поля тоже оставим пустыми.

Далее нажмите на кнопку «Change…», после чего появится окно выбора режимов разработки проекта «Development Mode». Наиболее простой режим – это режим редактора. В нем можно просто написать программу и проассемблировать ее. Этим режимом мы пользоваться не будем, т.к. помимо текстового написания программы, ее нужно еще и отладить. Для отладки в MPLAB имеется несколько режимов работы с эмуляторами. Нас будет интересовать «MPLAB-SIM Simulator». Это режим работы со встроенным программным эмулятором, имеющим достаточно большие возможности. После его выбора станут активными другие вкладки окна «Development Mode». Перейдите к выпадающему списку типов процессора и выберите требуемый (в данном случае PIC16F870). Чуть ниже в текстовом окне будет написано, что не вся переферийные устройства поддерживается эмулятором. Для того, что бы узнать, что не поддерживается для данного процессора более конкретно, нажмите кнопку «Details».

Затем перейдите на вкладку «Clock». Здесь можно выбрать тип генератора и тактовую частоту. Зададим ее равной частоте кварца 4 МГц. Далее перейдем к вкладке «Configuration». В ней задается режим работы сторожевого таймера.

«None» - таймер выключен. «WDT Chip Reset Enable» - по срабатыванию таймера произойдет сброс процессора. «WDT Break Enable» - По срабатыванию таймера исполнение программы будет остановлено. Выберем «None». В заключение щелкните по вкладке «Break Options». В ней находятся установки, имеющие отношения к работе с точками останова программы. Оставьте здесь все по умолчанию. Нажмите на «OK» окна «Development Mode». Возможно после этого Вы получите ряд предупреждений, однако для начала на них можно не обращать внимания. Замечу, что окно «Development Mode» доступно и через меню «Options».

В окне «Language Tool Suite» выбирается набор инструментальных средств разных компаний. Нас интересует «Microchip» (по умолчанию).

Если выделить мышью в поле «Project Files» название «test [.hex]», и нажать кнопку «Node Properties» то появится окно задания установок ассемблера. Пока в нем ничего менять не надо, позже рассмотрим назначение этих настроек.

Теперь нажмите на «OK» окна «Edit Project» и файл проекта будет создан. Сохраните его («File -> Save»). Посмотреть установки проекта можно командой «Window -> Project».

Далее создадим файл, в котором будет находится текст нашей программы. Это выполняется командой «File -> New». Сохраним его под названием «test.asm» в том же каталоге, где был сохранен файл проекта. Теперь необходимо подключить его к проекту. Для этого опять вызовем окно «Edit Project» командой «Project -> Edit Project». В нем нажмем кнопку «Add Node». При этом появится диалоговое окно загрузки файла, в котором необходимо выбрать только созданный файл программы test.asm. После выбора «test [.asm]» появится в поле «Project Files» под строкой «test [.hex]». Это будет означать, что данный файл подключен к проекту. Сохраните проект еще раз.

Это, собственно говоря, и все основные моменты создания проекта. Написано много, но пусть Вас это не пугает – на практике на создание проекта тратится не более полуминуты.

После того, как мы создали проект, можно приступать собственно к написанию программы. Пусть наша первая программа выполняет какую-нибудь простейшую задачу, например, мигает светодиодом с частотой 1Гц. Алгоритм ее работы получится следующий. По включению контроллера программа выставит единицу на линию, к которой подключен светодиод, подождет 0,5 сек, выставит нуль, опять подождет 0,5 сек, а затем повторит все заново. Напомню, что мы будем писать программу для PIC16F870, используя генератор с частотой 4 МГц. Светодиод подключим к линии RB1 (через резистор), зажигать его будем единицей.

Как уже упоминалось выше, каждый регистр специального назначения контроллера имеет свое название. В ассемблере для обращения к какому-то регистру можно использовать либо адрес этого регистра, либо его название. Например, записать значение из регистра W в регистр INTCON можно как командой movwf 0x0B так и командой movwf INTCON. Использование названия более предпочтительно, т.к. оно информативнее для программиста. Однако для того, что бы использовать название регистра сначала необходимо сопоставить само название числовому значению адреса. Сопоставление производится директивой ассемблера EQU. Например, INTCON EQU 0x0B. После этого при ассемблировании текста программы ассемблер при встрече строки “INTCON” будет вместо нее вставлять ее числовое значение “0x0B”. Таким образом, все символьные имена регистров, используемых в программе, должны быть предварительно описаны. Однако самостоятельно описывать регистры специального назначения необходимости нет, т.к. в MPLAB уже включены файлы с описаниями этих регистров для каждого PIC. Эти файлы необходимо только подключить к основной программе командой «#include». В нашем случае в начале программы нужно написать строку «#include p16f870.inc».

Кроме регистров специального назначения в программе будут использоваться и регистры общего назначения. Их надо описывать самостоятельно. Это можно делать так же с помощью директивы EQU, присвоив каждому символьному названию свой адрес. Однако это несколько громоздко. Проще воспользоваться директивами «cblock» и «endc». Эти директивы позволяют задать один начальный адрес, а ассемблер сам присвоит числовые значения заданным именам по возрастанию. В нашей программе мы будем пользоваться всего двумя регистрами – счетчиками времени при формировании задержки на 0,5 сек. Назовем их CounterLo и CounterHi. Таким образом, в программе необходимо написать следующий текст:

```
cblock 0x20
CounterLo
CounterHi
endc
```

0x20 – это адрес нулевой ячейки регистров общего назначения. После ассемблирования строке CounterLo будет присвоен адрес 0x20, а строке CounterHi адрес 0x21.

Теперь необходимо задать константы. Они задаются так же директивой EQU:

```
INIT_PORTA          EQU         b'00000000'
INIT_PORTB          EQU         b'00000000'
INIT_OPTION         EQU         b'00000000'
INIT_INTCON         EQU         b'00000000'
```

```
DELAYLO      EQU         .79
DELAYHI      EQU         .21
```

Первые четыре константы будут использоваться при инициализации соответствующих регистров. Т.к. они предназначены для инициализации отдельных битов, то их удобно задавать в двоичном формате в виде b'x… x'. Следующие две – в подпрограмме задержки на 0,5сек. Они заданы в десятичном формате, который обозначается точкой. Если вместо «.21» написать «21», то ассемблер воспримет это как шестнадцатеричный формат, который установлен по умолчанию.

Напишем подпрограмму инициализации:

```
Init
       bsf         STATUS,RP0
       movlw       INIT_PORTA
       movwf       TRISA
       movlw       INIT_PORTB
       movwf       TRISB
       movlw       INIT_OPTION
       movwf       OPTION_REG
       bcf         STATUS,RP0
       movlw       INIT_INTCON
       movwf       INTCON
       clrf        STATUS
       return
```

При входе в подпрограмму производится выбор первого банка данных ОЗУ установкой бита RP0. Далее производится выбор направления линий ввода/вывода. Т.к. у контроллера задействована всего одна линия на выход, то во все регистры TRIS можно записать нули, т.е. линии будут выходами. В регистры OPTION и INTCON также можно записать нули, т.к. таймер, сторожевой таймер и прерывания сейчас использоваться не будут. После этого производится обнуление всех битов регистра STATUS, что вызывает переключение на нулевой банк ОЗУ.

