Приложения

Количество страниц: 68.

Приложение №1

Как изготовить программатор ***PonyProg*** и работать с ним

Для начинающих, программатор ***PonyProg***, на мой взгляд, наиболее предпочтителен.
Он поддерживаетдостаточно большое количество типов микроконтроллеров и микросхем памяти последовательного типа и надежен в работе.
Его аппаратная часть проста в изготовлении, состоит из дешёвых, доступных всем, радиодеталей, и она совместима с программой, обслуживающей программатор ***IcProg105***.
Конечно же, существует и масса других программаторов, но это уже "дальнейшее развитие темы".
Русская версия программы ***PonyProg*** включает в себя 3 архивных файла:
- архив **ponyprogV205a.zip**
- архив **PonyProg2000-Russian.zip**
- архив **Pony.zip**

Они находятся в папке *PonyProg*, которая вложена в папку *Программы*.

Установка русской версии программы *PonyProg*

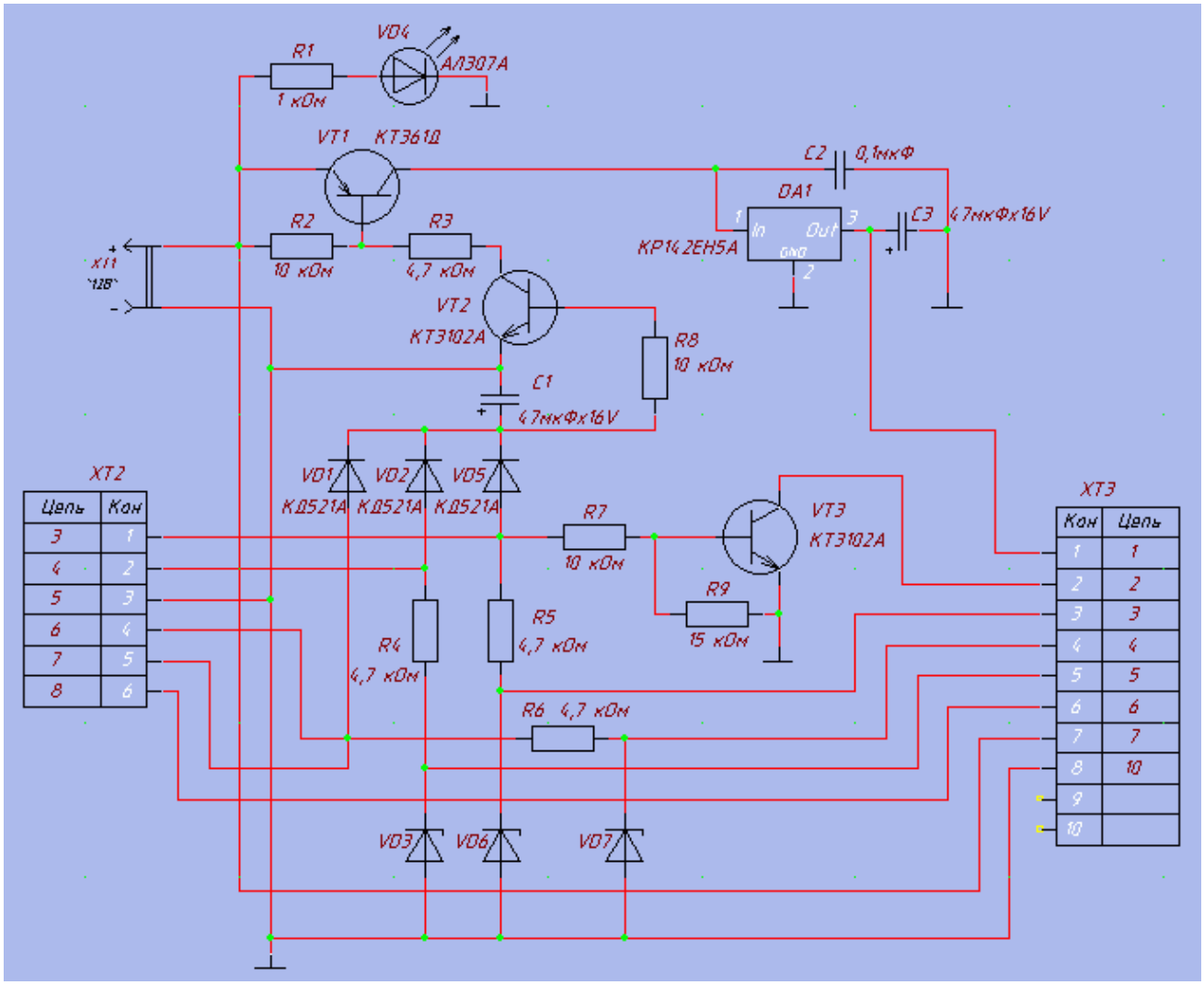
Создайте в папке ***Program Files*** диска ***C*** (или в любом удобном для Вас месте) папку ***PonyProg*** и разархивируйте в нее архивы **ponyprogV205a.zip** и **Pony.zip**.
Русификатор **PonyProg2000-Ruussian.zip** пока разархивировать не нужно.
После этого в папке ***PonyProg*** Вы увидите установочный файл программы: **setup.exe** и 10 файлов с расширением **.gif** .
Сначала необходимо установить английскую версию программы.
Щёлкните по файлу **setup.exe**.
"Запустится" стандартная процедура установки программы.
В ходе этой процедуры, соглашайтесь с тем, что Вам будет предлагаться.
Когда Вам будет предложено выбрать место "дислокации' программы, укажите папку ***PonyProg***, о которой говорилось выше.
Когда установка английской версии программы закончится, в списке программ, появится строка **PonyProg**.
Далее, нужно произвести русификацию.
Для этого, необходимо разархивировать архив **PonyProg2000-Russian.zip** в папке ***PonyProg***.
В процессе разархивации, появится окно с вопросом, нужно ли заменить файл **PonyProg.exe** английской версии программы, на файл русской версии программы?
Жмите на ***Да***.
На этом, процедура установки русской версии программы ***PonyProg*** будет закончена и с ней можно работать.
После этого, указанные выше архивы, можно либо вообще удалить, либо не удалять (если предполагается их использовать в дальнейшем).

Изготовление аппаратной части программатора *PonyProg*

"Материнская", принципиальная схема программатора si-prog-v2_2.pdf прилагается: папка *Программы* → папка *PonyProg*.
Программатор состоит из базового модуля и сменных модулей.
Базовый модуль имеет внешний разъем, к которому, в зависимости от типа микроконтроллера или микросхемы памяти, подключается тот или иной сменный модуль.

Базовый модуль

Найдите на принципиальной схеме схему с надписью **Base board**.
Это и есть базовый модуль.
Можно собрать и его, но я могу порекомендовать Вам собрать то же самое, но на отечественных радиодеталях (лично я, так и сделал).
Хлопот гораздо меньше, и работает он ничем не хуже чем "штатный", базовый модуль.
То, от чего я "отталкивался", выглядит так:



На резисторах **R4,R5,R6** и стабилитронах **VD3,VD6,VD7**, собраны три одинаковых преобразователя уровней (простейшие параметрические стабилизаторы), которые ограничивают амплитуду импульсов, поступающих с COM порта компьютера, до безопасного, для ПИКа, уровня.
Так как ПИКи запитываются напряжением 5V, то уровень ограничения должен быть не выше этого значения.
Поэтому, в качестве стабилитронов **VD3,VD6,VD7**, необходимо использовать стабилитроны **КС147А** с напряжением стабилизации 4,7V (или в крайнем случае, КС139А).
Естественно, что и "забугорные" стабилитроны тоже можно использовать.
На диодах **VD1,VD2,VD5** и конденсаторе **C1**, собрано устройство управления составным транзисторным ключём (транзисторы **VT1** и **VT2**).
Если на анодах **VD1,VD2,VD5** одновременно будет присутствовать нулевой уровень (например, если системный блок компьютера выключен или COM порт не подключен), то составной транзисторный ключ закрывается, и напряжение питания 5V, на программируемую микросхему, не подается.
На транзисторе **VT3** собран ключевой каскад управления сменными модулями.
На микросхеме **DA1** собран стандартный стабилизатор напряжения 5V.
К разъему **XT1**, от внешнего источника питания, подводится постоянное напряжение 12V.
Удобнее всего (лично я, сделал так), иметь внутренний, сетевой источник питания, который смонтирован в корпусе базового блока.
Рекомендую повысить величину выходного напряжения сетевого блока питания с 12V до 14,0 … 14,5V (мотивация – см. ниже).
При помощи разъема **XT2**, базовый модуль соединяется с COM портом компьютера.
То есть, в процессе работы с микроконтроллером или с микросхемой памяти, на контактах этого разъема присутствуют сигналы управления (контакты **6** или **7**), вырабатываемые

компьютером (с помощью программы ***PonyProg***), а также синхроимпульсы и записываемые или считываемые импульсные последовательности (контакты **3** и **4**).
Обмен данными между компьютером и "периферией" осуществляется по шине данных последовательного типа.
Именно поэтому ***PonyProg*** относится к программатором последовательного типа.
При помощи разъема **XT3**, к базовому модулю подключается тот или иной сменный модуль.
Для обеспечения соответствия принципиальной схемы отечественной версии базового модуля и принципиальной схемы "родного", базового модуля (по подключению цепей к разъему сменных модулей)***, в принципиальной схеме отечественной версии базового модуля, нужно "перебросить корпусной" провод с 8-го контакта разъема XT3 на 10-й контакт этого же разъема*** (или поменять местами столбцы с названиями **Контакт** и **Цепь**).
Транзистор VT3 отечественной версии базового модуля задействуется при работе с м/контроллерами AT90Sxx, а при работе с м/контроллерами PIC16Fxx, не задействуется.
По этой причине, 2-й контакт разъема J8 сменного модуля ПИКов, внутри этого сменного модуля, ни к чему не подключен ("висит")***.***
Принципиальную схему базового модуля можно и нужно упростить, исключив из него составной ключ **VT1,VT2,R2,R3,R8** и устройство управления им, собранное на диодах **VD1,VD2,VD5** и конденсаторе **C1**.
Какого-то большого, практического смысла, от введения разработчиками, в принципиальную схему базового модуля, перечисленных выше устройств, лично я, не вижу.
Если в процессе работы программатора
- программируемая м/схема будет запитываться постоянно,
- будет запитываться при выключенном системном блоке компьютера,
- будет запитываться при отсутствии электрического соединения между базовым модулем и COM портом,

то ничего страшного не произойдет.
Наличие стабилитронов и низкое входное сопротивление ключа, собранного на транзисторе **VT3**, практически полностью исключает (кроме совсем уж экстраординарных случаев) выход микросхемы из строя (в результате превышения предельно допустимых значений входных уровней).
Если возникнет необходимость в снятии с нее питающего напряжения 5V, то нужно просто выключить сетевой блок питания.
В своем программаторе, я сделал именно так, и никаких осложнений это не вызвало (лет пять с ним работаю), а конструкция базового модуля существенно упростилась.
Поэтому рекомендую исключить, из принципиальной схемы, **VT1,VT2,VD1,VD2,VD5,C1**, а между точками подключения коллектора и эмиттера удаленного транзистора **VT1**, поставить перемычку.
После этого, принципиальная схема базового модуля станет настолько простой, что ее сможет собрать даже начинающий радиолюбитель, причем, в этом случае, вовсе не обязательно изготавливать печатную плату.
Базовый модуль, с навесным монтажом, будет работать не хуже.
Сетевой блок питания → простейший: силовой трансформатор и диодный мост.
Я использовал **КЦ407** (из-за его малых размеров), но подойдут и другие малогабаритные, диодные мосты.
Если этого под рукой нет, то можно собрать его на диодах.
Так как используется сетевой блок питания, в котором пульсации не сглаживаются, необходимо увеличить емкости конденсаторов **C2** и **C3**.
В своем базовом модуле, я установил **C2 → 500мкф/25v** и **C3 → 1000мкф/16v** (с запасом).
Такие номиналы емкостей выше того, что требуется для нормальной работы программатора, но во-первых, "кашу маслом не испортишь", а во-вторых, такие значения емкостей обеспечивают защиту от кратковременных, резких колебаний напряжения в сети 220V, что немаловажно.
Можно исходить из принципа "чем больше, тем лучше", с учетом обеспечения приемлемых габаритов и материальных затрат.
Силовой трансформатор → малогабаритный (например, от сетевого адаптера).
На входе стабилизатора **DA1**, он должен обеспечивать постоянное напряжение **14,0 … 14,5V**

(можно до 15,0V), а не 12V, как это указано в схеме.
Объяснение этому следующее.
В сменном модуле, с помощью которого осуществляется программирование PIC контроллеров, используется параметрический стабилизатор на 12,5 - 13,0V.
Для его нормальной работы, требуется напряжение выше чем 12,5 - 13,0V.
Также нужно учесть то, что для м/схемы **КР142ЕН5А** (**142ЕН5А**), предельно допустимое, входное напряжение = 15V.
С учетом этого, на вход параметрического стабилизатора и на вход стабилизатора **DA1**, нужно подавать напряжение величиной **14,0 … 14,5V**.
Микросхема **DA1** может работать без радиатора, но при использовании металлического корпуса базового модуля, что я Вам рекомендую (не стоит пренебрегать экранировкой), ее можно механически закрепить в любом удобном месте корпуса.
Разъем **XT3** закрепляется на корпусе базового модуля таким образом, чтобы было удобно вставлять в него сменные модули.
Место установки сетевого тумблера включения блока питания, индикаторного светодиода **VD4** и вывода из модуля сетевого шнура - на Ваше усмотрение.
Рекомендую также убрать разъем **XT2** и распаять провода соединительного кабеля (между базовым модулем и COM портом компьютера) на плате базового модуля, зафиксировав этот кабель хомутиком (разъемные соединения, без которых можно обойтись, не желательны).
Длина этого кабеля не должна превышать 50 см. (чем меньше, тем лучше).
Желательно, чтобы он был экранированным.
На другом конце кабеля, распаивается стандартный, 9-штырьковый компьютерный COM разъем ("папа").
В дальнейшем, соединительный кабель будет подключаться к свободному COM порту.
Существуют также схемы подключения базового модуля к LPT порту.
На первых порах, с LPT портом советую "не связываться", а использовать простой, дешевый и проверенный в работе вариант, который предлагается.
В дальнейшем, если возникнет такая необходимость, можно будет перейти на работу с LPT портом.

Сменный модуль для работы с PIC контроллерами

В картинке файла **si-prog-v2_2.pdf**, в правом нижнем углу, найдите принципиальную схему с названием **PIC adapter**.
Это и есть сменный модуль.
Увеличьте его лупой "Акробата" так, чтобы были видны все детали схемы.
U9,U18,U17 соответственно: **18,28,8** - выводные панельки для микросхем типа DIP (самые распространенные).
На транзисторах **Q2** и **Q5** собран транзисторный ключ.
Если на 3-м выводе разъема **J8** присутствует **1**, то транзистор **Q5** открывается и напряжение 14,0 … 14,5V, с 7-го вывода разъема **J8**, подается на вход стабилизатора, а если **0**, то не подается.
Стабилизатор простейший. Собран он на стабилитроне **Z4** и резисторе **R10**.
С его выхода, напряжение (около **13V**) поступает (или не поступает) на 4-й вывод панельки **U9**.
Для повышения устойчивости работы ключа, к его выходу подключена активно - емкостная нагрузка **R19,C13**.
Резистор **R12** ограничивает базовый ток транзистора **Q2**.
Резисторы **R13** и **R14** образуют делитель напряжения, а напряжение, "падающее" на резисторе **R14**, управляет транзистором **Q5**.
Эти же резисторы являются суммарной, коллекторной нагрузкой транзистора **Q2**.
Резистор **R13**, кроме этого, еще и ограничивает, до безопасного уровня, базовый ток транзистора **Q5**.
На транзисторе **Q3** также собран ключ, управляемый сигналом, поступающим с 5-го вывода разъема **J8**.
На схеме Вы также увидите батарею 9V и переключатель **JP1**.
По замыслу разработчика, они необходимы для того чтобы организовать режим "батарейной" запитки ключа на транзисторах **Q2** и **Q5**.
Я не случайно поставил кавычки в слове "батарейной", так как, в этом случае (выводы 1 и 2 переключателя **JP1** замкнуты), на вход ключа подается суммарное напряжение 14V (5+9=14).
В этом случае, нормальная работа параметрического стабилизатора будет обеспечена.

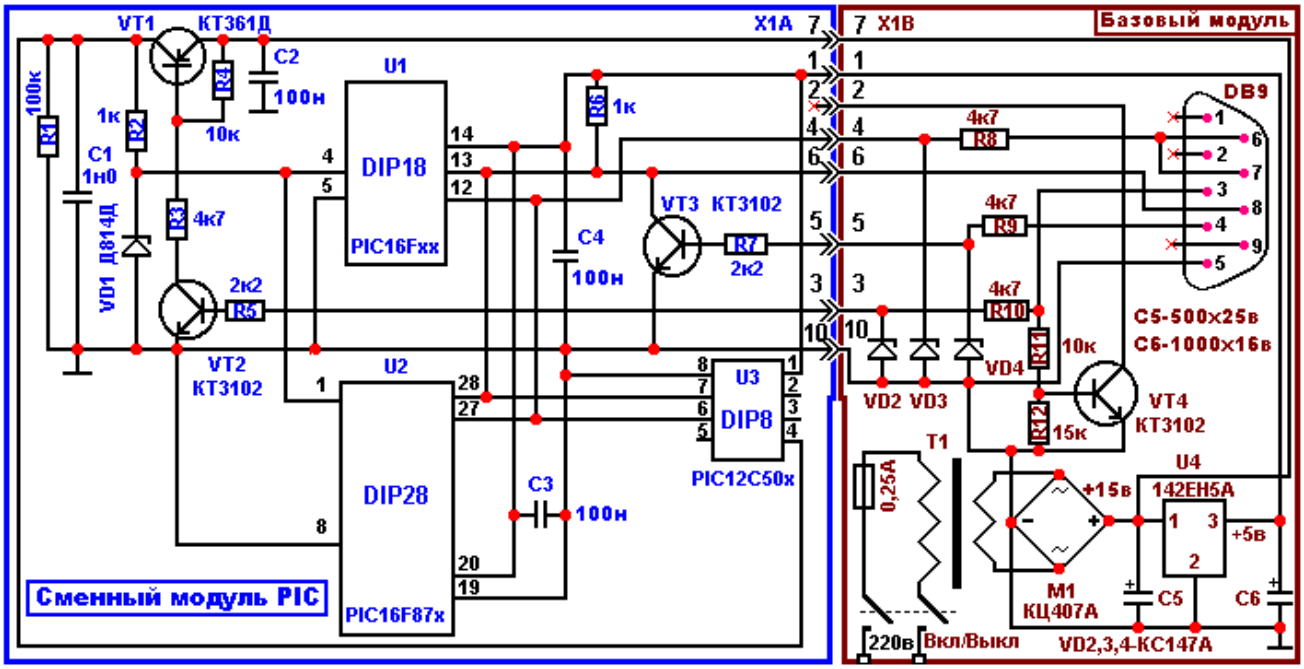
Но это "сработает" тогда, когда в наличии будет 5V, а 14,0 … 14,5V в наличии не будет. Вероятность такого события близка к нулю.
Если произойдет какая-то "бяка" с блоком питания, то "вырубятся" оба этих напряжения и программатор, по-любому, работать не будет.
Единственное разумное объяснение этому состоит в том, что от такого нововведения повысится помехоустойчивость, но она и без этого вполне прекрасна.
Вывод: батарею и переключатель **JP1** можно смело убрать. От этого ничего не ухудшится.
Гораздо проще и удобнее собрать этот сменный модуль на отечественных радиодеталях. Лично я, так и сделал. Советую то же самое сделать и Вам.
Модуль на отечественных радиодеталях работает ничем не хуже своего "родителя" и с его сборкой меньше хлопот.
Изменения, вносимые в схему:

- заменить транзистор **Q5** с BC557 на **КТ361** (с любой буквой. Я использовал с буквой "Д"),
- заменить транзистор **Q2** и **Q3** с BC547 на **КТ3102** (с любой буквой. Я использовал с буквой "А")
- в качестве стабилитрона **Z4**, используется **Д814Д** или любой другой, с напряжением стабилизации **12,5 … 13,0V**. Это "ответственный" радиоэлемент. Не поленитесь подобрать его по напряжению стабилизации.
- увеличить номинал **R16** с 1 ком до **2,2 ком**.
- исключить из принципиальной схемы батарею 9V и переключатель JP1. После этого, нужно подключить точку соединений эмиттера транзистора **Q5**, резистора **R14** и конденсатора **C21**, к **7-му выводу разъема J8**.
- остальное без изменений.

После того, как программа ***PonyProg*** будет установлена и будут изготовлены базовый модуль и сменный модуль, предназначенный для работы с PIC контроллерами, с ними можно работать.
Так как в контексте заявленной темы, в мои планы не входит "въезд" в микроконтроллеры других фирм, то на этом и ограничусь.
Если кто-то из Вас работает с другими типами микроконтроллеров, то изготовьте соответствующие сменные модули (см. принципиальную схему).
Они еще проще, чем сменный модуль для ПИКов.

Принципиальная схема базового модуля и сменного модуля для программирования PIC контроллеров.

Доработанный (в соответствии со сказанным) вариант принципиальной схемы базового модуля и сменного модуля для программирования PIC контроллеров, которые я продолжительное время использую в своей работе:

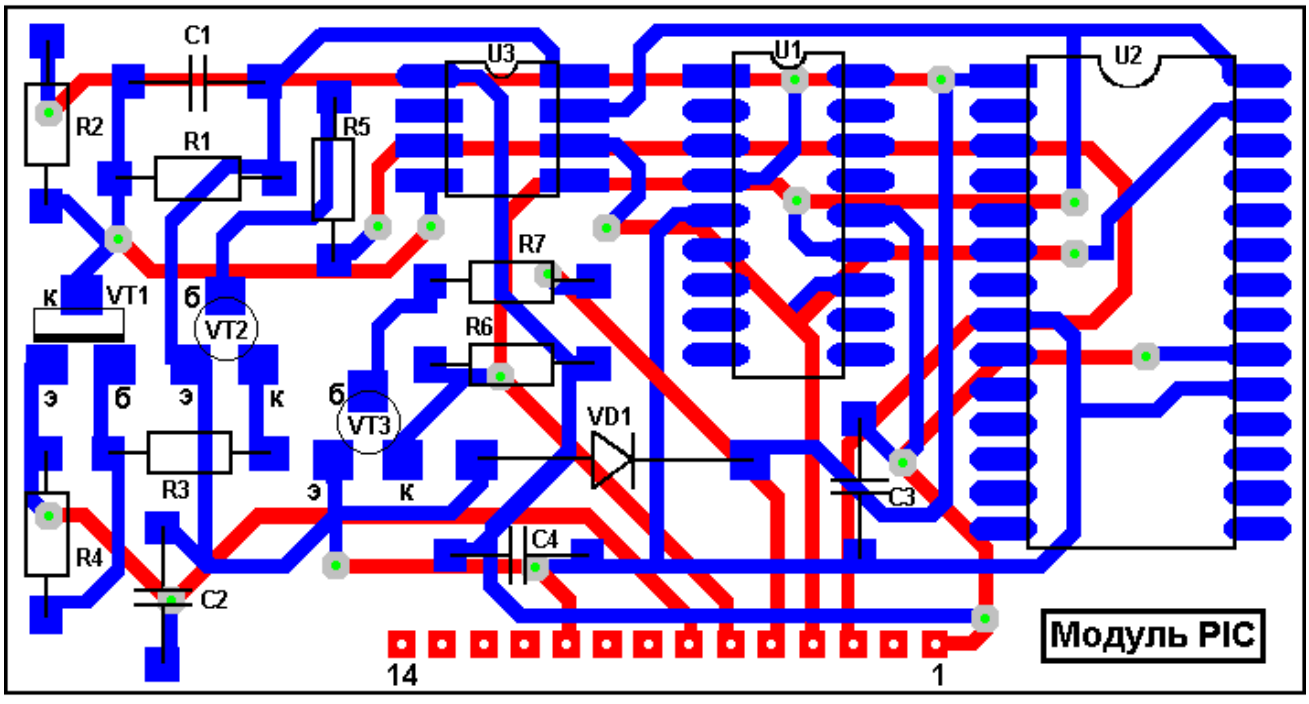


Примечание: в схеме базового модуля, не указан контрольный светодиод (контроль наличия U=5v.) с его гасящим резистором.
Для того чтобы не перегружать маломощный сетевой трансформатор, желательно выбрать номинал этого гасящего резистора не менее 1 ком.
Принципиальная схема базового модуля настолько проста, что можно обойтись навесным монтажом.
Мало того, если предполагается работать только с PIC контроллерами, то транзистор **VT4** и резисторы **R11** и **R12** можно исключить из принципиальной схемы.
Ниже, Вы найдете несколько печатных плат, из которых можно выбрать то, что Вам подходит.

Двухсторонняя печатная плата сменного модуля для программирования PIC контроллеров

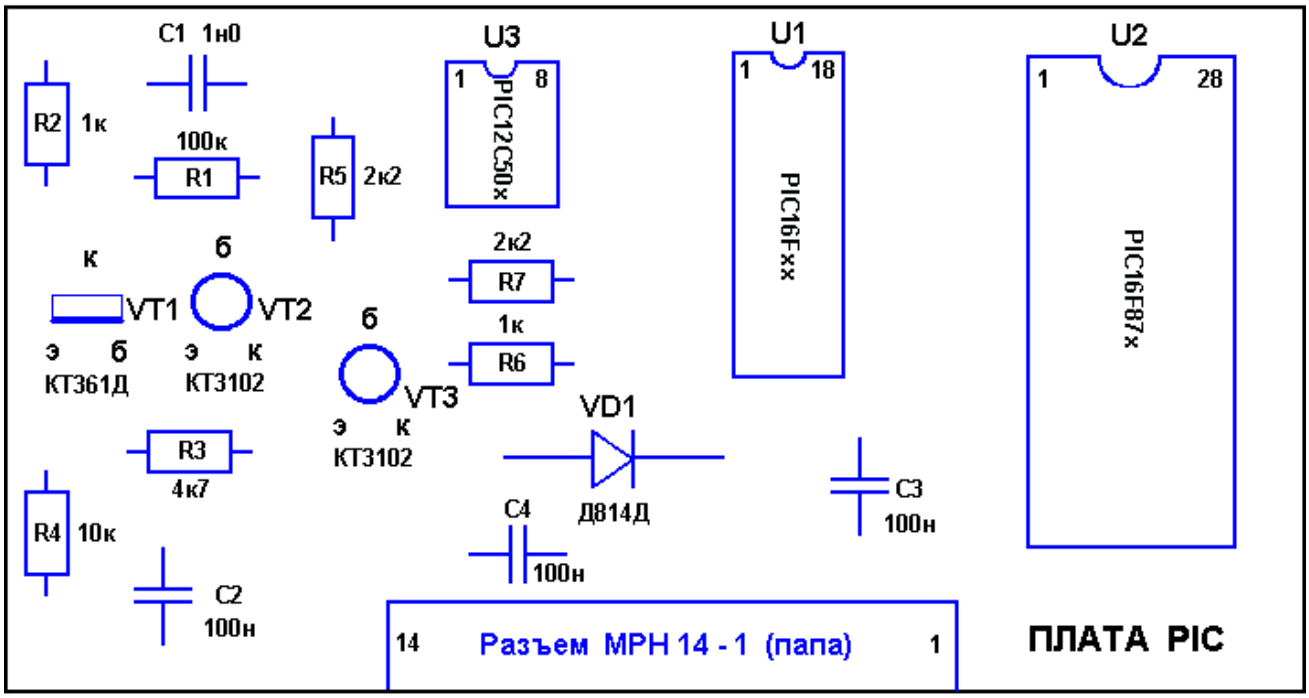
Разработана в ***P-CAD***. Размер платы 95x50 мм. Показана в увеличенном виде.
Этот сменный модуль я длительное время использую в работе.
Замечаний по его работе у меня нет (пользуюсь с 2004-го года).
Даже панельки не менял (по состоянию на 2008г.).
Применен поверхностный монтаж радиодеталей и DIP-панелек.
Выводы DIP-панелек отгибаются на 90 градусов наружу, и в таком виде они и припаиваются к контактным площадкам.
DIP-панельки обычные (условно, "копейка штука").
Если возникнет необходимость в их замене на более "свежие", то при таком способе монтажа, это делается довольно-таки просто (при помощи лезвия или чего-то подобного).
Остальные радиодетали подпаиваются к контактным площадкам таким же образом.
Серые кружочки - переходы с одной стороны платы на другую (запаиваются перемычки).
"Мама" разъема **MPH14** установлена в базовом модуле, а "папа", на плате сменного модуля.

Печатная сменного PIC-модуля с наложением всех слоев:



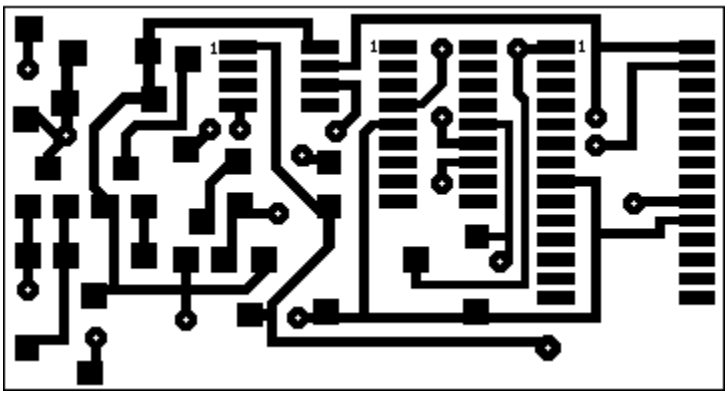
Правда не обошлось без конфуза: когда "ваял" эту плату в ***P-CAD***, по невнимательности "ткнул" не в ту панельку, и поэтому вместо узкой панельки **DIP28**, получилась широкая.
"Извернулся" так.
Левый ряд выводов распаял "как положено".
Вырезал из слюды прокладку и поместил ее между панелькой и платой.
Правый ряд выводов панельки **DIP28** удлиннил и соответствующим образом распаял.
Все это "изворачивание" работает до сих пор.

Расположение радиодеталей:

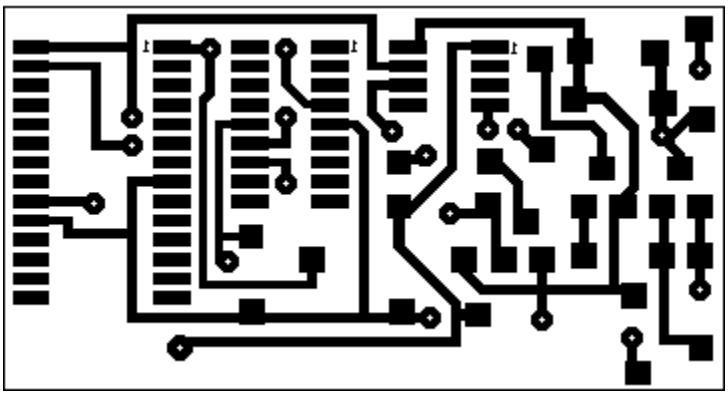


Картинки сторон печатной платы PIC-модуля "на все случаи жизни" (для различных технологий изготовления печатных плат).

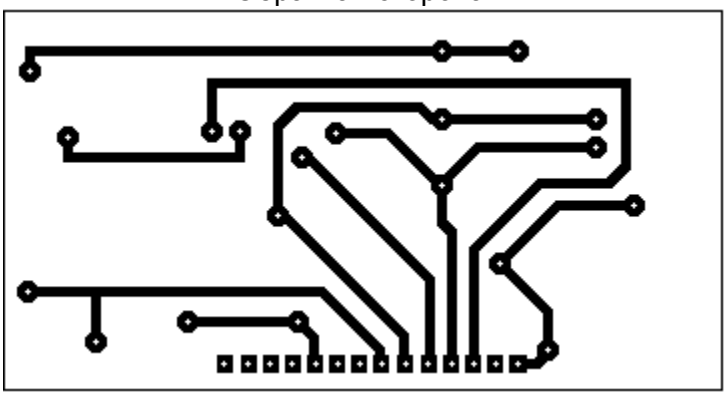
Сторона установки радиодеталей:



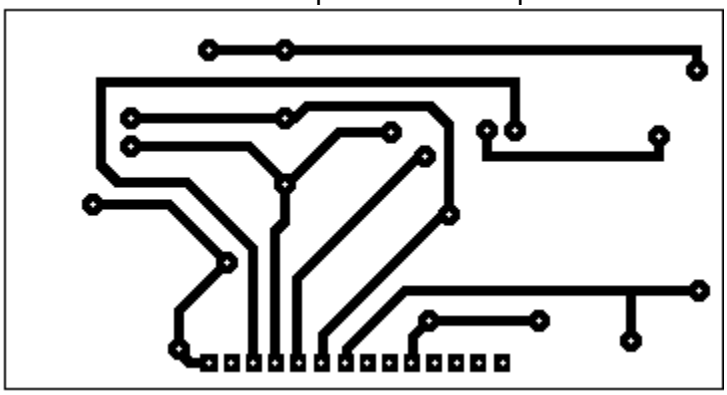
То же самое в зеркальном отображении:



Обратная сторона:



То же самое в зеркальном отображении:



Односторонняя "печатка" **от Михаила Лукьянова**.

Сменный модуль для PIC контроллеров с односторонним монтажом.

На рис.3, **красным цветом** показаны перемычки.

Из панелек нужно удалить те контакты, которые не задействованы.

А именно, №№: **Dip8 → 5, Dip18 → 6, Dip28 → 4, 7, 18.**

(Можно удалить все, кроме рабочих). Масштаб картинок в “Ворде”: 1:1.

Рис.1. Вид со стороны установки деталей.

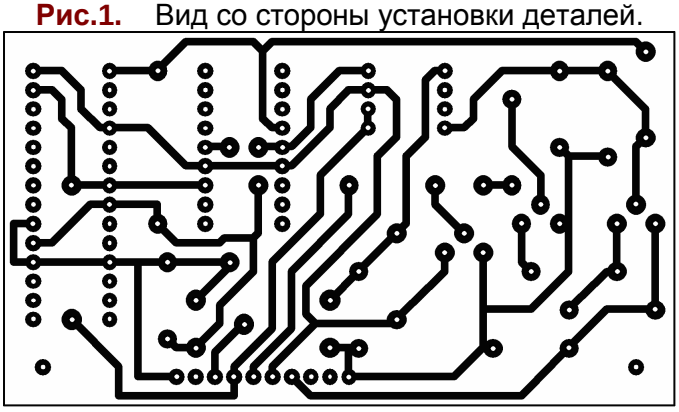


Рис.2. Вид со стороны дорожек.

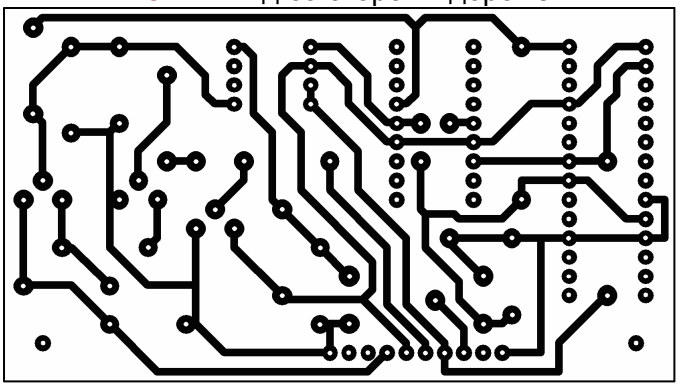
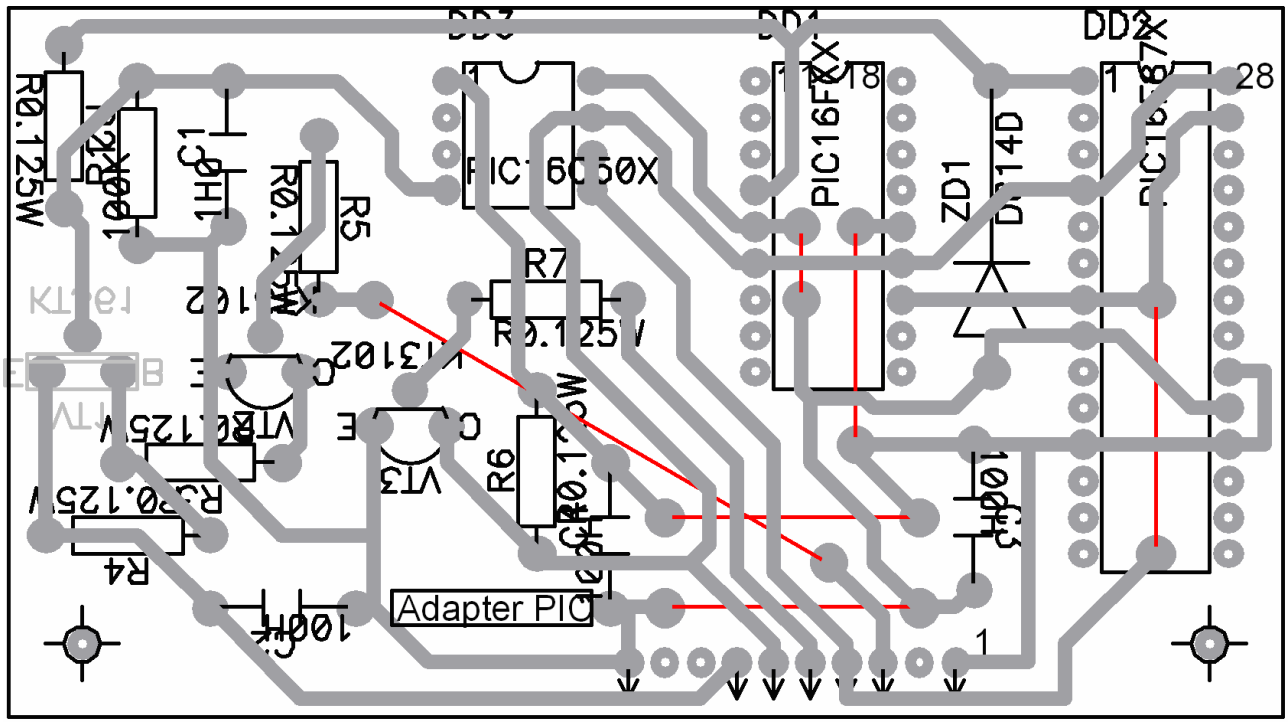


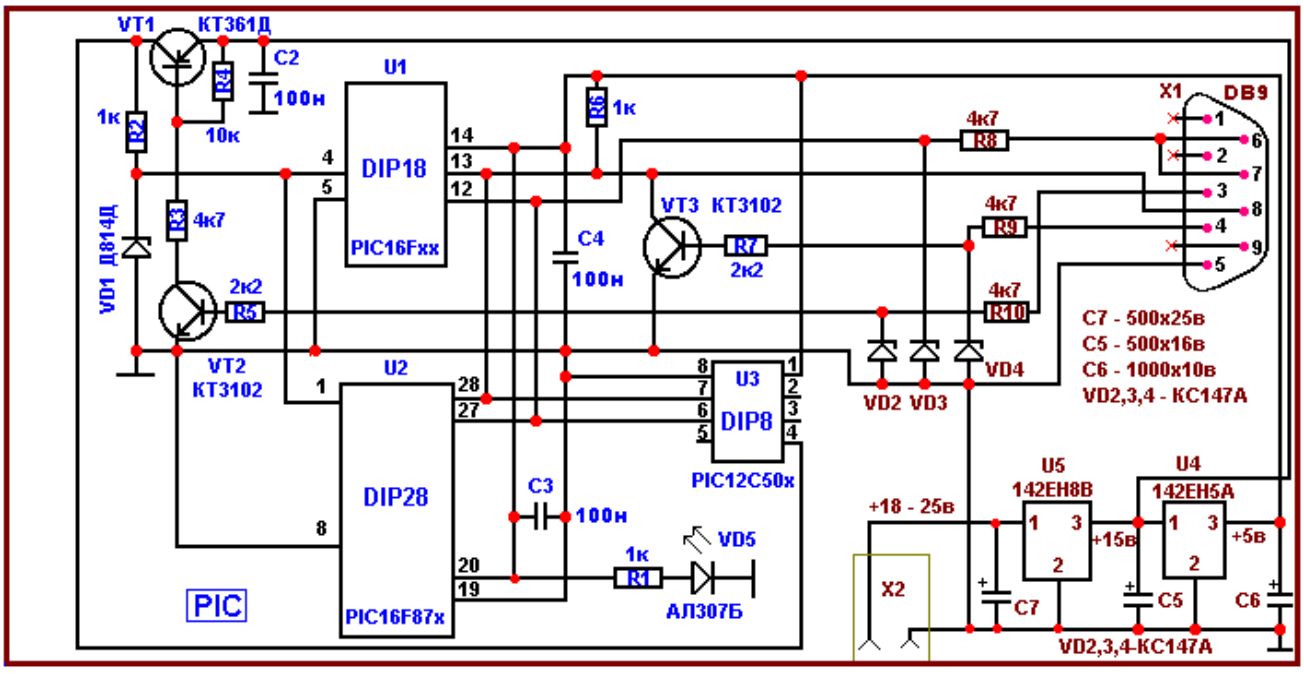
Рис.3. Расположение деталей (вид со стороны установки деталей).