Теперь составим подпрограмму задержки на 0,5 сек:

```
Delay05
       movlw DELAYLO
       movwf CounterLo
       movlw DELAYHI
       movwf CounterHi
Del05
       decfsz      CounterLo,1
       goto        Del05
       decfsz      CounterHi,1
       goto        Del05
       return
```

и основное тело программы:

```
Begin
        call        Init
        bsf         PORTB,0
        call        Delay05
        bcf         PORTB,0
        call        Delay05
        goto        Begin

        end
```

Нажмите F10 для ассемблирования. По окончании ассемблирования Вы получите три сообщения «Register in operand not in bank 0. Ensure that bank bits are correct.». Этими сообщениями ассемблер напоминает программисту, что он производит запись в регистры, находящиеся не в нулевом банке. Эти сообщения можно отключить, если, например, вместо «movwf TRISA» написать «movwf TRISA^0x80».

Полностью программа приведена в файле l_rab01.asm.

2.3. Программа измерения напряжения

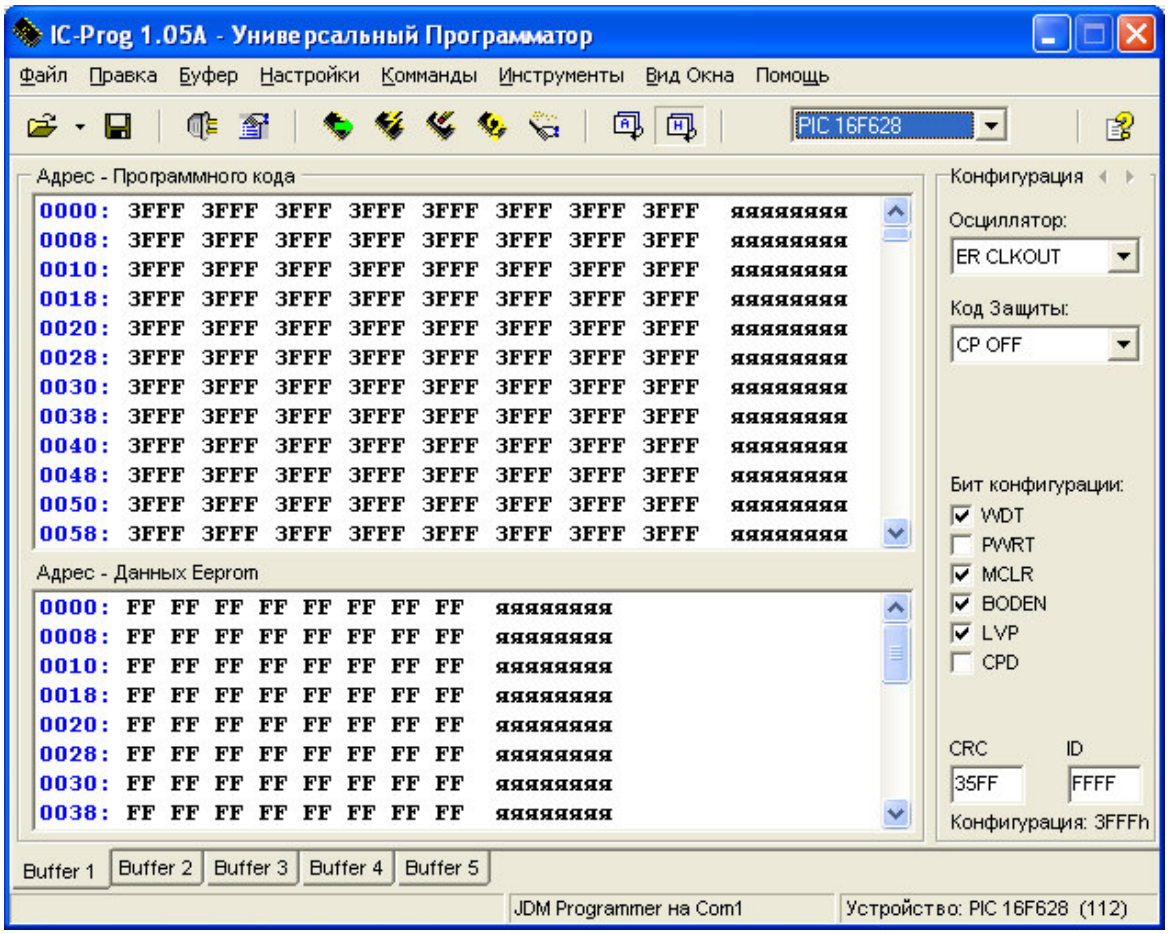
Программа измерения напряжения использует в своей работе модуль встроенного АЦП. Главный цикл программы приведен ниже:

```
    MAINLOOP
            call    IZM_AN0             ; Измерения в канале AN0
            call    B2toBCD             ; Преобразование в двоично-десятичное
значение
            call    PRINT_LCD           ; Вывод на ЖК индикатор
            call    PAUZA100            ; Пауза 100 мс
            goto    MAINLOOP            ; Возврат
```

Полностью программа программа приведена в файле l_rab02.asm.

2.4. Описание программатора и программы работы с ним

Описание работы с программатором ICPROG полностью приведено в файле icprog.chp. Здесь мы остановимся только на основных моментах. После того, как мы получили файл предназначеный для прошики PIC-контроллера, т.е. файл с расширением *.hex, нужно выйти из среды разработки в систему, и средствами системы запустить файл программатора icprog.exe. После загрузки программы мы попадем в главное окно программы icprog.exe:



Необходимо открыть файл кода предназначенного для прошивки, выбрать тип PIC-контроллера (pic16f870), и записать это файл, нажав кнопку с зеленой стрелкой. В случае записи без ошибок можно запускать программу PIC-контроллера на выполнение и проверять ее работу в реальных условиях.

2.5. Применение библиотек

В результате длительной работы разработчик накопит много исходных текстов программ, которые часто применяются в составе других разработок. Такой набор готовых программ принято называть библиотекой. Начиная работу над новым проектом, разработчик может создать папку, в которую скопирует все нужные библиотечные файлы. Если поступать так с каждым проектом, то в итоге получится несколько папок, в каждой из которых будут храниться одни и те же библиотечные файлы. А если в процессе работы над одним из проектов разработчика осенит мысль, как принципиально улучшить некий библиотечный файл? Тогда он изменит файл в одной папке, но в других папках аналогичные файлы останутся в старом виде! Очевидным решением проблемы выглядит мысль создать одну папку, в которой будут храниться все библиотечные файлы и подключать их с помощью строки

```
#include          <…\lib-pic\lib00.asm>
```

Так обычно рекомендуют работать с библитеками, модулями и т.д.

Однако это правило может породить трудноустранимые проблемы, например, в случае доработки уже сданной программы, которая с новой версией «улучшенной» библиотеки может и неработать. Поэтому в данном руководстве советуем необходимые библиотечные файлы копировать в папку проекта вместе с остальными файлами и не изменять, если проект уже сдан. Примененные в данном пособии библиотеки находятся в папки lib-pic. Вот их перечень:

eerom.asm - подпрограммы чтения и записи в EEROM;

ds1821i.asm - подпрограммы работы с микросхемой DS1821 с использованием прерываний;

b_bcd.asm - подпрограммы преобразования байта в десятичные числа (max .255) и обратно (max .99);

inc_09.asm, inc_09F.asm - подпрограммы инкремента цифры в w от 0 до 9, потом опять 0;

ds1821o.asm, ds1821.asm - подпрограммы работы с микросхемой DS1821;

b_d99_b.asm, b_bcd_b.asm - подпрограммы преобразования байта в десятичные числа и обратно;

macrof.asm - простые макросы;

lcd4p.asm, LCD.ASM - подпрограммы работы с ЖК индикатором;

I2C1307.ASM - подпрограммы работы с микросхемой DS1307 по интерфейсу I2C.