Одномодульная конструкция программатора для PIC контроллеров

Особенностью этого программатора является то, что базовый модуль и сменный модуль объединены на одной печатной плате, транзистор **VT4**, резисторы **R11**, **R12** удалены (при программировании ПИКов они не используются).
Также удалены резистор **R1** и конденсатор **C1**.
Программатор прекрасно работает и без них.
Примечание: эти названия р/деталей относятся к принципиальной схеме, приведенной выше.

Измененная принципиальная схема выглядит так:



Применена двойная стабилизация (позаимствовано из конструкции Станислава Гетманского См. подраздел этого раздела с названием "Дополнительная информация").
С внешнего блока питания (с выхода диодного моста) постоянное напряжение +18...25в подается на разъем **X2**.
Контроль включения программатора осуществляется при помощи светодиода **VD5** (АЛ307Б или любой подходящий).

Работа с PIC контроллерами в программе PonyProg

В качестве примера, я буду использовать **PIC16F84A**.
При выключенном компьютере, подключите разъем базового модуля к разъему свободного COM порта.
Вставьте **PIC16F84A** в панельку сменного модуля, а затем вставьте сменный модуль в разъем базового модуля.
Включите компьютер.
Подключите базовый модуль к сети.
Откройте программу ***PonyProg***.
После этого, Вы увидите заставку программы с симпатичной лошадкой.
Ничего особо интересного на этой заставке нет, поэтому щёлкайте по ***OK***.
После этого, Вы увидите окно программы и два предупреждения о том, что прежде чем работать с программой, нужно произвести
- калибровку
- и настройку оборудования.

Примите к сведению эти предупреждения и уберите их, щелкнув по ***OK***.
Для удобства, раскройте окно программы на весь экран обычным "Виндовским" способом.
В верхней, правой части окна программы, на одной линии, Вы увидите две строки: выбора вида и типа микроконтроллера (или микросхемы памяти).
Они снабжены кнопками, щёлкнув по которым, в выпадающем списке, можно просмотреть те виды и типы микроконтроллеров (и микросхем памяти), которые поддерживает программа ***PonyProg***.
В левом списке, нужно выбрать ***PIC 16 micro***, а в правом списке, ***PIC16F84A***.
Далее, выполняется процедура настройки программы (то, о чем программа предупреждала ранее).
В главном меню программы, щёлкните по кнопке ***Установки***.
В выпавшем списке, состоящем из двух пунктов, щелкните по пункту ***Настройка оборудования***.
Раскроется окно ***Настройка платы программатора***, в котором единственное что нужно сделать → поставить точку в кружочке с номером того COM порта, к которому подключена аппаратная часть программатора.
Остальное → по умолчанию (из выпадающего списка должна быть выбрана строка ***Si Prog API***, а в списке ***Выбор полярности сигналов управления***, все квадратики должны быть без галочек).
Затем щелкаете по кнопке ***Проверка***, после чего раскрывается маленькое окошко **Notice** с надписью ***Тест ОК*** (все в порядке).
Закройте окошко ***Notice***, щелкнув по ***OK***, и окно ***Настройка платы программатора***, еще раз щелкнув по ***OK***.
На этом настройка оборудования завершена.
Если в окошке ***Notice*** присутствует надпись ***Тест Ошибка***, то необходимо убедиться в том, что в настройках Вашего компьютера, выбранный Вами порт включен, свободен и базовый модуль программатора подключен именно к нему.
Если это так, то нужно поменять полярность сигналов управления или выбрать в выпадающем списке другую строку, каждый раз проводя тест, до получения сообщения ***Тест ОК***.
Естественно, что параметры настройки оборудования зависят от конкретного компьютера, но в подавляющем большинстве случаев, настройки именно те, которые были указаны выше.
Далее производим калибровку.
Для этого еще раз нужно щелкнуть по кнопке ***Установки*** и из выпадающего списка, выбрать пункт ***Калибровка***.

Программа сначала спросит разрешения на запуск калибровки.
Нажмите ***Yes***.
После этого, программа ***PonyProg*** начнет "договариваться с операционной системой о порядке их совместной работы".
Лично я, не наблюдал случаев, чтобы они не "договорились".
В результате, через определенное время, откроется окошко ***Notice*** с надписью ***Калибровка завершена***.
Закройте его, щелкнув по ***OK***.
Программатор готов к работе.

Запись данных в PIC контроллер и считывание данных из него

В ПИКах имеется два вида энергонезависимой памяти: память программ и память данных.
Запись в них можно производить многократно.
В память данных записываются данные, которые необходимо сохранить после выключения питания, с целью их использования после следующего включения питания.
В нее может также записываться информация об авторе программы, а также техническая информация (если в ней остается достаточно свободного места, то многие программисты так и поступают).
Объем ее не велик. Для **PIC16F84A**, он составляет 64 байта.
Объем памяти программ гораздо больше.
Память программ **PIC16F84A** позволяет разместить в ней до 1024 команд (максимум).
Именно в память программ и записывается программа.
Строго говоря, в ПИК записывается так называемая "прошивка" (HEX-файл программы), который программой ***PonyProg*** переводится в машинные коды.
Приступаем к работе.
Напоминаю, что на этот момент, выбран **PIC16F84A**, произведены необходимые установки, программатор подключен и в него вставлен **PIC16F84A**, в память программ и в память данных которого необходимо произвести запись.
Сначала нужно **открыть HEX-файл**.
В главном меню программы ***PonyProg***, щелкните по кнопке ***Файл*** и выберете из выпадающего списка строку ***Открыть файл с данными*** или щелкните по пиктограмме с желтой, открытой папкой (при наведении на нее стрелки, появляется всплывающая подсказка ***Открыть файл с данными***).
После этого, раскрывается стандартное окно открытия файлов.
Перед просмотром файлов, из выпадающего списка ***Тип файлов*** (в нижней части окна открытия файлов), нужно выбрать формат ***.HEX***.
Далее нужно отыскать то место (на жестком диске компьютера или на внешних носителях), где "лежит" HEX-файл, который нужно открыть, и щелкнуть по нему, а затем, по кнопке ***Открыть***.
Окно открытия файлов свернется, и в окне программы ***PonyProg***, Вы увидите то, что предназначено для записи в ПИК.
Запись производится следующим образом.
Необходимо либо щелкнуть по пиктограмме ***Записать устройство*** (ориентируйтесь по всплывающей подсказке), либо по строке ***Записать всё*** (в ***Командах***).
Дождитесь окончания процесса записи и проверки (верификации), в конце которого выдается сообщение об успешной записи.
Если выдается сообщение об ошибке записи, щелкните по пиктограмме с нарисованным на ней замком, и в открывшемся окне битов конфигурации посмотрите, есть ли галочка в маленьком окошке с названием ***CP*** (бит защиты).
Если она установлена, то сначала запишите, а потом считайте установленные программой биты конфигурации (менять их или вносить какие-либо изменения не нужно).
После считывания убедитесь, что считано то же самое, что записано.
Эта процедура бывает необходимой в случае установки программой бита защиты (подробнее об этом, ниже).
В других случаях (а таких большинство), это делать не обязательно.
После этого, окно битов конфигурации нужно закрыть (щелкнуть по ***OK***) и повторить процедуру записи.
Если после сборки программатора, постоянно выдаются сообщения об ошибках записи, то

"зайдите" в ***Настройку оборудования*** (см. выше), откройте окно ***Настройка платы программатора*** и попробуйте изменить настройки, каждый раз после этого контролируя результат записи до получения сообщения об успешной записи.
Несколько слов о **стирании** (строка ***Стереть*** в ***Командах*** или пиктограмма с нарисованным на ней ластиком и всплывающей подсказкой ***Erase all the device to FF*** - установить во всех байтах FF).
Перед записью, производить стирание вовсе не обязательно.
Дело в том, что запись производится "по верху" старых данных и по всему объему памяти.
По этой причине, при записи "новых" данных, "старые" данные автоматически "уничтожаются".
Стирание может оказаться полезным в том случае, если PIC контроллер передается кому-то, и при этом нужно, чтобы этот "кто-то" не смог считать то, что ранее было в нем записано.
Для того чтобы считать данные из энегонезависимой памяти PIC контроллера, нужно в ***Командах***, щелкнуть по строке ***Считать всё*** или по пиктограмме с всплывающей подсказкой ***Считать устройство*** и дождаться окончания процесса считывания.
Если запись производится успешно, то в подавляющем большинстве случаев, это является гарантией того, что и считывание будет произведено успешно.
Если, все-таки, Вы получите сообщение об ошибке, то повторите считывание еще раз.
Если Вы слишком часто получаете сообщения об ошибках чтения (это же относится и к записи), то займитесь экранировкой соединения между базовым модулем программатора и COM портом или сделайте его короче.
После окончания успешного процесса считывания, Вы не должны заметить никаких изменений в окне программы ***PonyProg***.
Это говорит о том, что то, что перед считыванием было записано, то и считано.
Основную часть окна программы ***PonyProg*** занимает содержимое памяти программ.
Содержимое памяти данных находится в самом низу (нужно сделать прокрутку).

О бите защиты.

После открытия HEX-файла, откройте окно установки битов конфигурации и обратите внимание на квадратик с надписью ***CP*** (бит защиты).
Если он пустой, то бит защиты не установлен и считывание (после записи) содержимого ПИКа возможно (см. выше).
Если в нем стоит галочка, то бит защиты установлен программой.
В этом случае, во избежание "выдачи" программой ***PonyProg*** сообщений об ошибках записи, перед записью, произведите запись-чтение битов конфигурации.
Если бит защиты установлен программой (в этом случае, его можно снять только в программе, а не в ***PonyProg***), считывание программатором (любым) данных, из энергонезависимой памяти микроконтроллера, не представляется возможным (считаются нули).
На работе программы внутри микроконтроллера, это никак не отразится.
Если у Вас есть программа (HEX-файл), в которой бит защиты не установлен, и Вам нужно сделать так, чтобы не допустить тиражирование "прошивки" (HEX-файла) кем-то другим, то Вы можете, перед записью программы в ПИК, в программе ***PonyProg***, установить бит защиты.
Для этого, нужно открыть окно битов конфигурации и поставить галочку в квадратике ***CP***.
После этого, необходимо произвести запись битов конфигурации, а затем, их контрольное чтение.
Далее → процедура записи (см. выше).
После ее завершения, для того чтобы убедиться в том, что считываются только нули, нужно произвести считывание.
Если в дальнейшем кто-то попытается снять бит защиты, то ПИК необратимо заблокирует исполнение программы (устройство перестанет работать и считать что-то толковое всё-равно не удастся).
По этой причине, бит защиты, записанный в процессе "прошивки" ПИКа, снимать бессмысленно.

Дополнительные возможности программы PonyProg

Основная из них - возможность редактирования HEX-файла.
Таким редактированием имеет смысл заниматься в том случае, если точно известно, что именно и как именно нужно редактировать.
Надобности в этом почти нет.
Есть один случай, когда такая надобность может возникнуть.
Об этом будет рассказано позднее.
Начинающим, я не советую "забивать этим голову".
После открытия любого HEX-файла, в правой части окна программы ***PonyProg***, Вы увидите результат работы простенькой программы - раскодировщика, который переводит числовые значения байтов в стандартные символы.
В данной версии ***PonyProg***, программа отображает только английскую раскладку (на месте символов русского алфавита будут прочерки).
Даже если бы она отображала и русскую раскладку, то практический смысл такой раскодировки достаточно "туманен", так как исполняемая часть программы ни на один язык народов мира не переводится.
Что-то читаемое и осмысленное может быть только в том случае, если это читаемое и осмысленное "заложено" в текст программы программистом (например, фамилия автора программы).
Есть еще несколько "архитектурных излишеств", без которых вполне можно обойтись.
Кому интересно и есть время → пощелкайте.
Примечание: приведенная выше информация, конечно же, не является детальной.
Я и не ставил перед собой такую задачу.
Она рассчитана на тех из Вас, которым нужно, особо не вдаваясь в тонкости программирования, просто "прошить" ПИК или считать из него данные, а также и на тех, кто только начинает серьезный "въезд" в программирование.
Надеюсь на то, что количество последних будет внушать оптимизм.

Дополнительная информация

Информация от **Станислава Гетманского** из г. Самара
(статья автора).

PonyProg-RUS

Программатор не имеет каких-либо оригинальных решений и мало чем отличается от классического варианта программатора ***PonyProg***.
Просто однажды я решил сам собрать схему программатора и «полопатив» Интернет, пришел к выводу, что для домашней сборки, а также по возможностям работы с разными контроллерами (***PonyProg*** поддерживает весьма большое количество контроллеров), самым оптимальным является именно ***PonyProg***.
Очень важную роль при этом сыграло то, что необходимое программное обеспечение, а также руководство по использованию, легко доступны на сайте Евгения Корабельникова.
За это отдельное спасибо ему и всем тем, кто настолько силен по жизни, что может поделиться с другими.
Немного переработав схему с учетом рекомендаций различных авторов, я пришел к этому варианту, который и предлагаю всем тем, кто, возможно, делает первые шаги в направлении освоения микроконтроллеров.
Программатор очень неприхотлив в плане компонентов – резисторов и транзисторов.
Его сможет собрать даже школьник, «дома на коленке».
И если будут использованы исправные детали, то все у вас заработает “с пол-оборота”.

СХЕМА

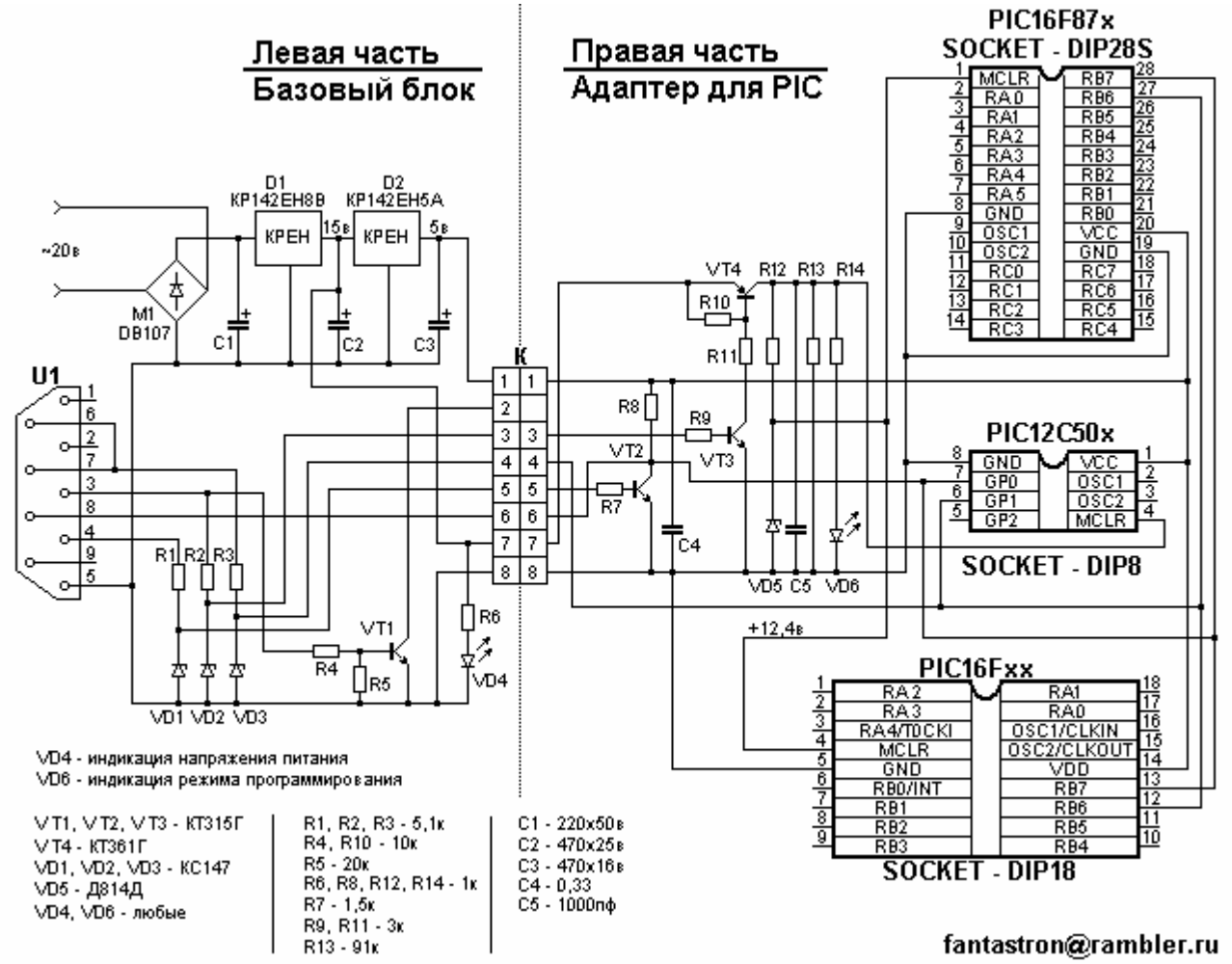
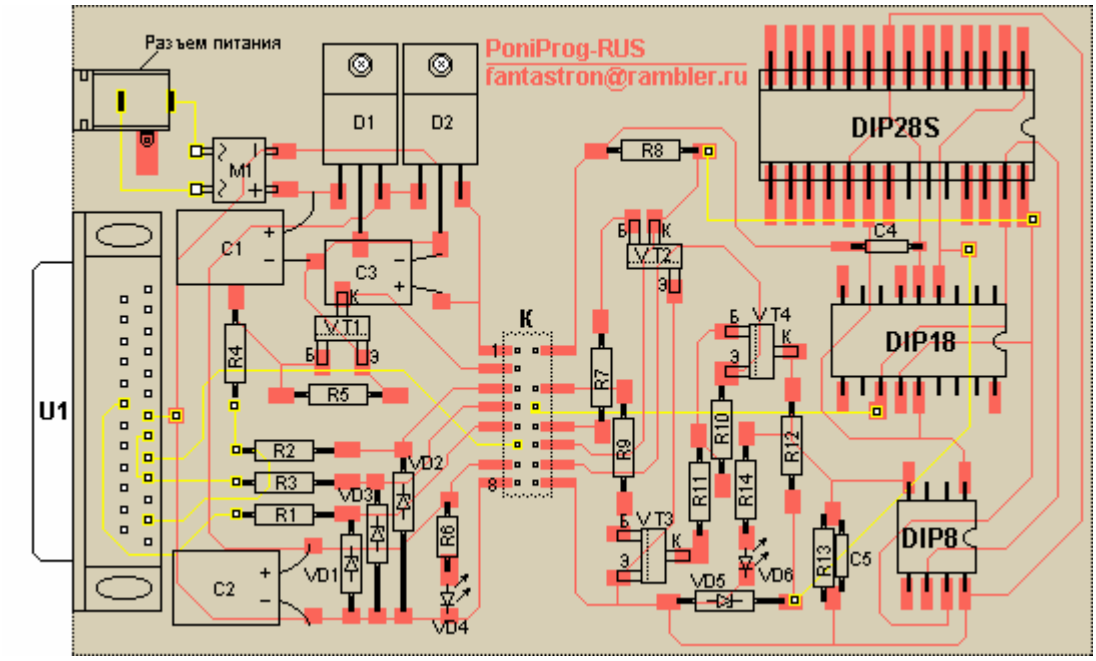
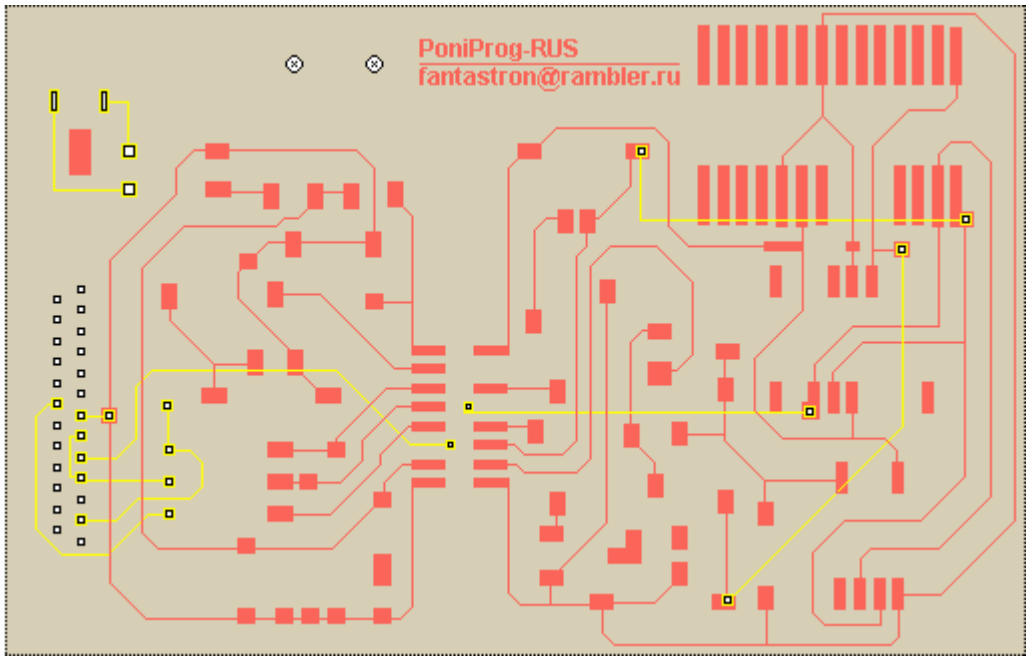