И напоследок, предлагаем самостоятельно разобраться в реальной программе, сданной ЗАКАЗЧИКУ (Приложение Б). Эта программа обслуживает автомат для упаковки в термоусадочную полиэтиленовую пленку пакета стеклобутылок и предназначена для поддержания рабочей температуры трех термоножей, отрезающих и сваривающих пакет с трех сторон (с четвертой стороны пакет уже сварен). Температура каждого термоножа должна устанавливаться и поддерживаться независимо. Программа состоит из настроечной части, измеряющей части и регулирующей части. Остальные пояснения приведены в комментариях текста программы poprt4.asm. Как это видно, реальные программы имеют значительный объем и разобраться в них совсем непросто. Поэтому актуальны способы облегчения написания и сопровождения программного обеспечения. Кроме уже упомянутых принципов модульности, применения макросов и подпрограмм в методическом пособии рекомендуется применять принцип, который можно назвать принципом «одного листа». Это когда главный цикл не разбросан по всему тексту программы, а находится в одном месте и помещается на одном листе. Это можно сделать следующим способом:

```
MainLoop
         call     Proc01
```

```
        call    Proc02
        call    Proc03
        call    Proc04
        call    Proc05
        call    Proc06
        call    Proc07
        call    Proc08
        goto    MainLoop
```

Это решение не является самым удачным, потому что часто одни подпрограммы нужно выполнять чаще других. Поэтому автор чаще применяет косвенные переходы, например так:

```
MainLoop
        call      INPUT_KEY         ; ввод кода кнопки
        call      ANALIZ_KEY        ; анализ кода кнопки
        clrf      PCLATH            ; находится в первых 256 байтах
        movf      CountProc,w       ; косвенный переход
        addwf     PCL,f             ;
        goto      VOLTMETR          ; 0  вольтметр
        goto      WORK              ; 1  рабочий режим
        goto      UST_T1            ; 2  уставка температуры 1-го канала
        goto      UST_T2            ; 3  уставка температуры 2-го канала
        goto      UST_T3            ; 4  уставка температуры 3-го канала
        ; и так далее
VOLTMETR
        call      DO_VOLTMETR       ; делаем измерения
        goto      MainLoop          ; возврат в главный цикл
WORK
        call      DO_WORK           ; делаем работу
        goto      MainLoop          ; возврат в главный цикл
UST_T1
        call      UST_T1            ; делаем уставку T1
        goto      MainLoop          ; возврат в главный цикл
UST_T2
        call      UST_T2            ; делаем уставку T2
        goto      MainLoop          ; возврат в главный цикл
UST_T3
        call      UST_T3            ; делаем уставку T3
        goto      MainLoop          ; возврат в главный цикл
```

Здесь подпрограммы INPUT_KEY, ANALIZ_KEY выполняются каждый проход главного цикла, а подпрограммы VOLTMETR, WORK, UST_T1 и другие догда когда это нужно. Управляет выбором этих подпрограмм переменная счетчик процессов COUNT_PROC с помощью косвенной адресации, т.е. содержимое этой переменной суммируется с текущим значением программного счетчика и происходит переход согласно новому значению программного счетчика. Здесь важно помнить что косвенная адресация правильно работает только в пределах блоков программной памяти 256 байт.

ПРОЕКТНОЕ ЗАДАНИЕ К МОДУЛЮ 2

1. Разработайте подпрограмму сложения 16-ти разрядного числа с 32-х разрядным.
2. Разработайте подпрограмму инкремента значения переменной TEMP до величины 80h.

ТЕСТОВОЕ ЗАДАНИЕ К МОДУЛЮ 2

Тестовое задание выполняется в течении 5 минут и содержит 7 вопросов (правильный ответ помечается любым значком).
Результаты теста оцениваются по следующей шкале:
7 или 6 правильных ответов – оценка «5»,
5 правильных ответов – оценка «4»,
4 правильных ответа – оценка «3»,
менее 4 правильных ответов – оценка «2».

1. Какая из нижеперечисленных команд переключит доступ к банку 1:
   ```
   а)  bcf         STATUS,RP0
   б)  bsf         STATUS,RP0
   в)  bcf         STATUS,RP1
   г)  bsf         STATUS,RP1
   ```
2. Что будет в рабочем регистре после выполнения команды **`clrf  W`**:
   а) 0
   б) 0FFh
   в) 1
   г) 2
3. В ячейке DELAY хранится число три. Что будет в ячейке DELAY после выполнения команды **`swapf  DELAY,W`**:
   а) 0
   б) 30h
   в) 3
   г) FFh
4. В ячейке DELAY хранится число три. Что будет в ячейке DELAY после выполнения команды **`swapf  DELAY,F`**:
   а) 0
   б) 30h
   в) 3
   г) FFh
5. Какая из нижеследующих команд произведет программный сброс микроконтроллера PIC16F870:
   ```
   а)  goto        0
   ```

б) `goto 0FFFh`

в) `sleep`

г) `goto $`

6. Что будет в регистре STATUS после выполнения команды **`clrf STATUS:`**

а) 0

б) 0FFh

в) 30h

г) 38h

7. В регистре W число шесть. Чему будет равно значение флага C после команды `sublw 5`:

а) 0

б) 1

в) не изменится

г) 0 или 1

МОДУЛЬ 3. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

Комплексная цель изучения модуля состоит в ознакомлении применения языков высокого уровня для программирования PIC-контроллеров, оптимизации программ на конкретных примерах.

3.1. Языки высокого уровня и PIC-контроллеры

Технические возможности PIC-контроллеров увеличиваются и вместе с ней возрастает сложность создания программы. Даже просто на распечатку большой программы тратится много материала (времени), а распечатывать приходится не один раз. Например, максимальный объем программы PIC16F876 8 килобайт, это 8000 строк команд контроллера, с учетом комментариев это примерно 10 тыс. строк. Если 10000 разделить на 70 получается 140 страниц текста - целая книга. Многократно увеличивается и трудноемкость разработки сложной программы. Поэтому существует потребность в средствах повышающих производительность труда программиста - языках высокого уровня [10]. В каком случае имеет смысл выбрать язык высокого уровня смотри таблицу 6.

Таблица 6. Выбор средства программирования.

	Ваша программа простая	Ваша программа сложная	Примечание
Вы программируете редко	Ассемблер	Ассемблер	
Вы программируете часто	Ассемблер	Язык высокого уровня	

Эта таблица представляет простую мысль, что на изучение языка высокого уровня необходимо время, а язык АССЕМБЛЕР необходимо знать каждому. После того как решили программировать на языке высокого уровня встает вопрос, а на каком именно языке программировать. Это вопрос не такой простой. Выработаем требования которым должен соответствовать язык высокого уровня который мы собираемся изучать. Во первых он должен быть бесплатным, во вторых он должен поддерживать наш тип PIC-контроллера, в третьих желательно чтобы он порождал ассемблерный файл, а не «hex» код, чтобы мы могли потом «прогнать» его на MPLAB. Среда разработки MPLAB настойчиво рекомендует применять компилятор языка Си версии PICC фирмы HI-TECH Sofware, предназначенного специально для PIC-контроллеров. К сожалению бесплатная версия этого языка PICC Lite не поддерживает примененный нами PIC-контроллер PIC16F870. Также этим компилятором сразу генерируется «hex» код, что затрудняет изучение кода программы начинающими пользователями. Поэтому от этой версии компилятора

пришлось отказаться. Существуют [12] и другие версии Си компиляторов, среди которых SDCC бесплатный, поддерживает PIC16F870 и при компиляции генерируется ассемблерный листинг. К сожалению своей среды разработки этот компилятор не имеет, а имеющаяся документация (на английском языке) требует длительного изучения чтобы создать приемлемый алгоритм разработки. Предлагается желающим попробовать освоить этот компилятор самостоятельно. Есть другие языки высокого уровня [13]. После поисков было решено остановиться на языке F2P (Forth to PIC compiler). Этот язык бесплатный, генерирует ассемблерный текст, и, после небольшого «улучшения» стал поддерживать PIC16F870.