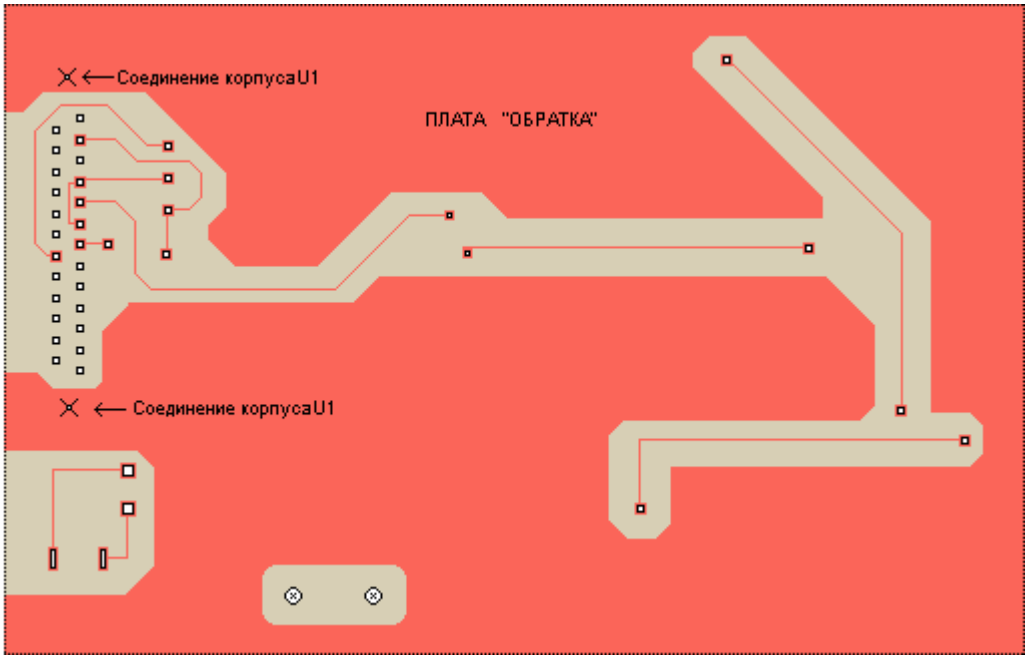
Схема состоит из двух условных частей, левой и правой.
Левая часть это базовый блок, правая это модуль, предназначенный для программирования PIC-контроллеров. Условная линия раздела проходит по центральному разъему «К».
Правая часть подключается к левой части перемычками - стандартными компьютерными джемперами, устанавливаемыми на разъем «К».
Подобное разделение вызвано тем, что если у вас возникнет необходимость работы с другими контроллерами, то, сняв все перемычки, можно подключить к базовому блоку другой модуль.
Если вас не интересуют другие контроллеры, то разъем можно вообще не устанавливать.
Сама по себе схема чрезвычайно проста, и в силу того, что все транзисторы работают в ключевых режимах, никакой особой настройки не требуется.
Единственно на что необходимо обратить внимание, **ЭТО ВАЖНО**, так это на входной разъем **U1**. Дело в том, что лично у меня на плате программатора установлен 25 пиновый разъем (“мама”), поэтому, на приведенной ниже плате, тоже 25 пина.
Просто мне так удобнее по причине того, что я отключаю свой модем, который «висит» у меня на COM2 и вместо модема подключаю программатор.
Так как в модеме у меня разъем 25 пин, то и, естественно, в программаторе тоже.
Кстати, питание программатора я так же беру от блока питания модема.
На схеме показан способ подключения к стандартному 9 пиновому разъему.
То есть, скажу проще: если у вашего компьютера порт COM имеет 9 пиновый разъем, то верхний конец резистора **R1** должен быть соединен с выводом 4 вашего разъема на компьютере и так далее.
Питание программатора: переменное напряжение **14 – 30в**.

В общем, с подключением вам будет нужно немного подумать, то есть, найти компромиссный вариант, устраивающий лично вас.
Помните только то, что если будете формировать кабель из отдельных проводов, не делайте его длинным (не более 50см), так как возможны помехи и, как следствие, ошибки при записи/чтении. У меня, как я говорил, программатор подключается вместо модема и, соответственно, используется кабель гарантированного качества.

МОНТАЖ (масштаб 1:1)







Монтаж программатора не сложен. Например, у меня, от покупки деталей и до первого запрограммированного PICa, ушло 3 дня.
Плата изготовлена из двухстороннего фольгированного стеклотекстолита.
Желтым цветом показаны дорожки на оборотной стороне.
Большинство элементов установлено методом поверхностного монтажа, то есть, они припаиваются непосредственно к предназначенным для них контактным площадкам.
На плате, под стабилизаторами **D1** и **D2**, показаны отверстия.
Рекомендую сделать их на тот случай, если Uпит. программатора будет превышать 25в.
При этом, возможно, придется установить один общий радиатор для **D1** и **D2**, в виде, например, дюралевого уголка.
Вся металлизация обратной стороны платы, за исключением показанных **желтым цветом** дорожек, у меня подключена к корпусу разъема U1.
Тем самым “обратка” платы, есть, по сути, продолжение экрана кабеля и корпуса компьютера.
Так как печатная плата нарисована в стандартном WINDOWS-овском редакторе **Paint** (расширение .bmp), то вы можете в этом же редакторе подредактировать плату с учетом вашего варианта подключения к компьютеру, а затем, любым доступным для вас способом, изготовить физическую плату как таковую, и напаять на нее необходимые элементы.
Центральный разъем «К» подпаивается к верхней части платы, и только выводы 6-левый, 4-правый, пропускаются сквозь плату и припаиваются с обратной стороны.

ДЕТАЛИ

МС стабилизаторов можно заменить любыми импортными аналогами.
Транзисторы подойдут любые биполярные соответствующей проводимости, с максимальным U коллектор - эмиттер не менее 20в.
Мощность резисторов 0,125вт, но можно использовать и резисторы другой мощности.
Номиналы резисторов:
R1, R2, R3 от 4 до 5,5к., **R4** от 7 до 12к., **R5, R10** от 10 до 30к., **R6, R14** от 800ом до 1,5к., **R7, R8, R9, R11** от 1 до 3к., **R12** от 1 до 1,5к., **R13** от 50 до 100к., этот резистор можно не ставить вообще.
Стабилитроны **VD1, VD2, VD3** – КС147А или любые другие, малой мощности на U стабилизации = 4,7в.
VD5 – малой мощности, на U стабилизации от 12,5 до 13,5в. Этот стабилитрон желательно подобрать по указанному напряжению. На схеме я указал отечественный вариант.
По справочникам подобных стабилитронов очень много.

Сам я поступил проще. Зашел в магазин и попросил продать мне то, что есть в наличии на 13в. В итоге у меня на плате стоит симпатичный и маленький.
Светодиоды любые, по вкусу.
Емкости всех конденсаторов могут отличаться от указанных на плюс-минус 50%.
Диодный мост **M1** любого типа на ток от 0,3А и U от 30в и выше.
Если вы предполагаете питать программатор от постоянного напряжения, то **M1** не нужен, хотя, даже в этом случае, мост рекомендую поставить.
Он часто спасает, если вы вдруг перепутаете полярность подключения питания, что бывает даже у опытных специалистов, поэтому лучше все-таки поставить мостик, стоимость его не велика, в районе 10 руб.
Разъем «К»: не знаю точно, как он называется, но это такая вилка, “папа”, наподобие той, которая стоит на материнской плате компьютера и к которой подключается шлейф от жесткого диска. Расстояния между штырьками 2,5мм.

НАЛАДКА

По существу, никакой наладки и не требуется. Нужно только, прежде чем подключаться к COM-порту, осуществить проверку, которая заключается в следующем:
Устанавливаются все 7 джемперов (перемычек) на разъем «К».
Подключается питание.
Затем, вольтметром измеряются напряжения на выходах стабилизаторов.
Оно указано на схеме.
Если все нормально, то возьмите кусок провода. Один конец подключите к выходу **D2**, то есть к +5в, а другим концом поочередно прикасайтесь (только поаккуратнее) к верхним концам резисторов **R1, R2, R3**, или к соответствующим клеммам разъема **U1**.
При касании **R2** у вас должен загораться светодиод **VD6**, при этом желательно, чтобы к стабилитрону **VD5** был подключен вольтметр: так вы убедитесь, что U на стабилитроне соответствует указанному.
Затем подключите вольтметр к резистору **R8** и подайте +5в на верхний конец **R1**.
При этом вольтметр должен изменить свои показания от 0 до 5в, то есть происходит открывание транзистора **VT2**.
Если после всех этих нехитрых манипуляций у вас все загорается и открывается, значит, все нормально.
Это гарантия того, что и ваш компьютер, через COM-порт, сможет управлять вашим программатором, даже если ваш COM-порт «слабенький».
Я не встречал COM-портов выдающих сигнал менее 5в.
Вот, в общем, и все.
Далее, установите программу, обслуживающую программатор ***PonyProg***, подключите программатор на свободный COM и запустите программу.
Все установки в программе - по умолчанию.
ВНИМАНИЕ: программатор не имеет выключателя питания, поэтому, перед установкой/выемкой контроллера из панели, необходимо отключать питание программатора.
Я делаю это, вытаскивая разъем питания. Хотя, были грехи, забывал это делать.
Пока проносило, но будьте осторожны, береженого как говорится…
Надеюсь, что моя нехитрая схемка и простенькая платка кому-то пригодятся.
P.S.
Специально гонял программатор в разных режимах, никаких сбоев не замечено. Работает безукоризненно. Все, что программатор пишет в контроллер, то же и читает.

Станислав Гетманский г. Самара

Переход с PonyProg на ICProg

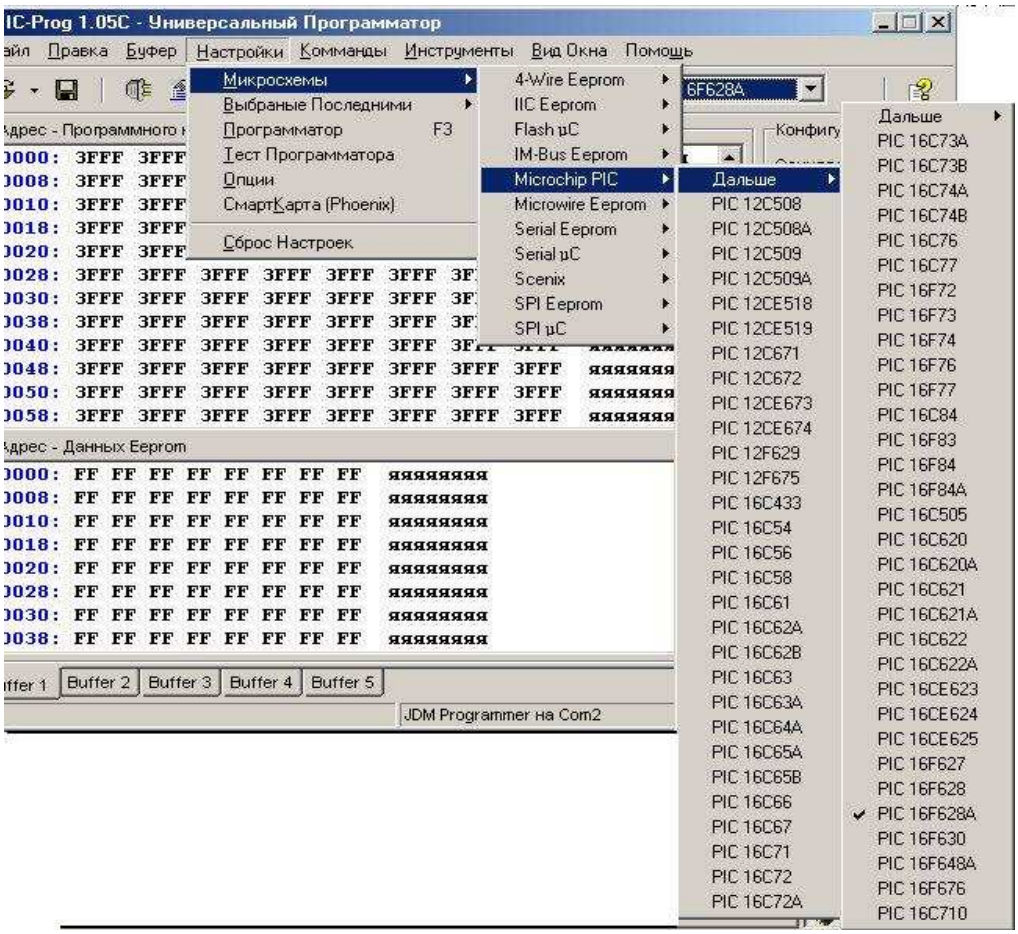
Используя аппаратную часть программатора ***PonyProg***, можно "состыковать" ее с более "навороченной" программой, которая называется ***ICProg105***.
Это очень удобно, так как используя одну и ту же аппаратную часть программатора, можно работать в одной из двух программ (***PonyProg*** и ***ICProg105***).
О том, как это сделать, рассказывает **Сосновский Александр**, г. Бердянск (статья автора).
Наверняка эта статья окажется полезной для многих людей.

Надеюсь, что Вы по достоинству оцените простое и эффективное решение довольно-таки непростой задачи.

Программатор *ICProg 105c-a*. Описание основных функций и возможностей.

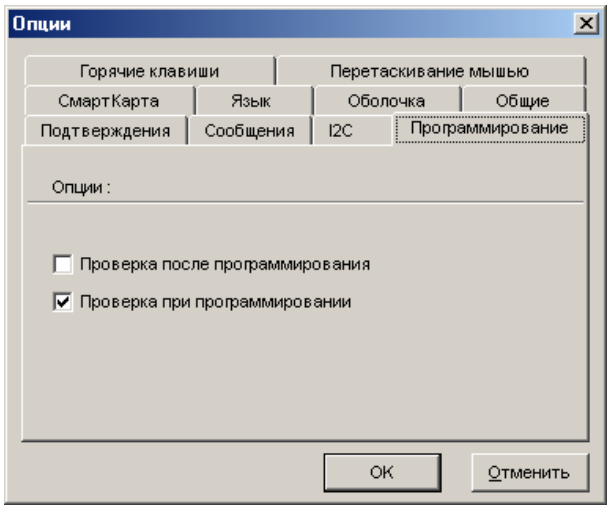
Для начала нужно сказать, что данный программатор это одна из последних версий которую можно скачать на сайте разработчика **www.ic-prog.com**.
Этот программатор не требует инсталляции, достаточно распаковать архив, и программа готова к работе.
При своей простоте интерфейса и минимальном размере дистрибутива, (примерно 2,7МБ) этот программатор заметно отличается своими возможностями и эргономичностью от других программаторов подобного типа.
В этой статье я остановлюсь только на основных функциях и возможностях этой программы в отношении программирования PIC контроллеров **PIC16F84** и **PIC16F628, 628A, 628A-I/P**.
ICProg 105c-a имеет в своем составе очень большую базу поддерживаемых контроллеров, а так же **FLASH** и **EEPROM**, что немаловажно: с его помощью можно работать со смарт-картами (имеется помощник программирования смарт-карт).

На картинке показан интерфейс и список поддерживаемых устройств.