3.2. Язык программирования F2p.

Язык высокого уровня, с которым мы познакомимся в методическом пособие, это форт-подобный язык [6] автора Michael Josefsson, разработанный еще в 1997 году, который он назвал F2P (Forth to PIC compiler). Выбор именно этого языка из других форт-подобных обусловлен тем, что при компиляции он генерирует ассемблерный код, который можно анализировать с помощью той же MPLAB. Небольшая «модернизация» кода компилятора позволила применить его и для PIC16F870.

Знакомство с языком F2P начнем с решения той же задачи (мигание светодиодом с частотой 1 Гц), что и в L_RAB01.ASM. Ниже приведен текст программы (fmp01.4th). Напомним, что в форт-подобных языках принято в комментариях (в скобках) приводить состояние стека до исполнения слова и после:

```
\ мигает светодиод с частотой 1 Гц
: MAIN  ( -- )
     BEGIN ON_LED 500MSEC OFF_LED AGAIN ;
\ зажигаем светодиод
: ON_LED ( -- )
$       bsf       portb,1             ; ON_LED
;
\ задержка 500 миллисекунд
:  500MSEC ( -- )
      5 0 DO 200 xDELAY500 LOOP ;
\ тушим светодиод
: OFF_LED ( -- )
$       bcf       portb,1             ; OFF_LED
;
```

Программа на языке F2P представляет собой предложение из слов, разделенных пробелами. Главное слово программы – MAIN, которое всегда выполняется. Слова начинаются с двоеточия (:) и заканчивается точкой с запятой

(;). В нашем случае слово MAIN

: MAIN (--) BEGIN ON_LED 500MSEC OFF_LED AGAIN ;

состоит из слов:

BEGIN – начало цикла,

ON_LED – подпрограмма, включающая светодиод,

500MSEC – подпрограмма задержки на 0,5 сек,

OFF_LED - подпрограмма, выключающая светодиод,

AGAIN – конец цикла.

Таким образом светодиод должен мигать с частотой 1 Гц.

Рассмотрим остальные слова: слова BEGIN и AGAIN – системные и компилируются самим компилятором, слово ON_LED создано нами:

```
: ON_LED ( -- )
$       bsf       portb,1             ; ON_LED
;
```

по общим правилам – в начале идет двоеточие (:), признак того что создается новое слово, потом имя этого слово, далее комментарий, в котором принято отображать что делает это слово со стеком форт-системы, затем идет ассемблерная вставка, начинающаяся со знака $ и закачивающаяся концом строки, затем слово «точка с запятой» завершает создание слова ON_LED, которое зажигает светодиод.

Следующее слово - 500MSEC – подпрограмма задержки на 0,5 сек

```
:  500MSEC ( -- )
      5 0 DO 200 xDELAY500 LOOP ;
```

представляет собой пять раз выполненное слова `200 XDELAY500`. Слово 200 кладет на стек форт-системы число 200, а слово xDELAY500 снимает это число со стека как параметр и пропорционально ему производит задержку 200*500=100000 микросекунд, т.е. 0,1 сек. В итоге получается задержка 0,5 сек. Слова DO (начало цикла), xDELAY500, LOOP (конец цикла) берутся из архива, т.е. файла f3park74.ark. И последнее слово OFF_LED аналогично ON_LED, только гасит светодиод. Откомпилируем полученную программу, запустив файл fmp01.bat:

```
\myf2p\f2pa fmp01.4th fmp01 -c74 > fmp01.txt
\myf2p\f2pfiltr fmp01 fmp01.asm
\MPLAB574\mpasm /c- fmp01.asm
del fmp01.asm
ren fmp01 fmp01.asm
```

Поясним работу этого файла. В первой строке запускается компилятор f2pa.exe, который в качестве параметра для компиляции получает имя нашего файла fmp01.4th с ключом -c74 (PIC-контроллер pic16c74, с этим ключом компилятор правильно компилирует и для pic16f870), выходной файл компилятор на языке ассемблер fmp01, сообщения о ходе компиляции переслать в файл fmp01.txt На этом работу можно было закончить, но существует проблема. Язык ФОРТ

допускает применение любых символов в именах слов(подпрограмм), что не скажешь о языке ассемблер. Такие важные слова как “!” (сохранить в памяти), “@” (извлечь из памяти) и другие в ассемблере запрещены. Поэтому второй строкой в командном файле запускается программа-фильтр, которая заменяет запрещенные символы на разрешенные и формирует выходной файл для MPASM (fmp01.asm). В третьей строке запускается ассемблер MPASM с входным файлом fmp01.asm. Выходные файлы с расширениями .hex, .cod получаются после работы MPASM по умолчанию. Теперь можно стереть временный файл с не «фортовскими» символами и переименовать с «фортовскими» в файл для среды MPLAB. Теперь запускает MPLAB и создаем новый проект fmp01.prj в папке fmp01, указав на файл fmp01.hex. Сохранив проект, можно начинать работу с ним в среде MPLAB стандартными средствами («симулировать» и т.д.).

Рассмотрим подробнее результат работы компилятора F2P, т.е. файл fmp01.asm

```
; Forth to PIC compiler -- Header file
 TITLE F2P
 LIST    P=16f870, F=INHX8M
 #INCLUDE "P16f870.INC"
      __CONFIG _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_ENABLE_OFF & _LVP_OFF & _CPD_OFF

;*****************************************************************
;
;       Заголовок для FMP00.4th
;
;       Version: 25.09.2008
;
;       Creator: Alpatov
;
;*****************************************************************
;
; Initialize the Forth system on a PIC16F870
;
; Forth system
MAXRAM      EQU     0x7f
TMP1        EQU     0x7e
TMP2        EQU     0x7d
TMP3        EQU     0x7c
TMP4        EQU     0x7b        ; Alpatov
TMP5        EQU     0x7a        ; Alpatov
TMP6        EQU     0x79        ; Alpatov
TMP7        EQU     0x78        ; Alpatov
TMP8        EQU     0x77        ; Alpatov
TMP9        EQU     0x76        ; Alpatov
TMP10       EQU     0x75        ; Alpatov
TMP11       EQU     0x74        ; Alpatov
TMP12       EQU     0x73        ; Alpatov
```

```
    _FLAG        EQU     0x72        ; Alpatov
    STATUS_ISR  EQU     0x71        ; Alpatov
    W_ISR       EQU     0x70        ; Alpatov
    ;
    RZERO       EQU     0xC0        ; Alpatov, 0xff нельзя, см.
pic16f870.pdf
    SZERO       EQU     W_ISR       ; 1 'Below' of Parameter Stack. See
Manual.
    RSP         EQU     MAXRAM      ; Last byte of BANK0
    ;
    #define _stroka0   _FLAG,0        ; напечатана СТРОКА0-?
    #define _stroka1   _FLAG,1        ; напечатана СТРОКА1-?
    ;*****************************************************************
    ; Start of code area
    ;
           ORG     0x000
    ;       movlw   0
           movlw   0x07             ; TABLEs are created in last page
F2PHEADR.ASM
           movwf   PCLATH           ;
           call    InitSYS          ; Инициализация системы
           goto    __MAIN            ; Goto Main Program
    ;================================================================
           ORG     0x004
           movwf   W_ISR            ;
           swapf   STATUS,W         ;
           movwf   STATUS_ISR       ;
           call    _IntSERVIS       ; Interrupt_servis
           swapf   STATUS_ISR,W     ;
           movwf   STATUS           ;
           swapf   W_ISR,F          ;
           swapf   W_ISR,W          ;
           bcf     INTCON,T0IF      ;
           retfie
    ;================================================================
    ;
    InitSYS
           call      InitPORT       ; Инициализация портов
           call      CLRF_RAM       ; Обнуление памяти
           return
    ;================================================================
    InitPORT
           clrf      PORTA          ;
           clrf      PORTB          ;
           clrf      PORTC          ;
    ;
           bsf       STATUS,RP0     ; Select Bank 1
           bcf       STATUS,RP1     ;
           movlw     B'01000111'    ; Int, R-on, 1:256 TMR0
           movwf     OPTION_REG
    ; иниц. рег.упр PORTA
           movlw     B'00011111'    ; 0-5 - INPUT
```

```
       movwf     TRISA           ;
; иниц. рег.упр PORTB
       movlw     B'11000000'     ; 6,7- INPUT, остальные - OUT
       movwf     TRISB           ;
; иниц. рег.упр PORTC
       movlw     B'01011000'     ; 3,4,6 - INPUT, остальные - OUT
       movwf     TRISC           ;
; иниц. АЦП
       movlw     B'10000110'     ; all DIGIT
       movwf     ADCON1          ; Configure A/D inputs
       bcf       PIE1,ADIE       ; Disable A/D interrupts
       bcf       STATUS,RP0      ; Select Bank 0
       movlw     B'10000001'     ; F:  Clock, A/D is on, Ch0 is selected
       movwf     ADCON0          ;
       bcf       PIR1,ADIF       ; Clear A/D interrupt flag bit
;
       movlw     B'00100000'     ; TMR0
       movwf     INTCON
       return
;-----------------------------------------------------------------
CLRF_RAM
       clrf      _FLAG
       movlw     0x020           ;
       movwf     FSR             ;
LP_CLRF_RAM
       clrf      INDF            ;
       incf      FSR,f           ;
       btfss     FSR,6           ; 3FH
       goto      LP_CLRF_RAM     ;
       return

; Forth to PIC compiler -- Startup Code

__MAIN
 movlw   SZERO
 movwf   FSR
 movlw   RZERO
 movwf   RSP
 goto    _MAIN

; Forth to PIC compiler -- Equates
_COUNT  EQU   D'32'

; Forth to PIC compiler -- Code Extracted From Archive
_@                      ; @
          MOVWF     TMP1
          MOVF      FSR,W
          MOVWF     TMP2
          MOVF      TMP1,W
          MOVWF     FSR
          MOVF      INDF,W
          MOVWF     TMP1
```

```
          MOVF      TMP2,W
          MOVWF     FSR
          MOVF      TMP1,W
          RETURN
_+                        ; +
          ADDWF     INDF,W
          INCF      FSR,F
          RETURN
_!                        ; !
          MOVWF     TMP1
          MOVF      INDF,W
          MOVWF     TMP2
          MOVF      FSR,W
          MOVWF     TMP3
          MOVF      TMP1,W
          MOVWF     FSR
          MOVF      TMP2,W
          MOVWF     INDF
          INCF      TMP3,W
          MOVWF     FSR
          MOVF      INDF,W
          INCF      FSR,F
          RETURN
_DO                       ; DO
          MOVWF     TMP1
          MOVF      INDF,W
          MOVWF     TMP2
          MOVF      FSR,W
          MOVWF     TMP3
          DECF      RSP,W
          MOVWF     FSR
          MOVF      TMP2,W
          MOVWF     INDF
          DECF      FSR,F
          MOVF      TMP1,W
          MOVWF     INDF
          MOVF      FSR,W
          MOVWF     RSP
          INCF      TMP3,W
          MOVWF     FSR
          MOVF      INDF,W
          INCF      FSR,F
          RETURN
_XDELAY500                    ; XDELAY500

        MOVWF   TMP1
LP_XDELAY500
        CALL    DELAY500
        DECFSZ  TMP1,F
        GOTO    LP_XDELAY500
        MOVF    INDF,W
        INCF    FSR,F
```

```
        RETURN
DELAY500
        MOVLW   .165
        MOVWF   TMP2
LP_DELAY500
        DECFSZ  TMP2,F
        GOTO    LP_DELAY500
        RETURN
_LOOP                   ; LOOP
          MOVWF     TMP1
          MOVF      FSR,W
          MOVWF     TMP3
          MOVF      RSP,W
          MOVWF     FSR
          INCF      INDF,F
          MOVF      INDF,W
          INCF      FSR,F
          XORWF     INDF,W
          BTFSS     STATUS,Z
          GOTO      LOOPMORE
          INCF      FSR,W
          MOVWF     RSP
          MOVF      TMP3,W
          MOVWF     FSR
          MOVF      TMP1,W
          BSF       STATUS,C
          RETURN
LOOPMORE
          DECF      FSR,W
          MOVWF     RSP
          MOVF      TMP3,W
          MOVWF     FSR
          MOVF      TMP1,W
          BCF       STATUS,C
          RETURN
_TEST                   ; TEST
         RETURN

; Forth to PIC compiler -- Compiled Forth Source
; fmp01.4TH - программа для методического пособи
; мигает светодиод PORTB,1
; Файл программы для f2pa.exe
; Схема controlZ.sch, 4 MHz, PIC16F870
;       data   22.10.2008 - 04h -
; ***************************************************************
; ***************************************************************
; мигает светодиод с частотой 1 Гц
_MAIN                   ; : MAIN
 decf        FSR,f      ; COUNT
 movwf       INDF
 movlw       _COUNT
 call        _@         ; @
```

```
 decf        FSR,f       ; 1
 movwf       INDF
 movlw       D'1'
 call        _+          ; +
 decf        FSR,f       ; COUNT
 movwf       INDF
 movlw       _COUNT
 call        _!          ; !
L1                       ; BEGIN
 Call        _ON_LED          ; ON_LED
 Call        _500MSEC         ; 500MSEC
 Call        _OFF_LED         ; OFF_LED
 Goto        L1
 return           ; ;
; зажигаем светодиод
_ON_LED
      bsf       portb,1             ; ON_LED
 return           ; ;
; задержка 500 миллисекунд
_500MSEC
 decf        FSR,f       ; 5
 movwf       INDF
 movlw       D'5'
 decf        FSR,f       ; 0
 movwf       INDF
 movlw       D'0'
 call        _DO         ; DO
L2
 decf        FSR,f       ; 200
 movwf       INDF
 movlw       D'200'
 call        _XDELAY500        ; XDELAY500
 call        _LOOP       ; LOOP
 btfss       STATUS,C
 goto        L2
 return           ; ;
; тушим светодиод
_OFF_LED
      bcf       portb,1             ; OFF_LED
 return           ; ;
; ------------------------------------------------------------------

_SETUP                   ; : SETUP
 call  _TEST       ; TEST
 return           ; ;
; ==================================================================
; ПП обработки прерываний

_INTSERVIS                    ; : INTSERVIS
 return           ; ;
; ==================================================================
      ORG     0x700
```

```
; 0123456789ABCDEF
; ================================================================
      ORG     0x2100
      DE      "FMP01.4TH"
; End of program
```

Как видим, компилятор сначало копирует в выходной файл файл заголовка f2phdr.asm, в котором определяются все нужные переменные, подпрограммы инициализации, обработки прерываний, задается точка начала работы и перехода на точку создания форт_системы (метка __MAIN). Код создания форт_системы генерируется самим компилятором. Вот его распечатка:

```
__MAIN
  movlw   SZERO
  movwf   FSR
  movlw   RZERO
  movwf   RSP
  goto    _MAIN
```

Далее компилятор извлекает из архивного файла коды тех слов, которые упомянуты в тексте программы. И наконец приступает к создании кода в соответствии с текстом программы, т.е. файла fmp01.4th. Как видите, получилась понятная, простая и надежная система компиляции, каждый шаг которой прозрачен.