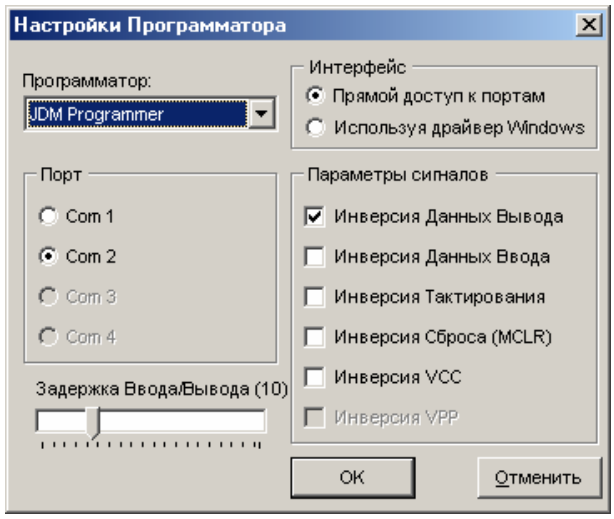


Как видно из картинки, ПИКов предостаточно.
Предполагается, что к компьютеру, через COM порт, подключена аппаратная часть программатора ***PonyProg***, информацию по изготовлению которой можно найти на сайте Корабельникова Евгения Александровича.
При использовании этой аппаратной части совместно с программой ***ICProg 105c-a***, никаких доработок ее принципиальной схемы не требуется.
После запуска программы, в большинстве случаев, автоматически происходит инициализация подключенного COM порта, т.е. тест можно не производить, хотя в меню (настройки) функция тестирования имеется.
По всей видимости, это сделано для нестандартных случаев.

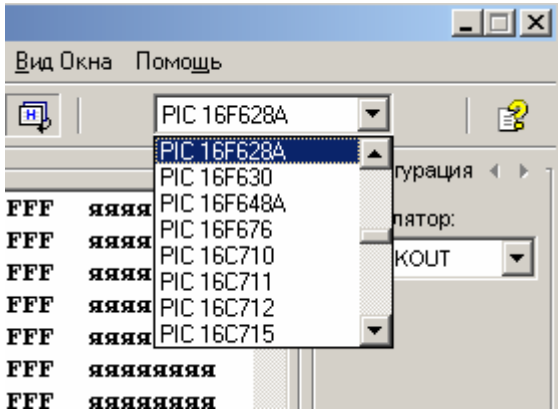
Кстати, программа ***ICProg 105c-a*** работает так же и с физическими программаторами, поддерживающими LPT порты.



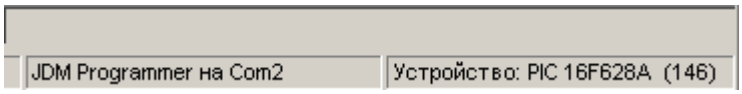
Далее, в меню ***настройки – опции,*** открываем вкладку ***программирование*** и ставим галочку в окошке ***проверка при программировании.*** После выбора этой опции, сообщение об ошибке будет выдаваться сразу же после ее возникновения, и не нужно будет дожидаться окончания полного цикла программирования и проверки данных, зашитых в ПИК, для того, чтобы получить это сообщение (в случае наличия ошибки).
Здесь же можно выставить и язык интерфейса.
Все остальное оставляем по умолчанию.
Далее заходим снова в меню ***настройки – программатор*** или просто жмем клавишу **F3** и попадаем в опции настройки физического программатора: в нашем случае это аппаратная часть программатора ***PonyProg***.



Здесь все должно быть выставлено, как показано на картинке. Программатор ***PonyProg*** здесь представляется как ***JDM Programmer***. Это что-то вроде универсального программатора, работающего с COM портом. Имеется большой выбор поддерживаемых программаторов, работающих, как было сказано выше, и с LPT портами.
Все остальное оставляем по умолчанию.
Хочу отметить, что, при использовании другого JDM программатора под COM порт, эти опции могут отличаться.
Например, в поле ***параметры сигналов***, нужно будет поставить галочку в поле ***Инверсия Данных Ввода***, а ***Инверсию Данных Вывода*** снять.



Далее, в окне выбора программируемых устройств, показанном на этом рисунке, выбираем PIC контроллер, который будет программироваться. Программируемый ПИК должен выбираться именно тот, который используется в действительности, т.е. если это **PIC16F628A** то именно его из перечня и выбираем, но никак не **PIC16F628**, иначе, при программировании, получим сообщение об ошибке типа “неизвестное устройство”.
Через меню ***файл***, открываем подготовленный HEX-файл.
После загрузки файла, в окне ***конфигурация***, можно наблюдать состояние битов конфигурации, которые были определены в “шапке” программы.
Здесь показаны все установленные биты конфигурации, а также и тип тактового генератора м/контроллера: в большинстве случаев это будет стандартный кварцевый генератор (XT).



В строке состояния будет указываться тип используемого программатора, номер COM – порта к которому он подключен, а также тип выбранного программируемого устройства.

После всех этих манипуляций, жмем кнопку  или клавишу **F5**, и ждем окончания процесса программирования.

Если при старте, сообщения об ошибке нет, то на практике, это на 99,9% означает то, что процесс программирования пройдет успешно.
Сказанное выше, справедливо для операционных систем **Windows 9x** и **Windows ME**.
С **Windows XP** дело обстоит немного иначе, о чем будет сказано ниже.
Необходимо особо отметить, что в ***ICProg 105xx*** имеется очень полезный для программистов встроенный дизассемблер, с помощью которого можно преобразовать "прошивку" (файл с расширением **.HEX**) в исходный ассемблерный код (файл с расширением **.ASM**), а это предоставляет возможность детального разбирательства с восстановленным таким образом текстом программы.
Дизассемблирование происходит так: сначала стандартным образом открывается **HEX-файл**, после чего щелкаем по кнопке  и получаем **ASM-файл**.
Правда, для того чтобы в полной мере "расшифровать" текст ASM-файла, полученного таким образом, и понять алгоритм работы программы, нужно быть программистом и обладать определенными навыками работы, плюс изрядно потрудиться.
Что бы вернуться обратно к HEX-файлу, достаточно нажать кнопку  .
Так что, в этом отношении, все очень удобно и универсально.
Однако, на мой взгляд, есть у этой программы и некоторые недостатки.
К ним можно отнести:
- довольно маленькое окно просмотра загружаемого кода, что очень не удобно, особенно для тех, кто привык работать с ***PonyProg***.
- скоростью программирования данный программатор также уступает ***PonyProg***, вероятно, из–за большого количества предварительных и последующих проверок в процессе программирования. Проверенно на собственном опыте: машина у меня не слабенькая (1700 Atlon, мозгов 512M) и проигрыш в скорости был заметен невооруженным глазом. Хотя, может это и лучше. Как говорят, "маслом каши не испортишь".

И последнее, на чем хотелось бы заострить внимание, это то, что до последней версии указанной в заголовке статьи, ***ICProg 105xx*** некорректно работал с операционной системой **Windows XP**.
В ***ICProg 105c-a*** все эти недостатки исправлены, хотя, по этому поводу, еще встречается много кривотолков при обсуждении данной темы на форумах.
Остается только отметить необходимые условия и настройки программы для работы с **Windows XP**, которые были описаны на форумах и проверены лично мной: у меня, все работало без проблем.
В первую очередь, для тех, кто работает с **XP**, нужно, с сайта разработчика, помимо самой ***ICProg105c-a***, скачать специальный драйвер (архив icprog – driver), который нужно распаковать в директорию, где находиться сама программа ***ICProg105c-a***.
После запуска программы, в меню ***настройки – опции***, на вкладке ***общие,*** устанавливаем опцию ***Вкл. /NT/2000/XP драйвер***.
Далее система спросит, установить драйвер или нет, естественно соглашаемся, и она его находит автоматом, т.к. он лежит там же где и сама прога.
В настройках программатора, т.е. в меню ***настройки – программатор*** (**F3**), оставляем все без изменений.
В заключение хотелось бы отметить, что, благодаря именно этому программатору, мне удалось прошить **PIC16F628A – I/P**.
Запрограммировать его в других программаторах, в том числе и в ***PonyProg***, было не возможно: при старте появлялось сообщение о неизвестном устройстве.
Если это сообщение игнорировать, то процесс программирования начинался, но в ПИК зашивались все нули.
Хотя в ***PonyProg 206*** и включена поддержка **PIC16F628**, но это не **PIC16F628A**, то есть, вероятно, существует явная разница между этими ПИКами.

Сосновский Александр Николаевич г. Бердянск

Работа программатора с операционной системой WindowsXP.

Информация от **Александра Волокитина** (статья автора).
Ознакомившись с материалами сайта http://ikarab.narod.ru решил изготовить программатор ***PonyProg***.
Схема привлекла своей простотой и доступностью программного обеспечения к ней.

Процесс изготовления не занял много времени.
Программатор выполнен в полном объеме, без сокращений.
В качестве ограничительных, использовались импортные стабилитроны номиналом 5,1 В.
В качестве средства инструментального контроля для изготовленного программатора, была использована программа ***TCOM***, скачанная с того же сайта.
Программатор ***PonyProg*** был подключен к компьютеру на порт COM1.
Компьютер на базе процессора AMD ATLON 64 3000+ с материнской платой ASUS K8V DELUXE.
Загружена операционная система **XP SP2**.
Выходной сигнал на COM1 исследовался на амплитуду размаха.
Данные выхода: от -10В до +10В.
При запуске программы выполнялись все тесты и калибровки без ошибок.
Попытки произвести запись данных в **PIC16F84A** заканчивались сообщениями ***Устройство неизвестно(-24)*** и ***Ошибка записи***.
Изменения в настройках программы ни к чему не привели.
Микросхема контроллера, ранее записанная на другом программаторе, также не читалась.
Попытки поднять уровень напряжения программирования до 5,5 В подключением диода в цепь общего провода микросхемы 7805 и увеличением тока стабилизации ограничительных стабилитронов, ни чему не привели.
Так как аппаратная часть программатора была проверена и перепроверена многократно, то оставалось грешить только на саму программу.
На сайте http://ikarab.narod.ru была выложена статья **Сосновского А. Н.** ***Программатор ICProg105c-a. Описание основных функций и возможностей***, где довольно подробно рассказано о программе ***ICProg105c***.
Установка этой программы не вызвало никаких трудностей.
Обратившись на сайт разработчика www.ic-prog.com, я скачал драйвер под версию **XP** и подключил его.
Перед программированием установил задержку ввода/вывода 20 единиц (кнопка **F3** ***Настройка программатора***).
Аппаратная часть изготовленного программатора не изменилась.
Результат порадовал.
Программирование **PIC16F84A** проходит устойчиво, без ошибок, как и вся работа программы в целом.
Проверялась программа во всех режимах и при разных выставленных задержках ввода/вывода данных.
Наиболее устойчивая работа программы наблюдалась при выставленных задержках не менее 20 единиц.
Программировались микросхемы **PIC16F84A-04I/P** и **PIC16F84A-20I/P**.
На сайте разработчика скачал обновленную версию ***ICProg105d***, которая адаптирована под **XP**. Таким образом, опытным путем было определено, что на компьютерах под ОС **XP**, программа ***ICProg105d*** работает наиболее устойчиво и позволяет достичь желаемых результатов ,без каких либо изменений в аппаратной части программатора.

Александр Волокитин

Архивный файл программы ***IcProg105d*** **(icprog105D.zip)** и драйвер под XP прилагаются **(папка *Программы* → папка *IcProg105*)**.

Откройте программу ***IcProg105d*** и в ***Опциях*** выберите русский язык.
Затем → перезагрузка компьютера.
Затем щелкните ***Настройки*** → ***Программатор***.
Откроется окно ***Настройки программатора***.
Выберите ***JDM Programmer*** (он установлен по умолчанию) и включите тот COM порт, к которому подключена аппаратная часть программатора ***PonyProg*****.**
В нижнем левом углу этого окна Вы увидите ползунок, которым выставляются задержки ввода/вывода.
Остальные настройки → по умолчанию.

Еще одна конструкция программатора.

Информация от **Васильева Геннадия** из Израиля.
Евгений Александрович.
Отправляю на Ваш суд схему программатора PIC.
Принципиально она ничем не отличается от Вашей схемы, правда я её сделал только для **PIC16Fxx**.
Компоненты, естественно, использовал буржуйские (КЕА: это не беда. Их можно заменить на отечественные).
Для улучшения помехозащищённости, ввёл несколько мелких конденсаторов (это всегда полезно) и резисторы **R3**, **R6** (они способствуют лучшему запиранию транзисторов **Q2**, **Q3**).
НО, САМОЕ ГЛАВНОЕ, Я ИЗМЕНИЛ ТОПОЛОГИЮ СХЕМЫ.
Если рисовать схему, отталкиваясь от реального расположения выводов на железе и проводить все линии питания и заземления (в смысле GND), то схема получается запутанной и загромождённой массой второстепенных линий и нужно долго водить пальцем, чтобы понять что, куда и зачем идёт.
Я думаю, что при рисовании схем, функциональный подход более нагляден.
То есть, законченные блочки (триггеры, мультивибраторы и пр.) рисуются в классическом виде и располагаются по ходу распространения сигналов, причём, слева направо.
Цепи питания и общие шины не рисуются, а только обозначаются.
Сигнальные линии проводятся с минимальным числом пересечений и изгибов, чтобы не рябило в глазах.
И чисто визуально, схема становится проще и понятнее и никуда не нужно водить пальцем.
Именно так я и постарался нарисовать предлагаемую схему.
Может быть это кому-нибудь и пригодится.
Конечно, всё это не очень принципиально, но я думаю, что такой подход сильно упрощает понимание схем и дальнейшую работу с ними.

Принципиальная схема программатора (**Pic16f.gif**) находится:
папка *Программы* → **папка *PonyProg***

О верификации

После открытия HEX-файла и **до** "прошивки" ПИКа, не нужно пытаться произвести так называемую **верификацию данных**, то есть, проверку записанных в ПИК данных, на предмет соответствия (или нет) "материнскому эталону", ведь если ничего не записано, то и проверять нечего.
Естественно, что после такой попытки, Вы получите сообщение об ошибочной верификации.
Речь идет о содержимом пункта меню программы ***PonyProg*** с названием ***Команды***.
Это строки выпадающего списка ***Проверить всё***, ***Проверить программу (FLASH)*** и ***Проверить данные (EEPROM)***.
Всё это относятся к процессу верификации данных, и щёлкать по этим строкам имеет смысл только после записи данных в ПИК.
Мало того, практического смысла в их применении маловато, так как в конце процедуры записи, всегда производится верификация.
То есть, наличие сообщения о безошибочной записи означает и то, что верификация завершилась успешно (проверяется и ***FLACH***, и ***EEPROM***).
Можно не щёлкать по строкам ***Программирование*** и ***Настройка программирования***, так как те же самые действия можно произвести и без "ухода" в **Команды**, хотя, может быть, кому-то такой способ работы и покажется удобным.
Лично я, редко когда пользуюсь пунктом меню ***Команды***. Удобнее щёлкать по пиктограммам.
В ***IcProg105***, верификация производится "оптом" (***Команды*** → ***Сравнить с буфером***).

Пример выполнения операции "чтение-модификация-запись" при работе с м/схемами памяти последовательного типа.

Используемый программатор: ***PonyProg*** любой версии.
"Привяжусь" к русифицированной версии ***2.05a***.
Используемый тип м/схемы памяти: **24C64** (64 Кбит или 64:8=8 Кбайт) фирмы **ATMEL**.

1. Подключаю аппаратную часть программатора к COM порту: в данном случае, к COM2 (а можно и к другому свободному), включаю компьютер, и после окончания его загрузки, вставляю **24C64** в панельку соответствующего, сменного модуля.
   Включаю питание программатора (а можно сделать это и позже).
2. Запускаю программу ***PonyProg***, на заставке щёлкаю по ***OK***, последовательно, 2 раза, в окнах предупреждений, щёлкаю по ***OK***, раскрываю окно программы на полный экран.
3. **24C64** имеет 2-х байтную адресацию (всего 16 битов), следовательно выставляю ***I2C Bus 16bit eeprom***.
4. Открываю список типов и выставляю ***2464/2465***.
5. Щёлкаю по надписи ***Установки***, а затем выбираю ***Настройку оборудования***.
6. В окне ***Настройка платы программатора***, выбираю ***Последовательный*** и ***COM2*** (остальные настройки → по умолчанию***: SI Prog API*** и все квадратики пустые).
7. Щёлкаю по кнопке ***Проверка*** и получаю сообщение ***Тест Ok***.
8. Последовательно щёлкаю по ***OK*** в окне ***Notice*** и в окне ***Настройка платы программатора***.
9. Еще раз щёлкаю по надписи ***Установки*** и выбираю ***Калибровка***.
10. В открывшемся окне ***Yes or No*** жму ***Yes*** и дожидаюсь окончания процесса калибровки.
11. В окне ***Notice***, щёлкаю по ***OK***. Подготовка завершена. Можно читать.
12. Навожу указатель мыши на пиктограмму с всплывающей подсказкой ***Считать устройство*** и щелкаю по ней (а можно щёлкнуть ***Команды*** → ***Считать всё***, без разницы).
13. Дожидаюсь окончания процесса считывания (появления окна ***Notice*** с сообщением ***Считывание завершено. Размер памяти 8192 байта***).
    Примечание: 1 Кбайт = 1024 байта, а не 1000 байтов.
14. В окне ***Notice***, щёлкаю по ***OK***.
    Примечание: что делать со считанной информацией, каждый решает для себя. Обычно, она, после ее конвертации в удобный формат (а можно и без этого), анализируется на предмет того, "а что бы такое изменить для того, чтобы получить желаемое?"
    После того, как найдены байты, которые нужно соответствующим образом изменить, начинается процедура их изменения.
    Например, в первом байте записано число **19h** (в окне ***PonyProg***, все числа представлены в 16-ричной системе исчисления, признака этой системы исчисления нет, то есть, число **19h** отображается как **19**).
15. Редактирование производится в буфере. В начале редактирования, нужно щёлкнуть по кнопке ***Правка***, а затем по строке ***Редактирование буфера***.
16. Навожу указатель мыши на числовое значение байта, содержимое которого нужно изменить (в данном случае, на 1-й байт) и 1 раз нажимаю левую кнопку мыши.
    Открывается окно ***Редактировать буфер***, в котором, в 16- и 10-ричной системах исчисления, показывается значение выбранного для редактирования байта, а также его символьное значение в международной кодировке.
17. Предположим, нужно изменить **19h** на **10h**.
    Ставлю мышкой курсор справа от цифры **9** (в строке ***Шестн.***), убираю ее и на ее месте, "настукиваю" **0**.
    Щёлкаю по ***OK***. Контролирую изменение значения 1-го байта: число **19h** заменилось на **10h**. Если в этом есть необходимость, то можно еще что-нибудь поменять.
18. Итак, модификация произведена. Можно записывать.
    Навожу указатель мыши на пиктограмму с всплывающей подсказкой ***Записать устройство*** и щёлкаю по ней.
19. В окне ***Yes or No***, щёлкаю по ***Yes*** и дожидаюсь окончания процедуры записи.
20. По окончании записи, появляется окно ***Notice*** с сообщением ***Запись завершена*** (критерий успешной, безошибочной записи). В окне ***Notice***, щёлкаю по ***OK***.
21. Теперь нужно (желательно, но не обязательно) убедиться в том, что считается то, что записалось (контрольное считывание).
    Произвожу это считывание щелчком по пиктограмме ***Считать устройство***.
22. Открывается окно ***Yes or No*** с вопросом ***Сохранить буфер перед чтением?***.
    Если ***Yes***, то далее следует стандартная процедура сохранения файла, а если ***No***, то он

не сохраняется (это кому как нравится).
Например, щёлкаю по ***No***. "Запускается" процедура чтения.

23. Дожидаюсь окончания процедуры чтения, в окне ***Notice***, щёлкаю по ***OK*** и убеждаюсь, что в первом байте считалось число **10h**.

Если нужно сохранить файл этого контрольного считывания (или другие, в том числе и "черновые" файлы), то это делается стандартно:
Файл → ***Сохранить файл с данными как ...***.

Примечание: если микросхема памяти последовательного типа "не секретная" (например, куплена в магазине, работала в составе какого-то "не секретного" устройства и т.д.), то обычно, никаких сложностей при работе с ней нет (естественно, если м/схема исправна). Если используется м/схема, снятая с какого-нибудь "секретного" устройства (разработчики не заинтересованы в том, чтобы кто-то "в ней хозяйничал"), то в часть ячеек ее памяти (или во все ячейки) невозможно будет произвести запись (можно только считать).