3.3. Программа измерения напряжения

Программа измерения напряжения с помощью встроенного АЦП находится в файле fmp02.4th. Она измеряет напряжение канала AN0 и выводит результат на ЖК индикатор:

```
\ ******************************************************************
2VARIABLE SUM_AN0   \ 20h-23h
\ ================================================================
\ измеряем и выводим
: MAIN  ( -- )
    SETUP BEGIN 0. SUM_AN0 2!
    16 0 DO IZM_AN0 SUM_AN0 2@ UD+ SUM_AN0 2! 20 XDELAY500 LOOP
    GOTOLINE0 'U' LCDPUTCHAR '=' LCDPUTCHAR SPACE
    SUM_AN0 2@ 4./ .UD SPACE 'm' LCDPUTCHAR 'V' LCDPUTCHAR AGAIN
;
\ ----------------------------------------------------------------
: SETUP  ( -- ) InitLCD TEST ;
: .UD   ( uL uH -- )   UDtoBCD
     30'H + LCDPUTCHAR 30'H + LCDPUTCHAR 30'H + LCDPUTCHAR
     30'H + LCDPUTCHAR 30'H + LCDPUTCHAR ;
: SPACE ( -- ) 20'H LCDPUTCHAR ;
: IZM_AN0 ( -- uD ) IZM_ADC ;
```

Алгоритм программы: после инициализации 16-ть раз производятся измерения с помощью предварительно настроенного на канал AN0 встроенного АЦП (слово

IZM_AN0) и результат суммируется в 16-ти разрядной переменной SUM_AN0. После этого на индикатор выводится строка:

U=XXXXX mV

- где XXXXX есть значение из ячейки SUM_AN0, деленное на четыре. Встроенный АЦП имеет десять разрядов, т.е. максимальное число измерения будет 1023. Программа измеряет 16 раз, поэтому в ячейки SUM_AN0 будет максимум 16*1023=16368. После деления на четыре максимальное число - 4092. Следовательно, если мы установим опорное напряжение 4,092 В, то значение из ячейки SUM_AN0 можно без дополнительных преобразований выводить на индикатор. Это будет входное напряжение в милливольтах. Слово, которое непосредственно производит измерения - IZM_ADC - находится в архиве. Вот его распечатка:

```
; ------------------------------------------------------------------
#archive IZM_ADC
;  ( -- ud )
        bsf     ADCON0,GO       ; Start A/D Conversion
        decf    FSR,F           ; TOS --> NOS
        MOVWF   INDF            ;
        decf    FSR,F           ; for REZ_ADCL
LOOP_IZM_ADC
        btfss   PIR1,ADIF       ; OK-?
        goto    LOOP_IZM_ADC    ; NO
        bsf     STATUS,RP0      ; BANK 1
        movf    ADRESL,W        ;
        bcf     STATUS,RP0      ; BANK 0
        movwf   INDF            ; REZ_ADCL
        movf    ADRESH,W        ; REZ_ADCH --> TOS
        bcf     PIR1,ADIF       ;
        return
```

В нем в начале запускается модуль АЦП, потом смещается указатель стека с сохранением значения на вершине и далее подпрограмма ждет окончания АЦП преобразования. После окончания преобразований результат сохраняется на вершине стека и сбрасывается флаг прерывания преобразования АЦП. Напомним, что настраивается модуль АЦП на работу с каналом AN0 время инициализации, которая осуществляется в заголовочном файле f2phdr74.asm. Вот распечатка фрагмента, осуществляющего инициализацию:

```
; иниц. АЦП
       bsf       STATUS,RP0      ; Select Bank 1
       movlw     B'10000101'     ; AN3-Vref; AN1,AN0-A
       movwf     ADCON1          ; Configure A/D inputs
       bcf       PIE1,ADIE       ; Disable A/D interrupts
       bcf       STATUS,RP0      ; Select Bank 0
       movlw     B'10000001'     ; F:  Clock, A/D is on, Ch0 is selected
       movwf     ADCON0          ;
```

```
        bcf       PIR1,ADIF       ; Clear A/D interrupt flag bit
```

Если потребуется повторная перенастройка модуля АЦП на другой канал, ее можно осуществить с помощью слова InitADC:

```
; ====================================================================
#archive InitADC
;  ( INTCON ADCON1 -- )
        bsf     STATUS,RP0      ; BANK 1
        bcf     STATUS,RP1      ;
        MOVWF   ADCON1          ; ADC
        MOVF    INDF,W
        INCF    FSR,F
        bcf     STATUS,RP0      ; BANK 0
        MOVWF   INTCON          ; INTCON
        MOVF    INDF,W          ; 2-й элемент - на вершину СТЕКА
        INCF    FSR,F           ; сместим УКАЗАТЕЛЬ
        return
```

3.4. Программа измерения частоты

Программа измерения частоты сигнала поступающего на вход встроенного 16-ти разрядного таймера TMR1 находится в файле fmp03.4th. Таймер TMR1 работает в режиме счетчика внешних импульсов Программа определяет число импульсов за время в одну секунду (т.е. частоту) и выводит результат на ЖК индикатор:

```
\ ********************************************************************
VARIABLE GCount   \ 20h-23h
\ ====================================================================
\ ждем
: MAIN  ( -- )
    SETUP GOTOLINE0 20 GCount ! \ OprosTMR1 2drop
$     bsf       INTCON,GIE      ; разрешим прерывания
    BEGIN AGAIN ;
\ ====================================================================
: SETUP  ( -- ) InitLCD TEST ;
: .UD   ( uL uH -- )   UDtoBCD
     30'H + LCDPUTCHAR 30'H + LCDPUTCHAR 30'H + LCDPUTCHAR
     30'H + LCDPUTCHAR 30'H + LCDPUTCHAR ;
: SPACE ( -- ) 20'H LCDPUTCHAR ;
\ ====================================================================
\ ПП обработки прерываний
: IntSERVIS  ( -- )
     61 InitTMR0
$     decfsz    _GCount,F
$     return
     OprosTMR1 GOTOLINE0 'F' LCDPUTCHAR '=' LCDPUTCHAR SPACE
     .UD SPACE 'H' LCDPUTCHAR 'z' LCDPUTCHAR GOTOLINE0
     20 GCount ! ;
```

В этой программе основной цикл осуществляется при обработки прерывания от таймера TMR0, настроенного на срабатывания каждые 50 миллисекунд. Поэтому переменной GCount присвоено значение 20 (20x50=1000мсек). При каждом прерывании значение GCount декрементируется и когда становится равным нулю с помощью слова OprosTMR1 происходит останов таймера TMR1, считывания его значения на вершину стека и повторный запуск. Вот распечатка слова OprosTMR1:

```
; ================================================================
#archive OprosTMR1 ( -- FC0L FC0H )
        bcf     T1CON,TMR1ON        ; ostanov TMR1
        decf    FSR,F               ;
        movwf   INDF                ; TOS --> NOS
        movf    TMR1L,W             ;
        decf    FSR,F               ;
        movwf   INDF                ; FC0L
        movf    TMR1H,W             ; FC0H
        clrf    TMR1L
        clrf    TMR1H
        bsf     T1CON,TMR1ON        ; zapusk TMR1
        return
```

Напомним, что настраивается таймер TMR1 на этот режим работы вовремя инициализации, которая осуществляется в заголовочном файле f2phdr74.asm. Вот распечатка фрагмента, осуществляющего инициализацию:

```
          movlw     B'00000111'     ; TMR1 -async
          movwf     T1CON           ;
```