PIC16F84A ОБЛАСТЬ ОПЕРАТИВНОЙ ПАМЯТИ

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	INDF	TMR0	PCL	STATUS	FSR	PORTA	PORTB		EEDATA	EEADR	PCLATH	INTCON				
0010																
0020																
0080	INDF	OPTION	PCL	STATUS	FSR	TRISA	TRISB		EECON1	EECON2	PCLATH	INTCON				
0090																
00A0																

Пояснение

В этой распечатке отображены не всё регистры общего назначения (пояснения - на **стр.14** "Самоучителя…"). Полностью, область оперативной памяти можно увидеть в **MPLAB** (в окне оперпамяти. В опциях, должен быть выставлен **PIC16F84A**).

Приложение №3

СОСТАВ РЕГИСТРОВ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ PIC16F84A

Адрес	Регистр	Бит	Название	Назначение
03h/83h Банк0,1	STATUS	0	C	Флаг переноса-заема (работа с байтом).
		1	DC	Флаг переноса-заема (работа с младшим полубайтом).
		2	Z	Флаг нулевого результата.
		3	-PD	Флаг включения питания.
		4	-TO	Флаг переполнения сторожевого таймера WDT.
		5	RP0	Биты выбора банка
		6	RP1	в пределах группы банков.
		7	IRP	Бит выбора групп банков (0,1 и 2,3).
81h Банк1	OPTION_R	0	PS0	Биты установки
		1	PS1	коэффициента деления
		2	PS2	предделителя.
		3	PSA	Бит выбора способа включения предделителя.
		4	TOSE	Бит выбора способа приращения TMR0 при внешнем такте.
		5	TOCS	Бит выбора способа подачи такта.
		6	INTEDG	Бит выбора активного фронта сигнала на входе прерывания INT.
		7	-RBPU	Бит вкл-выкл подтягивающих R выводов PORTB
0Bh/8Bh Банк0,1	INTCON	0	RBIF	Флаг прерываний по изменению уровней сигналов на выводах RB4...RB7 PORTB.
		1	INTF	Флаг прерываний по входу INT.
		2	T0IF	Флаг прерываний по переполнению TMR0
		3	RBIE	Бит разрешения прерываний по изменению уровней сигналов на выводах RB4...RB7 PORTB.
		4	INTE	Бит разрешения прерываний по входу INT.
		5	TOIE	Бит разрешения прерываний по переполнению TMR0.
		6	EEIE	Бит разрешения прерываний по окончанию EEPROM записи.
		7	GIE	Бит глобального разрешения прерываний.
88h Банк1	EECON1	0	RD	Бит инициализации чтения из EEPROM.
		1	WR	Бит инициализации записи в EEPROM.
		2	WREN	Бит разрешения записи в EEPROM.
		3	WRERR	Флаг ошибки записи в EEPROM.
		4	EEIF	Флаг прерывания по окончанию записи в EEPROM
		5		
		6		
		7		
05h Банк0	PORTA	0	RA0	Двунаправленный вывод порта А с уровнями ТТЛ
		1	RA1	Двунаправленный вывод порта А с уровнями ТТЛ
		2	RA2	Двунаправленный вывод порта А с уровнями ТТЛ
		3	RA3	Двунаправленный вывод порта А с уровнями ТТЛ
		4	RA4/TOCKI	При работе на выход – защелка с открытым стоком При работе на вход – триггер Шмидта. TOCKI – вход внешнего такта для TMR0.
		5		
		6		
		7		
06h Банк0	PORTB	0	RB0/INT	Двунаправленный вывод порта В с уровнями ТТЛ или внешний вход прерывания INT.
		1	RB1	Двунаправленный вывод порта В с уровнями ТТЛ
		2	RB2	Двунаправленный вывод порта В с уровнями ТТЛ
		3	RB3	Двунаправленный вывод порта В с уровнями ТТЛ
		4	RB4	Двунаправленный вывод порта В с уровнями ТТЛ
		5	RB5	Двунаправленный вывод порта В с уровнями ТТЛ
		6	RB6	Двунаправленный вывод порта В с уровнями ТТЛ
		7	RB7	Двунаправленный вывод порта В с уровнями ТТЛ

2007h	Биты конфигурации	0	FOSC0	Биты выбора типа генератора.
		1	FOSC1	
		2	WDTE	Бит разрешения работы сторожевого таймера WDT
		3	-PWRTE	Бит разрешения работы таймера включения питания PWRT.
		4	CP	Бит защиты.
		5-13		

		7	6	5	4	3	2	1	0
03h(83h)	STATUS	IRP	RP1	RP0	-TO	-PD	Z	DC	C
05h	PORTA				RA4/TOCKI	RA3	RA2	RA1	RA0
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
0Bh(8Bh)	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81h	OPTION_REG	-RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
88h	EECON1				EEIF	WRERR	WREN	WR	RD
2007h	Конфигур.	Биты с 5 по 13			CP	-PWRTE	WDTE	FOSC1	FOSC0

00h/80h	INDF	Регистры косвенной адресации. Обращение к регистру INDF вызывает действие с регистром, адрес которого указан в FSR.
04h/84h	FSR	
02h/82h	PCL	Регистры счетчика программ PC. В PCL, младшие 8 бит. В PCH, старшие 5 бит.
0Ah/8Ah	PCLATH	
08h	EEDATA	Регистр хранения данных при чтении и записи. Регистр хранения адреса ячейки при чтении и записи.
09h	EEADR	
85h	TRISA	Выбор направления работы выводов порта A. Выбор направления работы выводов порта B.
86h	TRISB	
01h	TMR0	Регистр хранения байта для таймера TMR0.
89h	EECON2	Используется в операциях записи в EEPROM для реализации обязательной последовательности команд.

Приложение №4

Регистр OPTION_REG

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

Бит 7	-RBPU	Включение подтягивающих резисторов порта B: 1 – подтягивающие резисторы отключены 0 – подтягивающие резисторы включены
Бит 6	INTEDG	Выбор активного фронта сигнала на входе внешнего прерывания INT: 1 – прерывания по переднему фронту сигнала (0/1) 0 – прерывания по заднему фронту сигнала (1/0)
Бит 5	T0CS	Выбор такта для TMR0: 1 – внешний такт с вывода RA4/T0CKI 0 – внутренний такт CLKOUT
Бит 4	T0SE	Выбор фронта приращения TMR0 при внешнем такте: 1 – приращение при перепаде (на выводе RA4/T0CKI) от 1 к 0 0 – приращение при перепаде (на выводе RA4/T0CKI) от 0 к 1
Бит 3	PSA	Выбор способа включения предделителя: 1 – предделитель включен после сторожевого таймера WDT 0 – предделитель включен перед TMR0
Бит 2 - 0	PS2 PS1 PS0	Установка коэффициента деления предделителя

КОЭФФИЦИЕНТЫ ДЕЛЕНИЯ ПРЕДДЕЛИТЕЛЯ

Значение	Для TMR0	Для WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Регистр STATUS

IRP	RP1	RP0	-TO	-PD	Z	DC	C
7	6	5	4	3	2	1	0

Бит 7	IRP	Бит выбора групп банков. 0 – банк 0,1 (000h – 0FFh) 1 – банк 2,3 (100h – 1FFh)
Бит 6-5	RP1:RP0	Биты выбора банка в пределах группы банков. банк 3 (180h – 1FFh) банк 2 (100h – 17Fh) банк 1 (080h – 0FFh) банк 0 (000h – 07Fh)
Бит 4	-TO	Флаг переполнения сторожевого таймера WDT: 1 – после сброса по включению питания (POR) или выполнения команд CLRWDT, SLEEP 0 – после переполнения WDT
Бит 3	-PD	Флаг включения питания: 1 – после сброса по включению питания (POR) или выполнения команды CLRWDT 0 – после выполнения команды SLEEP
Бит 2	Z	Флаг нулевого результата: 1 – нулевой результат выполнения операции 0 – результат выполнения операции отличный от нуля
Бит 1	DC	Флаг переноса-заема (работа с младшим полубайтом) (для команд ADDWF, ADDLW, SUBWF, SUBLW): 1 – был перенос из младшего полубайта 0 – не было переноса из младшего полубайта
Бит 0	C	Флаг переноса-заема (работа с байтом) (для команд ADDWF, ADDLW, SUBWF, SUBLW): 1 – был перенос из байта 0 – не было переноса из байта

Примечания: 1) **При заеме, состояние флага имеет инверсное значение.**
2) **При выполнении команд сдвига, бит C загружается старшим или младшим битом сдвигаемого регистра.**
3) При помощи флагов **–TO** и **–PD**, можно определить причину сброса микроконтроллера.

Регистр PORTA

			RA4/TOCKI	RA3	RA2	RA1	RA0
7	6	5	4	3	2	1	0

Регистр PORTB

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
7	6	5	4	3	2	1	0

При работе вывода **на выход**, в соответствующий бит регистра **TRIS…** нужно записать **0**.
При работе вывода **на вход**, в соответствующий бит регистра **TRIS…** нужно записать **1**.
При этом, от вывода порта, отключается выход соответствующей защелки.

Регистр INTCON

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
7	6	5	4	3	2	1	0

Бит 7	GIE	Гобальное разрешение прерываний: 1 – разрешены все немаскированные прерывания 0 – все прерывания запрещены
Бит 6	EEIE	Разрешение прерываний по окончанию записи в EEPROM: 1 – прерывание разрешено 0 – прерывание запрещено
Бит 5	TOIE	Разрешение прерывания по переполнению TMR0: 1 – прерывание разрешено 0 – прерывание запрещено
Бит 4	INTE	Разрешение прерывания по входу RB0/INT: 1 – прерывание разрешено 0 – прерывание запрещено
Бит 3	RBIE	Разрешение прерывания по изменению уровней сигналов на выводах RB4...RB7: 1 – прерывание разрешено 0 – прерывание запрещено
Бит 2	T0IF	Флаг прерывания по переполнению TMR0 (сбрасывается программно): 1 – произошло переполнение TMR0 0 – переполнения TMR0 не было
Бит 1	INTF	Флаг прерывания по входу RB0/INT (сбрасывается программно): 1 – произошло прерывание по входу RB0/INT 0 – прерывания по входу RB0/INT не было
Бит 0	RBIF	Флаг прерывания по изменению уровней сигналов на выводах RB4...RB7 (сбрасывается программно): 1 – зафиксировано изменение уровня сигнала на одном из входов RB4...RB7 0 – не было изменения уровня сигнала ни на одном из входов RB4...RB7

СИМВОЛЫ КОНФИГУРАЦИИ

Состав символов можно узнать из файла .INC

Назначение	Символ	Назначение	Символ
Тактовый генератор	_RC_OSC	Сброс по снижению U пит.	_BODEN_ON
	_EXTRC_OSC		_BODEN_OFF
	_EXTRC_OSC_ CLKOUT	Режим работы вывода –MCLR	_MLCRE_ON
	_EXTRC_OSC_NOCLKOUT		_MLRCE_OFF
	_INTRC_OSC	Защита памяти программ	_CP_ALL
	_INTRC_OSC_CLKOUT		_CP_ON
	_INTRC_OSC_NOCLKOUT		_CP_75
	_LP_OSC		_CP_50
	_XT_OSC		_CP_OFF
	_HS_OSC	Защита EEPROM памяти данных	_DP_ON
Сторожевой таймер WDT	_WDT_ON		_DP_OFF
	_WDT_OFF	Защита калибровочной информации	_CPC_ON
Таймер вкл –я питания PWRT	_PWRTE_ON		_CPC_OFF
	_PWRTE_OFF		

Приложение №7

Регистр EECON1

7:5 – читаются как 00h

			EEIF	WRERR	WREN	WR	RD
7	6	5	4	3	2	1	0

Бит4	**EEIF**	**Флаг прерывания по окончанию записи в EEPROM (сбрасывается программно):** **1 – запись данных в EEPROM завершена** **0 – запись данных в EEPROM не завершена или не была начата**
Бит 3	**WRERR**	**Флаг ошибки записи в EEPROM:** **1 – запись прервана** **(произошел один из сбросов: по сигналу MLCR, по переполнению WDT в нормальном режиме, по снижению U питания BOR)** **0 – запись завершена**
Бит 2	**WREN**	**Разрешение записи в EEPROM:** **1 – запись разрешена** **0 – запись запрещена**
Бит 1	**WR**	**Инициализизация записи в EEPROM (программно может быть установлен только в 1, сбрасывается аппаратно):** **1 – инициализация записи** **0 – запись завершена**
Бит 0	**RD**	**Инициализизация чтения из EEPROM (программно может быть установлен только в 1, сбрасывается аппаратно):** **1 – инициализация чтения** **0 – чтение завершено**

Приложение №8

Биты КОНФИГУРАЦИИ

Адрес 2007h

13	12	11	10	9	8	7	6	5	CP	-PWRTE	WDTE	FOSC1	FOSC0
Не задействованы (читаются как 1)									4	3	2	1	0

Бит 4	**CP**	**Бит защиты:** **1 – защита выключена** **0 – защита включена**
Бит 3	**-PWRTE**	**Бит разрешения работы таймера включения питания PWRT:** **0 – будет производиться выдержка при включении питания** **1 – выдержки производиться не будет**
Бит 2	**WDTE**	**Бит разрешения работы сторожевого таймера WDT:** **1 – WDT включен** **0 – WDT выключен**
Бит 1,0	**FOSC1** **FOSC0**	**Биты выбора типа генератора**

Режимы тактового генератора

00	**LP - генератор**	**НЧ генератор для экономичных приложений**
01	**XT - генератор**	**Стандартный кварцевый генератор**
10	**HS - генератор**	**ВЧ кварцевый генератор**
11	**RC - генератор**	**RC генератор с внешней RC цепью**

Напряжение питания	Тип генератора	Макс. рабочая частота
2 – 3 v	RC LP	2 Мгц 200 Кгц
3 – 6 v	RC, XT LP	4 Мгц 200 Кгц
4,5 – 5,5 v	HS	10 Мгц

***Биты конфигурации задаются с помощью директивы* CONFIG.**

ПРИМЕР:

```
LIST         p=p16C77      ; Выбор типа м/контроллера.
#INCLUDE     <P16C77.INC> ; Подключение вспомогательного файла.

;Настройка битов конфигурации

 __CONFIG _XT_OSC & _PWRTE_ON & _CP_OFF &_WDT_ON
 ............................................
 ............................................

 org          0            ; Установка нулевого адреса в памяти программ.
 goto         START        ; Начало исполнения программы.
 . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . .
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  end
```

Набор символов, доступных для конкретного микроконтроллера, находится в соответствующем файле .INC

Приложение №9

Таблица чисел, отображаемых в одном байте

D	B	H	D	B	H	D	B	H	D	B	H
0	0000 0000	00	64	0100 0000	40	128	1000 0000	80	192	1100 0000	C0
1	0000 0001	01	65	0100 0001	41	129	1000 0001	81	193	1100 0001	C1
2	0000 0010	02	66	0100 0010	42	130	1000 0010	82	194	1100 0010	C2
3	0000 0011	03	67	0100 0011	43	131	1000 0011	83	195	1100 0011	C3
4	0000 0100	04	68	0100 0100	44	132	1000 0100	84	196	1100 0100	C4
5	0000 0101	05	69	0100 0101	45	133	1000 0101	85	197	1100 0101	C5
6	0000 0110	06	70	0100 0110	46	134	1000 0110	86	198	1100 0110	C6
7	0000 0111	07	71	0100 0111	47	135	1000 0111	87	199	1100 0111	C7
8	0000 1000	08	72	0100 1000	48	136	1000 1000	88	200	1100 1000	C8
9	0000 1001	09	73	0100 1001	49	137	1000 1001	89	201	1100 1001	C9
10	0000 1010	0A	74	0100 1010	4A	138	1000 1010	8A	202	1100 1010	CA
11	0000 1011	0B	75	0100 1011	4B	139	1000 1011	8B	203	1100 1011	CB
12	0000 1100	0C	76	0100 1100	4C	140	1000 1100	8C	204	1100 1100	CC
13	0000 1101	0D	77	0100 1101	4D	141	1000 1101	8D	205	1100 1101	CD
14	0000 1110	0E	78	0100 1110	4E	142	1000 1110	8E	206	1100 1110	CE
15	0000 1111	0F	79	0100 1111	4F	143	1000 1111	8F	207	1100 1111	CF
16	0001 0000	10	80	0101 0000	50	144	1001 0000	90	208	1101 0000	D0
17	0001 0001	11	81	0101 0001	51	145	1001 0001	91	209	1101 0001	D1
18	0001 0010	12	82	0101 0010	52	146	1001 0010	92	210	1101 0010	D2
19	0001 0011	13	83	0101 0011	53	147	1001 0011	93	211	1101 0011	D3
20	0001 0100	14	84	0101 0100	54	148	1001 0100	94	212	1101 0100	D4
21	0001 0101	15	85	0101 0101	55	149	1001 0101	95	213	1101 0101	D5
22	0001 0110	16	86	0101 0110	56	150	1001 0110	96	214	1101 0110	D6
23	0001 0111	17	87	0101 0111	57	151	1001 0111	97	215	1101 0111	D7
24	0001 1000	18	88	0101 1000	58	152	1001 1000	98	216	1101 1000	D8
25	0001 1001	19	89	0101 1001	59	153	1001 1001	99	217	1101 1001	D9
26	0001 1010	1A	90	0101 1010	5A	154	1001 1010	9A	218	1101 1010	DA
27	0001 1011	1B	91	0101 1011	5B	155	1001 1011	9B	219	1101 1011	DB
28	0001 1100	1C	92	0101 1100	5C	156	1001 1100	9C	220	1101 1100	DC
29	0001 1101	1D	93	0101 1101	5D	157	1001 1101	9D	221	1101 1101	DD
30	0001 1110	1E	94	0101 1110	5E	158	1001 1110	9E	222	1101 1110	DE
31	0001 1111	1F	95	0101 1111	5F	159	1001 1111	9F	223	1101 1111	DF
32	0010 0000	20	96	0110 0000	60	160	1010 0000	A0	224	1110 0000	E0
33	0010 0001	21	97	0110 0001	61	161	1010 0001	A1	225	1110 0001	E1
34	0010 0010	22	98	0110 0010	62	162	1010 0010	A2	226	1110 0010	E2
35	0010 0011	23	99	0110 0011	63	163	1010 0011	A3	227	1110 0011	E3
36	0010 0100	24	100	0110 0100	64	164	1010 0100	A4	228	1110 0100	E4
37	0010 0101	25	101	0110 0101	65	165	1010 0101	A5	229	1110 0101	E5
38	0010 0110	26	102	0110 0110	66	166	1010 0110	A6	230	1110 0110	E6
39	0010 0111	27	103	0110 0111	67	167	1010 0111	A7	231	1110 0111	E7
40	0010 1000	28	104	0110 1000	68	168	1010 1000	A8	232	1110 1000	E8
41	0010 1001	29	105	0110 1001	69	169	1010 1001	A9	233	1110 1001	E9
42	0010 1010	2A	106	0110 1010	6A	170	1010 1010	AA	234	1110 1010	EA
43	0010 1011	2B	107	0110 1011	6B	171	1010 1011	AB	235	1110 1011	EB
44	0010 1100	2C	108	0110 1100	6C	172	1010 1100	AC	236	1110 1100	EC
45	0010 1101	2D	109	0110 1101	6D	173	1010 1101	AD	237	1110 1101	ED
46	0010 1110	2E	110	0110 1110	6E	174	1010 1110	AE	238	1110 1110	EE
47	0010 1111	2F	111	0110 1111	6F	175	1010 1111	AF	239	1110 1111	EF
48	0011 0000	30	112	0111 0000	70	176	1011 0000	B0	240	1111 0000	F0
49	0011 0001	31	113	0111 0001	71	177	1011 0001	B1	241	1111 0001	F1
50	0011 0010	32	114	0111 0010	72	178	1011 0010	B2	242	1111 0010	F2
51	0011 0011	33	115	0111 0011	73	179	1011 0011	B3	243	1111 0011	F3
52	0011 0100	34	116	0111 0100	74	180	1011 0100	B4	244	1111 0100	F4
53	0011 0101	35	117	0111 0101	75	181	1011 0101	B5	245	1111 0101	F5
54	0011 0110	36	118	0111 0110	76	182	1011 0110	B6	246	1111 0110	F6
55	0011 0111	37	119	0111 0111	77	183	1011 0111	B7	247	1111 0111	F7
56	0011 1000	38	120	0111 1000	78	184	1011 1000	B8	248	1111 1000	F8
57	0011 1001	39	121	0111 1001	79	185	1011 1001	B9	249	1111 1001	F9
58	0011 1010	3A	122	0111 1010	7A	186	1011 1010	BA	250	1111 1010	FA
59	0011 1011	3B	123	0111 1011	7B	187	1011 1011	BB	251	1111 1011	FB
60	0011 1100	3C	124	0111 1100	7C	188	1011 1100	BC	252	1111 1100	FC
61	0011 1101	3D	125	0111 1101	7D	189	1011 1101	BD	253	1111 1101	FD
62	0011 1110	3E	126	0111 1110	7E	190	1011 1110	BE	254	1111 1110	FE
63	0011 1111	3F	127	0111 1111	7F	191	1011 1111	BF	255	1111 1111	FF

КОМАНДЫ АССЕМБЛЕРА ДЛЯ ПИКов (вариант с ошибками №1)

МНЕМОНИКА	ОПИСАНИЕ	ПРИМЕРЫ	Цикл.	Флаги
Байт-ориентированные команды				

ADDWF f, d
 	
Сложение W и f
	Сложить содержимое регистров W и f.
	