3.5. Программа измерения температуры

Измерение температуры осуществляется с помощью микросхемы DS1821, специально для этого сделанной [14]. Диапазон измерения - от минус 50 до +125 oC. Интерфейс связи с этой микросхемой однопроводный, библиотеки обслуживание находятся в файле DS1821.ARK и подключены в файл архива f2park74.ark. Программа измерения температуры находится в файле fmp04.4th и приведена ниже. Она работает следующим образом: после инициализации словом **ZapuskDS1821** запускается преобразование температуры в код, который хранится в самой микросхеме DS1821, потом, после задержки в 0,1 сек, производится считывание этого кода и сохранение в переменной **TempTC** и микросхема DS1821 останавливается. Начинается вывод значения величины температуры на индикатор. Курсор словом **GOTOLINE0** устанавливается в левый верхний угол индикатора, потом выводится **T=** и словом **.N** значение переменной **TempTC** со знаком и завершается символами **oC**. Цикл завершен.

```
\ ***************************************************************
```

```
VARIABLE TempTC   \ 20h-23h
\ ================================================================
\ измеряем и выводим ТЕМПЕРАТУРУ
: MAIN  ( -- )
    SETUP BEGIN ZapuskDS1821 200 xDELAY500
    ReadDS1821 TempTC ! OstanovDS1821
    GOTOLINE0 'T' LCDPUTCHAR '=' LCDPUTCHAR SPACE
    TempTC @ .N SPACE 'o' LCDPUTCHAR 'C' LCDPUTCHAR AGAIN
;
\ ----------------------------------------------------------------
: SETUP  ( -- ) InitLCD InitDS1821 TEST ;
: .N ( n -- )
     NtoBCD IF '-' LCDputCHAR THEN 30'H + LCDputCHAR
     30'H + LCDputCHAR 30'H + LCDputCHAR ;
: SPACE ( -- ) 20'H LCDPUTCHAR ;
\ ================================================================
```

Особенностью библиотеки DS1821.ARK состоит в том, что словом **`InitDS1821`** подключается сразу много слов (так тоже можно делать). Код одного из них, **`ReadDS1821`** приведен ниже. Видно, чтобы прочитать данные из микросхемы DS1821 нужно послать в нее код чтения, равный 0xAA.

```
; ReadDS1821 ( -- n )
_ReadDS1821
        decf     FSR,F             ;
        movwf    INDF              ; TOS
        MOVLW   0xAA               ;
        CALL    SendDS1821         ;
        CALL    RD_DS1821          ;
        RETURN
```

Остальные слова работают аналогично.

3.6. Программа фиксации текущего времени

Очень часто приходится не только измерять какую-либо величину, но и фиксировать момент времени измерения. Для этой цели в стенде применена микросхема часов DS1307. Она имеет интерфейс I2C, резервную батарейку питания и кварц на 32758 Гц. Программа фиксации текущего времени находится в файле fmp05.4th. Предлагается разобраться в ее работе самостоятельно.

3.7. Оптимизация кода программы

Для любого языка высого уровня существует проблема оптимизации кода, в основнос целью уменьшения объема. Не избежал этого и язык F2p. Оптимизацией кода программы можно заниматься во время написания текста, так и после того как программа написана. Возможность оптимизации существует тогда когда эффективность кода на ассемблере выше чем аналогичного кода генерируемого

компилятором F2P. Поясним сказанное примером. Пусть в тексте программы есть фрагмент увеличивающий значение переменной COUNT на единицу:

```
COUNT @ ! + COUNT !
```

Эта строка скомпилируется компилятором в следующий код:

```
decf        FSR,f           ; COUNT
movwf       INDF
movlw       _COUNT
call        _@              ; @
decf        FSR,f           ; 1
movwf       INDF
movlw       D'1'
call        _+              ; +
decf        FSR,f           ; COUNT
movwf       INDF
movlw       _COUNT
call        _!              ; !
```

Совершенно безболезненно можно все это заменит на одну строку - вставку на языке ассемблер, выполняющее то же действие:

```
$       incf        COUNT,F          ; увеличим COUNT на 1
```

Таким образом код стал на одиннадцать строк короче, не считая подпрограмм. Существуют и другие возможности оптимизации.

Но не все слова нуждаются в оптимизации. Большинство выполняется весьма эффективно. Например, подпрограмму 16-ти разрядного деления на 4:

```
    ; разделим 16-ти разрядное значение в переменных TEMPH, TEMPL на четыре
DIV4
     bcf        STATUS,C
     rrf        TEMPH,F
     rrf        TEMPL,F     ; делим на 2
     bcf        STATUS,C
     rrf        TEMPH,F
     rrf        TEMPL,F     ; делим на 4
     return
```

сравним со словом 4./ из архива F2P:

```
#archive 4./
;  ( uD1 -- uD1/4 )
        movwf   TMP1            ; запомним uD1H
        bcf     STATUS,C        ;
        rrf     TMP1,F          ; делим на 2
        rrf     INDF,F          ;
        bcf     STATUS,C        ;
        rrf     TMP1,W          ; делим на 4
        rrf     INDF,F          ;
        return
```

Получается что эти подпрограммы выполняются практически одинаково быстро. Таким образом правильно оптимизированная программа на языке f2p будет

выполняться практически одинаково быстро с программой написанной на ассемблере.

3.8. Библиотеки языка F2pa

В процессе разработок программ возникает необходимость в новых словах(подпрограммах), которые желательно иметь в архивном файле f2park74.ark. На сегодняшний день имеются библиотеки в следующих файлах:

2stack.ARK - работа на стеке с 16-ти разрядными значениями;

EEROM870.ARK - чтение и запись в EEROM;

math16.ARK - 16-ти разрядная математика;

preob_to_BCD.ARK - преобразование в двоично-десятичное и обратно;

DS1821.ARK - работа с микросхемой DS1821;

DS1307.ARK - работа с микросхемой DS1307;

oprosSQ.ARK - опрос датчиков;

Init870.ARK - инициализация pic16f870;

inc_decrement.ARK - инкремент-декремент;

out_HC595.ark - вывод байта в микросхему 74HC595.

По мере надобности слова из этих библиотек нужно вставить в архивный файл f2park74.ark путем простого копирования (команда `#include <file>` в языке f2p отсутствует) средствами системного программного обеспечения.

Новое слово для архива сравнительно легко разработать. Покажем это на двух примерах. Первый пример – разработаем слово 2DROP, которое снимает со стека 16-ти разрядное значение. Напомним, что слово DROP, которое есть в архиве f2park74.ark, снимает со стека байт. Код слова DROP:

```
#archive drop
; ( u1 --- )
          movf      INDF,W
          incf      FSR,F
          return
```

Это слово работает следующим образом: первой командой помещает на вершину стека второй элемент, второй командой смещает указатель на третий элемент и все. Аналогично сделаем слово 2DROP:

```
#archive 2drop
; ( uL uH -- )
        incf    FSR,F
        movf    INDF,W
        incf    FSR,F
        return
```

Это слово работает следующим образом: первой командой смещает указатель на третий элемент, второй командой помещает его на вершину стека и третьей командой смещает указатель на четвертый третий элемент.