	Если d=0, результат сохраняется в регистре W.
	Если d=1, результат сохраняется в регистре f.
	Косвенная адресация: для ее выполнения необходимо обратиться к регистру INDF. Оно вызовет действие с регистром, адрес которого указан в FSR.
	Косвенная запись в регистр INDF не вызовет никаких действий (кроме воздействия на флаги в регистре STATUS)
	Косвенное чтение INDF (FSR=0) даст результат 00h.
	9-й бит косвенного адреса (IRP) сохраняется в регистре STATUS<7>.
	Изменение адреса счетчика команд PC (вычисляемый переход) выполняется командой приращения к регистру PCL. При этом необходимо следить, чтобы значение PCL не пересекало границу блока памяти данных (256 байт, иначе работа по кольцу).
	PCL – младший байт (8 бит <7:0>) счетчика команд (PC), доступен для чтения записи.
	PCH – старший байт (5 бит <12:8>) счетчика команд PC, не доступен для чтения и записи). Все операции с PCH происходят через дополнительный регистр PCLATH.
		ADDWF ABC,0
		До выполнения
		W=0x17
		ABC=0xC2
		После выполнения
		W=0xD9
		ABC=0xC2
		Косвенная адресация
		ADDWF INDF,1
		До выполнения
		W=0x17
		FSR=0xC2 (значение регистра с адресом в FSR=0x20)
		После выполнения
		W=0x17
		FSR=0xC2 (значение регистра с адресом в FSR=0x37)
		Изменение адреса PC
		ADDWF PCL,0
		До выполнения
		W=0x10
		PCL=0x37
		C=x
		После выполнения
		PCL=0x47
		C=0
		ADDWF PCL,0
		До выполнения
		W=0x10
		PCL=0xF7
		PCH=0x08
		C=x
		После выполнения
		PCL=0x07
		PCH=0x08
		C=1
			1	C,DC,Z

ANDWF f, d Побитное И W и f	Выполняется побитное “И” содержимого регистров W и f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	ANDWF ABC,1 До выполнения W=0x17 (00010111) ABC=0xC2(11000010) После выполнения W=0x17 ABC=0x02 (00000010) ANDWF ABC,0 До выполнения W=0x17 (00010111) ABC=0xC2(11000010) После выполнения W=0x02 (00000010) ABC=0xC2 *Косвенная адресация* ANDWF INDF,1 До выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x5A) После выполнения	1	Z

		W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x15)		
CLRF Очистить f	Очистить содержимое регистра f и установить флаг Z	CLRF FLAG_REG До выполнения FLAG_REG=0x5A После выполнения FLAG_REG=0x00 Z=1 *Косвенная адресация* CLRF INDF До выполнения FSR=0xC2 (значение регистра с адресом в FSR=0xAA) После выполнения FSR=0xC2 (значение регистра с адресом в FSR=0x00) Z=1	1	Z
CLRW Очистить W	Очистить содержимое регистра W и установить флаг Z	CLRW До выполнения W=0x5A После выполнения W=0x00 Z=1	1	Z
COMF f, d Инвертиро-вать f	Инвертировать все биты в регистре f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f	COMF REG1,0 До выполнения REG1=0x13 После выполнения REG1=0x13 W=0xEC COMF REG1,1 До выполнения REG1=0xFF После выполнения REG1=0x00 Z=1 *Косвенная адресация* COMF INDF,1 До выполнения FSR=0xC2 (значение регистра с адресом в FSR=0xAA) После выполнения FSR=0xC2 (значение регистра с адресом в FSR=0x55)	1	Z
DECF f, d Вычесть 1 из f	Декрементировать содержимое регистра f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	DECF CNT,1 До выполнения CNT=0x01 Z=0 После выполнения CNT=0x00 Z=1 DECF CNT,0 До выполнения CNT=0x10 W=x Z=0 После выполнения CNT=0x10 W=0x0F Z=0	1	Z

		Косвенная адресация DECF INDF,1 До выполнения FSR=0xC2 (значение регистра с адресом в FSR=0x01) После выполнения FSR=0xC2 (значение регистра с адресом в FSR=0x00) Z=1		
DECFSZ f, d Вычесть 1 из f и пропустить если 0	Декрементировать содержимое регистра f с пропуском если 0 Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. Если результат не равен 0, исполняется следующая инструкция. Если результат = 0, то следующая инструкция не исполняется, а команда выполняется за 2 цикла (во втором цикле выполняется NOP).	HERE DECFSZ CNT,1 GOTO LOOP CONTINUE . . 1) До выполнения CNT=0x01 PC=адрес HERE После выполнения CNT=0x00 PC=адрес CONTINUE 2) До выполнения CNT=0x02 PC=адрес HERE После выполнения CNT=0x01 PC=адрес HERE+1	1(2)	
INCF f, d Прибавить 1 к f	Инкрементировать содержимое регистра f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	1) INCF CNT,1 До выполнения CNT=0xFF Z=0 После выполнения CNT=0x00 Z=1 2) INCF CNT,0 До выполнения CNT=0x10 W=x Z=0 После выполнения CNT=0x10 W=0x11 Z=0 *Косвенная адресация* INCF INDF,1 До выполнения FSR=0xC2 (значение регистра с адресом в FSR=0xFF) Z=0 После выполнения FSR=0xC2 (значение регистра с адресом в FSR=0x00) Z=1	1	Z
INCFSZ f, d Прибавить 1 к f и пропустить если 0	Инкрементировать содержимое регистра f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. Если результат не равен 0, исполняется следующая инструкция. Если результат = 0, то следующая инструкция не исполняется, команда выполняется за 2 цикла (во втором цикле – NOP).	HERE INCFSZ CNT,1 GOTO LOOP CONTINUE . . 1) До выполнения PC=адрес HERE CNT=0xFF После выполнения CNT=0x00 PC=адрес CONTINUE 2) До выполнения PC=адрес HERE CNT=0x00 После выполнения CNT=0x01 PC=адрес HERE+1	1(2)	

IORWF f,d Побитное “ИЛИ” W и f	Побитное “ИЛИ” содержимого регистров W и f Если d=0 – результат сохраняется в регистре W Если d=1 – результат сохраняется в регистре f.	1) IORWF RES,0 До выполнения RES=0x13 W=0x91 После выполнения RES=0x13 W=0x93 Z=0 2) IORWF RES,1 До выполнения RES=0x13 W=0x91 После выполнения RES=0x93 W=0x91 Z=1 3) IORWF RES,1 До выполнения RES=0x00 W=0x00 После выполнения RES=0x00 W=0x00 Z=1 *Косвенная адресация* IORWF INDF,1 До выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x30) После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x37) Z=0	1	Z

MOVF f,d
 	
Переслать f
	Содержимое регистра f пересылается в регистр адресата
	
	Если d=0 – значение сохраняется в регистре W.
	Если d=1 – значение сохраняется в регистре f.
	
	d=1 используется для проверки содержимого f на ноль.
		MOVF FSR,0
		До выполнения
		W=0x00
		FSR=0xC2
		После выполнения
		W=0xC2
		FSR=0xC2
		Z=0
		MOVF FSR,1
		1) До выполнения
		FSR=0x43
		После выполнения
		FSR=0x43
		Z=0
		2) До выполнения
		FSR=0x00
		После выполнения
		FSR=0x00
		Z=1
		Косвенная адресация
		MOVF INDF,1
		До выполнения
		W=0x17
		FSR=0xC2 (значение регистра с адресом в FSR=0x00)
		После выполнения
		W=0x17
		FSR=0xC2 (значение регистра с адресом в FSR=0x00)
		Z=1
			1	Z

MOVWF f Переслать W в f	Переслать содержимое W в f	MOVWF OPTION До выполнения OPTION=0xFF W=0x4F После выполнения OPTION=0x4F W=0x4F *Косвенная адресация* MOVWF INDF До выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x00) После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x17)	1	
NOP	Нет операции	HERE NOP До выполнения PC=адрес HERE После выполнения PC=адрес HERE+1	1	
RLF f, d Циклический сдвиг f влево через перенос	Выполняется циклический сдвиг влево содержимого регистра f через бит C регистра STATUS Если d=0 – результат сохраняется в регистре W. Если d=1 – результат сохраняется в регистре f.  	RLF REG,0 До выполнения REG=11100110 C=0 После выполнения REG=11100110 W=11001100 C=1 *Косвенная адресация* RLF INDF,1 До выполнения W=xxxxxxxx FSR=0xC2 (значение регистра с адресом в FSR=0x3A-00111010) C=1 После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x75-01110101) C=0 RLF INDF,1 До выполнения W=xxxxxxxx FSR=0xC2 (значение регистра с адресом в FSR=0xB9-10111001) C=0 После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x72-01110010) C=1	1	C

RRF f, d Циклический сдвиг f вправо через перенос	Выполняется циклический сдвиг вправо содержимого регистра f через бит C регистра STATUS Если d=0 – результат сохраняется в регистре W. Если d=1 – результат сохраняется в регистре f.  	RRF REG,0 До выполнения REG=11100110 W=xxxxxxxx C=0 После выполнения REG=11100110 W=01110011 C=0 *Косвенная адресация* RRF INDF,1 До выполнения W=xxxxxxxx FSR=0xC2 (значение регистра с адресом в FSR=0x3A-00111010 C=1 После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x9D-10011101 C=0 RRF INDF,1 До выполнения W=xxxxxxxx FSR=0xC2 (значение регистра с адресом в FSR=0x39-00111001 C=0 После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x1C-00011100 C=1	1	C
SUBWF f, d Вычесть W из f	Вычесть содержимое регистра W из регистра f. Если d=0 – результат сохраняется в регистре W. Если d=1 – результат сохраняется в регистре f.	SUBWF REG,1 До выполнения REG=0x03 W=0x02 C=x Z=x После выполнения REG=0x01 W=0x02 C=1 (+ результат) Z=0 SUBWF REG,1 До выполнения REG=0x02 W=0x02 C=x Z=x После выполнения REG=0x00 W=0x02 C=1 (0 – результат) Z=1 SUBWF REG,1 До выполнения REG=0x01 W=0x02 C=x Z=x После выполнения REG=0xFF W=0x02 C=0 (- результат) Z=0	1	C,DC,Z

SWAPF f, d Поменять местами полубайты в f	Поменять местами старший и младший полубайты регистра f. Если d=0 – результат сохраняется в регистре W. Если d=1 – результат сохраняется в регистре f.	SWAPF REG,0 До выполнения REG=0xA5 W=x После выполнения REG=0xA5 W=0x5A SWAPF REG,1 До выполнения REG=0xA5 После выполнения REG=0x5A *Косвенная адресация* SWAPF INDF,1 До выполнения W=0x17 FSR=0xC2 (значение Регистра с адресом в FSR=0x20 После выполнения W=0x17 FSR=0xC2 (значение Регистра с адресом в FSR=0x02	1	
XORWF f, d Побитное “исключающее ИЛИ” W и f	*Сравнение содержимого регистров W и f (проверка на одинаковость)* Побитное “Исключающее “ИЛИ” содержимого регистров W и f. Если d=0 – результат сохраняется в регистре W. Если d=1 – результат сохраняется в регистре f.	XORWF REG,1 До выполнения REG=0xAF W=0xB5 После выполнения REG=0x1A W=0xB5 XORWF REG,0 До выполнения REG=0xAF W=0xB5 После выполнения REG=0xAF W=0x1A *Косвенная адресация* XORWF INDF,1 До выполнения W=0xB5 FSR=0xC2 (значение регистра с адресом в FSR=0xAF) После выполнения W=0xB5 FSR=0xC2 (значение регистра с адресом в FSR=0x1A)	1	Z
Бит - ориентированные команды (b-от 0 до 7)				
BCF f, b Установить в “0” бит в регистре f	Очистить бит b в регистре f	BCF REG,7 До выполнения REG=0xC7-11000111 После выполнения REG=0x47- 01000111 *Косвенная адресация* BCF INDF,3 До выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x2F) После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x27)	1	

BSF f, b Установить в “1” бит в регистре f	Установить бит b в регистре f	BSF REG,7 До выполнения REG=0x0A- 00001010 После выполнения REG=0x8A- 10001010 *Косвенная адресация* BSF INDF,3 До выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x20) После выполнения W=0x17 FSR=0xC2 (значение регистра с адресом в FSR=0x28)	1	
BTFSC f, b Проверить бит b в регистре f, если b=0, то пропустить следующую инструкцию	Если бит b в регистре f =1, исполняется следующая инструкция Если бит b в регистре f =0, то следующая инструкция не выполняется, а команда выполняется за 2 цикла (во 2-м цикле выполняется NOP)	HERE BTFSC FLAG,4 FALSE GOTO ABC TRUE . . 1) До выполнения PC=адрес HERE FLAG=xxx0xxxx После выполнения T.к. FLAG<4>=0, PC=адрес TRUE 2) До выполнения PC=адрес HERE FLAG=xxx1xxxx После выполнения T.к. FLAG<4>=1, PC=адрес FALSE	1(2)	
BTFSS f, b Проверить бит b в регистре f, если b=1, то пропустить следующую инструкцию	Если бит b в регистре f=0, исполняется следующая инструкция Если бит b в регистре f=1, то следующая инструкция не выполняется, а команда выполняется за 2 цикла (во 2-м цикле выполняется NOP)	HERE BTFSS FLAG,4 FALSE GOTO ABC . . 1) До выполнения PC=адрес HERE FLAG=xxx0xxxx После выполнения T.к. FLAG<4>=0, PC=адрес FALSE 2) До выполнения PC=адрес HERE FLAG=xxx1xxxx После выполнения T.к. FLAG<4>=1, PC=адрес TRUE	1(2)	
Команды операций с константами (k – от 0 до 255)				
ADDLW k Сложить константу с W	Содержимое регистра W складывается с 8 – разрядной константой k. Результат сохраняется в регистре W	ADDLW 0x15 До выполнения W=0x10 После выполнения W=0x25 ADDLW REG До выполнения W=0x10 REG=0x37(адрес регистра) После выполнения W=0x47	1	C,DC,Z

		ADDLW HIGH(LU_TAB) До выполнения W=0x10 LU_TAB=0x9375 (адрес в памяти программ) После выполнения W=0xA3		
SUBLW k Вычесть W из константы	Вычесть содержимое регистра W из 8 – разрядной константы k. Результат сохраняется в регистре W.	SUBLW 0x02 До выполнения W=0x01 C=? Z=? После выполнения W=0x01 C=1 (результат +) Z=0 SUBLW 0x02 До выполнения W=0x03 C=? Z=? После выполнения W=0xFF C=0 (результат -) Z=0 SUBLW 0x02 До выполнения W=0x02 C=? Z=? После выполнения W=0x00 C=1 (“0”- результат) Z=1 SUBLW REG До выполнения W=0x10 REG=0x37 (адрес регистра) После выполнения W=0x27 C=1	1	C,DC,Z
MOVLW k Переслать константу в W	Переслать константу k в регистр W В неиспользуемых битах ассемблер устанавливает 0	MOVLW 0x5A До выполнения W=0x10 (любое) После выполнения W=0x5A MOVLW REG До выполнения W=0x10 REG=0x37 (адрес регистра) После выполнения W=0x37 Z=0 MOVLW HIGH(LU_TAB) До выполнения W=0x10 LU_TAB=0x9375 (адрес в памяти программ) После выполнения W=0x93	1	

ANDLW k Побитное “И” константы и W	Выполняется побитное “И” содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W	ANDLW 0x5F (01011111) До выполнения W=0xA3 (10100011) После выполнения W=0x03 (00000011) ANDLW REG До выполнения W=0xA3 REG=0x37 (адрес регистра) После выполнения W=0x23 ANDLW HIGH(LU_TAB) До выполнения W=0xA3 LU_TAB=0x9375 (адрес в памяти программ) После выполнения W=0x83	1	Z
IORLW k Побитное “ИЛИ” константы и W	Выполняется побитное “ИЛИ” содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W.	IORLW 0x35 До выполнения W=0x9A После выполнения W=0xBF Z=0 IORLW REG До выполнения W=0x9A REG=0x37 (адрес регистра) После выполнения W=0x9F Z=0 IORLW HIGH(LU_TAB) До выполнения W=0x9A AB=0x9375 (адрес в памяти программ) После выполнения W=0x9B IORLW 0x00 До выполнения W=0x00 После выполнения W=0x00 Z=1	1	Z

XORLW k Побитное “Исключающее ИЛИ” константы и W	*Сравнение содержимого регистра W и константы (проверка на одинаковость)* Выполняется побитное “Исключающее ИЛИ” содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W.	XORLW 0xAF(10101111) До выполнения W=0xB5 (10110101) После выполнения W=0x1A (00011010) Z=0 XORLW REG До выполнения W=0xAF REG=0x37 (адрес регистра) После выполнения W=0x18 Z=0 XORLW HIGH(LU_TAB) До выполнения W=0xAF LU_TAB=0x9375 (адрес в памяти программ) После выполнения W=0x3C Z=0	1	Z
Команды управления				
CALL Вызов подпрограммы (переход по коду команды с условием (стек))	Вызов подпрограммы. Адрес следующей инструкции (PC+1) помещается в вершину стека (TOS). 11 бит адреса загружаются из кода команды в счетчик команд PC<10:0>. 2 старших бита загружаются в счетчик команд PC<12:11> из регистра PCLATH.	HERE CALL ABC До выполнения PC=адрес HERE После выполнения PC=адрес ABC TOS=адрес HERE+1	2	
GOTO k Переход по коду команды без условия	Выполнить безусловный переход. 11 бит адреса загружаются из кода команды в счетчик команд PC<10:0>. 2 старших бита загружаются в счетчик команд PC<12:11> из регистра PCLATH.	GOTO ABC После выполнения PC= адрес ABC	2	
RETURN Возврат из подпрограммы	Возврат из подпрограммы. Вершина стека (TOS) загружается в счетчик команд PC.	RETURN После выполнения PC=TOS(адрес из стека)	2	
RETLW k Возврат из подпрограм-мы с загрузкой константы в W	В регистр W загружается 8-разрядная константа. Вершина стека (TOS) загружается в счетчик команд PC.	CALL TABLE . . TABLE ADDWF PCL,1 RETLW k1 RETLW k2 . . RETLW kn До выполнения W=0x07 После выполнения W=значение k8 PC=TOS=адресHERE+1	2	

RETFIE Возврат из подпрограммы с разрешением прерываний	Возврат из подпрограммы обработки прерываний. Вершина стека (TOS) загружается в счетчик команд PC. Устанавливается в “1” флаг глобального разрешения прерываний GIE (INTCON<7>).	RETFIE После выполнения PC=TOS GIE=1	2	
CLRWDT Очистить WDT (сторожевой таймер)	Сбрасывает WDT и предделитель (если он подключен к WDT). В регистре STATUS устанавливает биты –TO и -PD. На коэффициент делителя WDT не влияет	CLRWDT До выполнения Счетчик WDT-? Счетчик делителя WDT=1:128 После выполнения Счетчик WDT=0 Счетчик делителя WDT=0 -TO=1 -PD=1 Делитель WDT=1:128	1	-TO -PD
SLEEP Перейти в режим SLEEP	Сбросить флаг включения питания -PD в “0”. Установить флаг переполнения WDT -TO в “1”. Очистить таймер WDT и его предделитель. Перевести микроконтроллер в режим SLEEP и выключить тактовый генератор.	SLEEP	1	-TO -PD
OPTION	Переслать содержимое регистра W в регистр OPTION. Инструкция поддерживается для совместимости программы с семейством PIC16C5x. Запись-чтение регистра OPTION можно выполнить прямой или косвенной адресацией. Не рекомендуется использовать для совместимости программного обеспечения с последующими выпускаемыми PIC16Cxx.	OPTION		
TRIS	Переслать содержимое регистра W в регистр TRIS. См. комментарии для команды OPTION	TRIS		

КОМАНДЫ АССЕМБЛЕРА ДЛЯ ПИКов (вариант с ошибками №2)

МНЕМОНИКА	ОПИСАНИЕ	ПРИМЕРЫ	Цикл	Флаги
Байт-ориентированные команды				

ADDWF f, d
 	
Сложение W и f
	Сложить содержимое регистров W и f.
	