Второй пример длинее и сложнее. В нем применены ячейки памяти, играющие роль локальных переменных. Имеется слово `UtoBCD ( u – L M H )`. Это слово преобразует байт беззнака в цифры, которые потом можно вывести на индикатор. Нам требуется слово, которое преобразовало бы байт со знаком для вывода на индикатор. Назовем это слово `NtoBCD ( n – L M H Z )`. В дополнение к существующим цифрам это слово помещает на вершину стека код знака Z, который равен 0 когда число положительное и FF когда число отрицательное. Вывести байт со знаком с помощью этого слова можно например так:

```
: .N ( n -- )
     NtoBCD IF '-' LCDputCHAR THEN 30'H + LCDputCHAR
     30'H + LCDputCHAR 30'H + LCDputCHAR ;
```

Теперь по аналогии со словом `UtoBCD ( u – L M H )` разработаем слово `NtoBCD ( n – L M H Z )`.

```
#archive NtoBCD
; ( n -- L M H Z ) test ( FE -- 2 0 0 FF )
        clrf    TMP2
        clrf    TMP3
        movwf   TMP1            ; TOS
        movwf   TMP4            ; ZNAK
        btfsc   TMP1,7          ; minus-?
        call    NtoBCDminus
LP_NtoBCD
        movlw   D'100'          ;
        subwf   TMP1,W          ;
        btfss   STATUS,C        ;
        goto    Cont_NtoBCD
        incf    TMP2,F          ; H
        movwf   TMP1            ;
        goto    LP_NtoBCD
CONT_NtoBCD
        movlw   D'10'           ;
        subwf   TMP1,W          ;
        btfss   STATUS,C        ;
        goto    END_NtoBCD
        incf    TMP3,F          ; M
        movwf   TMP1            ;
        goto    CONT_NtoBCD
END_NtoBCD
        movf    TMP1,W          ; L
        decf    FSR,F           ;
        movwf   INDF            ;
```

```
        movf    TMP3,W          ; M
        decf    FSR,F           ;
        movwf   INDF            ;
        movf    TMP2,W          ; H
        decf    FSR,F           ;
        movwf   INDF            ;
        movlw   0x0FF           ;
        btfss   TMP4,7          ;
        movlw   0               ; ZNAK
        return
NtoBCDminus
        comf    TMP1,F          ;
        incf    TMP1,F          ;
        return
```

Работает это слово следующим образом: сначала очищаем вспомогательные переменные. Потом помещаем преобразуемый байт в ячейку TMP1 для преобразования и в ячейку TMP4 для сохранения знака. Следующей командой определим знак числа. Если оно отрицательное (бит 7 байта равен 1) то подпрограммой NtoBCDminus переведем его в дополнительный код, то есть оно станет положительным и равным по модулю исходному. Далее идет определение цифр данного байта. И наконец последними тремя командами еще раз определяется сохраненный знак байта и на верхушку стека кладется «FF» если он отрицательный или «0» если число положительное.

И напоследок, предлагаем самостоятельно разобраться в реальной программе, написанной на языке F2pa и сданной ЗАКАЗЧИКУ (Приложение В). Эта программа обслуживает полуавтомат для поляризации пьезоэлементов и предназначена для управления моторами конвейера, источником высокого напряжения. Программа состоит из настроечной части и регулирующей части. Остальные пояснения приведены в комментариях текста программы в руководстве оператора и файле magpu.4th.

ПРОЕКТНОЕ ЗАДАНИЕ К МОДУЛЮ 3

1. Разработай ассемблерный код слова `NIP ( u1 u2 -- u2 )`, т.е снимающее со стека второе слово (аналог `: NIP ( u1 u2 -- u2 )SWAP DROP ;`).
2. Разработайте слово `INC_BYTE32 ( -- )`, при обращении к которому происходит увеличение на единицу 32-х разрядной переменной находящейся в ячейках `BYTE0, BYTE1, BYTE2, BYTE3`.

ТЕСТОВОЕ ЗАДАНИЕ К МОДУЛЮ 3

Тестовое задание выполняется в течении 5 минут и содержит 7 вопросов (правильный ответ помечается любым значком).

Результаты теста оцениваются по следующей шкале:

7 или 6 правильных ответов – оценка «5»,
5 правильных ответов – оценка «4»,
4 правильных ответа – оценка «3»,
менее 4 правильных ответов – оценка «2».

1. На стеке находятся числа **`n1 n2 n3 n4 --`** **`.`** Какое число находится на вершушке стека:
   а) n1
   б) n2
   в) n3
   г) n4
2. На стеке находятся числа **`n1 n2 n3 n4 --`** **`.`** Какое число будет находится на вершине стека после операций **`+ SWAP`** :
   а) n1
   б) n2
   в) n3
   г) n3+n4
3. Адрес второго элемента на стеке находится в регистре:
   а) W
   б) INDF
   в) PCL
   г) FSR
4. Число **.3** занесено на вершину стека**.** Какое число будет находится на вершине стека после операции **`DROP`**:
   а) 0
   б) 1
   в) 2
   г) 3
5. На стеке находятся числа **`n1 n2 n3 n4 --`** **`.`** Какое число будет вторым от вершины стека после операций **`+ SWAP`** :
   а) n1
   б) n2
   в) n3
   г) n3+n4
6. Какая из строчек команд дублирует второй элемент на стеке (n1 n2 n3 n4 -- команды-- n1 n2 n3 n3 n4):
   а) over drop
   б) dup over
   в) swap dup

г) over swap6.

7. Ассемблерный код какого слова приведен ниже:

```
addwf       INDF,W
incf        FSR,F
```

а) 1+

б) 2+

в) +

г) 1-

Заключение

Кроме рассмотренного семейства PIC-контроллеров существуют еще: PIC10xxx, PIC12xxx, PIC14xxx, PIC17xxx, PIC18xxx, PIC24xxx, dsPIC и другие семейства. Между ними существует преемственность и они перекрывают значительный диапазон технических возможностей [11].

Для правильной разработки законченного изделия остановимся на микросхемах и модулях, удачно применяемых совместно с PIC-контроллерами, так называемой периферии. Применяются:

Модули питания, генераторы сигнала сброса

Оптроны. Цифровые изоляторы.

Операционные усилители. Компараторы

Цифровые потенциомеры

Миросхемы для расширения входов и выходов. Интерфейсные преобразователи.

Микросхемы памяти

Часы.

Термометры.

Остановимся подробнее на первой строчке. Примение модулей питания с гальванической разъвязкой позволяет применять PIC-контроллеры в промышленном оборудовании, работающем в условиях повышенных помех.

Сброс микроконтроллера - важное событие, поскольку служит для запуска программы в определенном порядке. Если напряжение питания нарастает быстро, то никаких проблем не возникает, поскольку внутренняя схема сброса функционирует корректно. Если же электропитание нестабильное, то необходимо позаботится о том, чтобы микроконтроллер находился в состоянии сброса до тех пор пока на будет получено номинальное значение питающего напряжения.

Список литературы

1. Файл «pic_review.doc».
2. Файл «mpasm.pdf».
3. Файл «mplab_ide.pdf».
4. Файл «30569a.pdf» (фирменное описание pic16f870).
5. Файл «pic16f87x.pdf» (перевод фирменного описания pic16f873 и других).
6. Ссылка в Интернете www.forth.org.ru
7. Предко М. Справочник по PIC-микроконтроллерам. М.: ДМК Пресс, Додека-XXI, 2002.
8. Ульрих В.А. Микроконтроллеры PIC16C7X. «Наука и техника». Санкт-Петербург, 2000.-253 с., ил.
9. Тавернье К. PIC-микроконтроллеры. Практика применения. М.: ДМК, 2002.
10. Предко М. Создайте робота своими руками на PIC-контроллере./ Майл Предко; Пер. с английского Земского Ю.В. – М.: ДМК Пресс, 2006. – 408 с.: ил. – (В помощь радиолюбителю).
11. Кениг А. и М. Полное руководство по PIC-микроконтроллерам.: Пер. с нем.-К.: “МК-Пресс”,2007.-256 с., ил.
12. Журнал «Схемотехника» №2 за 2007г, стр. 32. Компиляторы, которые мы выбираем.
13. Файл «Ресурс радиолюбителя.htm»