	Если d=0, результат сохраняется в регистре W.
	Если d=1, результат сохраняется в регистре f.
	
	Косвенная адресация: для ее выполнения необходимо обратиться к регистру INDF. Оно вызовет действие с регистром, адрес которого указан в FSR.
	Косвенная запись в регистр INDF не вызовет никаких действий (кроме воздействия на флаги в регистре STATUS).
	Косвенное чтение INDF (FSR=0) даст результат 00h.
	9-й бит косвенного адреса (IRP) сохраняется в регистре STATUS<7>.
	Изменение адреса счетчика команд PC (вычисляемый переход) выполняется командой приращения к регистру PCL (ADDWF PCL,F). При этом необходимо следить, чтобы при исполнении вычисляемого перехода не происходило пересечения границы блока памяти программ (256 слов).
	PCL – младший байт (8 бит <7:0>) счетчика команд (PC), доступен для чтения и записи.
	PCH – старший байт (5 бит <12:8>) счетчика команд PC, не доступен для чтения и записи. Все операции с PCH происходят через дополнительный регистр PCLATH.
	В случае вычисляемого перехода, при переполнении PCL, инкремента PCH не происходит.
		ADDWF REG,0
		До выполнения
		W=0x17
		REG=0xC2
		После выполнения
		W=0xD9
		REG=0xC2
		Косвенная адресация
		ADDWF INDF,1
		До выполнения
		W=0x17
		FSR=0xC2 (по этому адресу “лежит” число 0x20)
		После выполнения
		W=0x17
		FSR=0xC2 (по этому адресу “лежит” число 0x37)
		Вычисляемый переход
		ADDWF PCL,1
		До выполнения
		W=0x10
		PCL=0x37
		После выполнения
		PCL=0x47
		C=0
		ADDWF PCL,1
		До выполнения
		W=0x10
		PCL=0xF7
		PCH=0x08
		После выполнения
		PCL=0x07
		PCH=0x08
		C=1
			1	C,DC,Z

ANDWF f, d Побитное И W и f	Выполняется побитное “И” содержимого регистров W и f. Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	ANDWF REG,1 До выполнения W=0x17 REG=0xC2 После выполнения W=0x17 REG=0x02 ANDWF REG,0 До выполнения W=0x17 REG=0xC2 После выполнения W=0x02 REG=0xC2 Косвенная адресация ANDWF INDF,1 До выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x5A) После выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x15)	1	Z

CLRF f Очистить f	Очистить содержимое регистра f и установить флаг Z	CLRF REG До выполнения REG=0x5A После выполнения REG=0x00, Z=1 Косвенная адресация CLRF INDF До выполнения FSR=0xC2 (по этому адресу “лежит” число 0xAA) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x00) Z=1	1	Z
CLRW Очистить W	Очистить содержимое регистра W и установить флаг Z	CLRW До выполнения W=0x5A После выполнения W=0x00 Z=1	1	Z
COMF f, d Инвертировать f	Инвертировать все биты в регистре f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f	COMF REG,0 До выполнения REG=0x13 После выполнения REG=0x13 W=0xEC COMF REG,1 До выполнения REG=0xFF После выполнения REG=0x00 Z=1 Косвенная адресация COMF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0xAA) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x55)	1	Z
DECF f, d Вычесть1 из f	Декремент содержимого регистра f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	DECF REG,1 До выполнения REG=0x01 Z=0 После выполнения REG=0x00 Z=1 DECF REG,0 До выполнения REG=0x10 W=x, Z=0 После выполнения REG=0x10 W=0x0F, Z=0 Косвенная адресация DECF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0x01) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x00) Z=1	1	Z

DECFSZ f, d Вычесть 1 из f с ветвлением	Декремент содержимого регистра f с ветвлением Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. Если результат не равен 0, то исполняется следующая инструкция. Если результат = 0, то следующая инструкция не исполняется (пропускается, вместо нее исполняется “виртуальный” NOP), а команда выполняется за 2 цикла.	LOOP DECFSZ REG,1 GOTO LOOP CONTINUE .... 1) До выполнения REG=0x01 После выполнения REG=0x00 PC=адрес CONTINUE 2) До выполнения REG=0x02 После выполнения REG=0x01 Переход на LOOP	1(2)	
INCF f, d Прибавить 1 к f	Инкремент содержимого регистра f Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	1) INCF REG,1 До выполнения REG=0xFF Z=0 После выполнения REG=0x00 Z=1 2) INCF REG,0 До выполнения REG=0x10 W=x Z=0 После выполнения REG=0x10 W=0x11 Z=0 Косвенная адресация INCF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0xFF) Z=0 После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x00) Z=1	1	Z
INCFSZ f, d Прибавить 1 к f с ветвлением	Инкремент содержимого регистра f с ветвлением Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. Если результат не равен 0, то исполняется следующая инструкция. Если результат = 0, то следующая инструкция не исполняется (пропускается, вместо нее исполняется “виртуальный” NOP), а команда выполняется за 2 цикла.	LOOP INCFSZ REG,1 GOTO LOOP CONTINUE ..... 1) До выполнения REG=0xFF После выполнения REG=0x00 PC=адрес CONTINUE 2) До выполнения REG=0x02 После выполнения REG=0x03 Переход на LOOP	1(2)	
IORWF f,d Побитное “ИЛИ” W и f	Побитное “ИЛИ” содержимого регистров W и f Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f.	1) IORWF REG,0 До выполнения REG=0x13 W=0x91 После выполнения REG=0x13 W=0x93 Z=0 2) IORWF REG,1 До выполнения REG=0x13 W=0x91 После выполнения REG=0x93 W=0x91 Z=0	1	Z

		3) IORWF REG,1 До выполнения REG=0x00 W=0x00 После выполнения REG=0x00 W=0x00 Z=1 Косвенная адресация IORWF INDF,1 До выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x30) После выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x37) Z=0		

MOVF f,d
Переслать f
	Содержимое регистра f пересылается в регистр адресата
	
	Если d=0 – значение сохраняется в регистре W
	Если d=1 – значение сохраняется в регистре f
		MOVF REG,0
		До выполнения
		W=0x00
		REG=0xC2
		После выполнения
		W=0xC2
		REG=0xC2
		Z=0
		MOVF REG,1
		1) До выполнения
		REG=0x43
		После выполнения
		REG=0x43
		Z=0
		2) До выполнения
		REG=0x00
		После выполнения
		REG=0x00
		Z=1
		
		Косвенная адресация
		MOVF INDF,1
		До выполнения
		W=0x17
		FSR=0xC2 (по этому адресу “лежит” число 0x00)
		После выполнения
		W=0x17
		FSR=0xC2 (по этому адресу “лежит” число 0x00)
		Z=1
			1	Z

MOVWF f Переслать W в f	Переслать содержимое W в f	MOVWF REG До выполнения REG=0xFF W=0x4F После выполнения REG=0x4F W=0x4F Косвенная адресация MOVWF INDF До выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x00) После выполнения W=0x17 FSR=0xC2 (по этому адресу “лежит” число 0x17)	1	

NOP	Нет операции	NOP До выполнения PC=адрес X После выполнения PC=адрес X+1	1	
RLF f, d Циклический сдвиг f влево через перенос	Выполняется циклический сдвиг влево содержимого регистра f через бит C регистра STATUS Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f  	RLF REG,0 До выполнения REG=11100110 W=xxxxxxxx C=0 После выполнения REG=11100110 W=11001100 C=1 Косвенная адресация RLF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0x3A - 00111010) C=1 После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x75 - 01110101) C=0 RLF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0xB9 - 10111001), C=0 После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x72 - 01110010) C=1	1	C
RRF f, d Циклический сдвиг f вправо через перенос	Выполняется циклический сдвиг вправо содержимого регистра f черезбит C регистра STATUS Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f  	RRF REG,0 До выполнения REG=11100110 W=xx, C=0 После выполнения REG=11100110 W=01110011, C=0 Косвенная адресация RRF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит число 0x3A – 00111010), C=1 После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x9D – 10011101), C=0 RRF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0x39- 00111001), C=0 После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x1C – 00011100), C=1	1	C

SUBWF f, d Вычесть W из f	Вычитание содержимого регистра W из регистра f. Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f.	SUBWF REG,1 До выполнения REG=0x03 W=0x02 C=x, Z=x После выполнения REG=0x01 W=0x02 C=1, Z=0 (“+” результат) SUBWF REG,1 До выполнения REG=0x02 W=0x02 C=x, Z=x После выполнения REG=0x00 W=0x02 C=1, Z=1 (“0” результат) SUBWF REG,1 До выполнения REG=0x01 W=0x02 C=x, Z=x После выполнения REG=0xFF W=0x02 C=0, Z=0 (“-“ результат)	1	C,DC,Z
SWAPF f, d Поменять местами полубайты в f	Поменять местами старший и младший полубайты регистра f. Если d=0 – результат сохраняется в регистр W Если d=1 – результат сохраняется в регистре f	SWAPF REG,0 До выполнения REG=0xA5 (1010 0101) W=x После выполнения REG=0xA5 W=0x5A **(0101 1010)** SWAPF REG,1 До выполнения REG=0xA5 После выполнения REG=0x5A Косвенная адресация SWAPF INDF,1 До выполнения FSR=0xC2 (по этому адресу “лежит” число 0x20 – 0010 0000) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x02 – 0000 0010)	1	
XORWF f,d Побитное "исключающее ИЛИ" W и f	Сравнение содержимого регистров W и f (проверка на одинаковость) Побитное "Исключающее "ИЛИ" содержимого регистров W и f Если d=0 – результат тохраняется в регистре W Если d=1 – результат сохраняется в регистре f	XORWF REG,1 До выполнения REG=0xAF W=0xB5 После выполнения REG=0x1A W=0xB5 XORWF REG,0 До выполнения REG=0xAF W=0xB5 После выполнения REG=0xAF W=0x1A Косвенная адресация XORWF INDF,1 До выполнения W=0xB5 FSR=0xC2 (значение	1	Z

		регистра с адресом в FSR=0xAF) После выполнения W=0xB5 FSR=0xC2 (значение регистра с адресом в FSR=0x1A)		
Бит - ориентированные команды (b-от 0 до 7)				
BCF f, b Установить в 0 бит b регистра f	Установить в 0 бит b регистра f	BCF REG,7 До выполнения REG=0xC7-11000111 После выполнения REG=0x47- 01000111 Косвенная адресация BCF INDF,3 До выполнения FSR=0xC2 (по этому адресу “лежит” чис-ло 0x2F – 0010 1111) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x27 – 0010 0111)	1	
BSF f, b Установить в 1 бит b регистра f	Установить в 1 бит b регистра f	BSF REG,7 До выполнения REG=0x0A- 00001010 После выполнения REG=0x8A- 10001010 Косвенная адресация BSF INDF,3 До выполнения FSR=0xC2 (по этому адресу “лежит” чис-ло 0x20 – 0010 0000) После выполнения FSR=0xC2 (по этому адресу “лежит” число 0x28 – 0010 1000)	1	
BTFSC f, b Проверить бит b в регистре f, если b=0, то пропустить следующую инструкцию	Если бит b в регистре f =1, то исполняется следующая инструкция Если бит b в регистре f =0, то следующая инструкция не исполняется (пропускается, вместо нее исполняется “виртуальный” NOP), а команда исполняется за 2 цикла.	BTFSC REG,4 GOTO LOOP TRUE . . 1) До выполнения REG=xxx0xxxx После выполнения Т.к. REG<4>=0, PC=адрес TRUE 2) До выполнения REG=xxx1xxxx После выполнения Т.к. REG<4>=1, (исполняется GOTO LOOP)	1(2)	
BTFSS f, b Проверить бит b в регистре f, если b=1, то пропустить следующую инструкцию	Если бит b в регистре f=0, исполняется следующая инструкция Если бит b в регистре f=1, то следующая инструкция не исполняется (пропускается, вместо нее исполняется “виртуальный” NOP), а команда исполняется за 2 цикла.	BTFSS REG,4 GOTO LOOP TRUE . . 1) До выполнения REG=xxx0xxxx После выполнения Т.к. REG<4>=0, (исполняется GOTO LOOP) 2) До выполнения REG=xxx1xxxx После выполнения Т.к. REG<4>=1, PC=адрес TRUE	1(2)	

Команды операций с константами (k – от 0 до 255)				
ADDLW k Сложить константу с W	Содержимое регистра W складывается с 8 – разрядной константой k. Результат сохраняется в регистре W	ADDLW 0x15 До выполнения W=0x10 После выполнения W=0x25 ADDLW REG До выполнения W=0x10 REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x47 ADDLW CONST До выполнения “Прописка” в “шапке” программы: CONST EQU 0x37 W=0x10 После выполнения W=0x47	1	C,DC,Z
SUBLW k Вычесть W из константы	Вычесть содержимое регистра W из 8 – разрядной константы k. Результат сохраняется в регистре W.	SUBLW 0x02 До выполнения W=0x01 C=? Z=? После выполнения W=0x01 C=1, Z=0 (“+” результат) SUBLW 0x02 До выполнения W=0x03 C=? Z=? После выполнения W=0xFF C=0, Z=0 (“-“ результат) SUBLW 0x02 До выполнения W=0x02 C=? Z=? После выполнения W=0x00 C=1, Z=1 (“0” результат) SUBLW REG До выполнения W=0x10 REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x27 C=1	1	C,DC,Z

MOVLW k Переслать константу в W	Переслать константу k в регистр W	MOVLW 0x5A До выполнения W=x После выполнения W=0x5A MOVLW REG До выполнения W=x REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x37 Z=0 MOVLW CONST До выполнения “Прописка” в “шапке” программы: CONST EQU 0x37 W=x После выполнения W=0x37	1	
ANDLW k Побитное “И” константы и W	Выполняется побитное “И” содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W	ANDLW 0x5F (01011111) До выполнения W=0xA3 (10100011) После выполнения W=0x03 (00000011) ANDLW REG До выполнения W=0xA3 REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x23 ANDLW CONST До выполнения “Прописка” в “шапке” программы: CONST EQU 0x37 W=0xA3 После выполнения W=0x23	1	Z
IORLW k Побитное “ИЛИ” константы и W	Выполняется побитное “ИЛИ” содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W.	IORLW 0x35 (0x35 – 00110101) До выполнения W=0x9A (10011010) После выполнения W=0xBF (10111111) Z=0 IORLW REG До выполнения W=0x9A REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x9F Z=0 IORLW CONST До выполнения W=0x9A “Прописка” в “шапке” программы: CONST EQU 0x37 После выполнения W=0x9F Z=0	1	Z

		IORLW 0x00 До выполнения W=0x00 После выполнения W=0x00 Z=1		
XORLW k Побитное "Исключающее ИЛИ" константы и W	Сравнение содержимого регистра W и константы (проверка на "одинако-вость") Выполняется побитное "Исключающее ИЛИ" содержимого регистра W и 8 – разрядной константы k. Результат сохраняется в регистре W.	XORLW 0xAF(10101111) До выполнения W=0xB5 (10110101) После выполнения W=0x1A (00011010) Z=0 XORLW REG До выполнения W=0xAF REG=0x37 (адрес регистра, а не его содержимое) После выполнения W=0x18 Z=0 XORLW CONST До выполнения W=0xAF "Прописка" в "шапке" программы: CONST EQU 0x37 После выполнения W=0x18 Z=0	1	Z
Команды управления				
CALL Условный переход (переход по стеку)	Выполнить условный переход. Адрес следующей инструкции (PC+1) "загружается" в вершину стека (TOS). 11 бит адреса "загружаются" из кода команды в счетчик команд PC<10:0>. 2 старших бита "загружаются" в счетчик команд PC<12:11> из регистра PCLATH.	HERE CALL ABC До выполнения PC=адрес HERE После выполнения PC=адрес ABC TOS=адрес HERE+1	2	
GOTO k Безусловный переход (стек не задействован)	Выполнить безусловный переход. 11 бит адреса "загружаются" из кода команды в счетчик команд PC<10:0>. 2 старших бита "загружаются" в счетчик команд PC<12:11> из регистра PCLATH.	GOTO ABC После выполнения PC= адрес ABC	2	
RETURN Возврат по стеку	Возврат из подпрограммы. Вершина стека (TOS) "выгружается" в счетчик команд PC.	RETURN После выполнения PC=TOS (адрес, "выгруженный" из TOS)	2	
RETLW k Возврат по стеку с загрузкой константы в W	Возврат из подпрограммы. В регистр W загружается 8-разряд-ная константа. Вершина стека (TOS) "выгружается" в счетчик команд PC.	CALL TABLE .... TABLE ADDWF PCL,1 RETLW k1 RETLW k2 . . RETLW kn	2	

		Возврат на адрес PC+1		
RETFIE Возврат по стеку из ПП обработки прерываний	Возврат из подпрограммы обработки прерываний. Вершина стека (TOS) загружается в счетчик команд PC. Осуществляется предварительное разрешение прерываний (бит №7 регистра INTCON {GIE} устанавливается в 1).	RETFIE После выполнения PC=TOS GIE=1	2	
CLRWDT Сброс WDT (сторожевого таймера)	Сброс WDT и предделителя (если он подключен к WDT). В регистре STATUS, биты -TO и -PD устанавливаются в 1.	CLRWDT До выполнения WDT и предделитель не сброшены После выполнения WDT и предделитель сброшены -TO=1 -PD=1	1	-TO -PD
SLEEP Переход в режим SLEEP	Переход в “спящий режим” Сброс флага включения питания (-PD) в 0. Установка флага переполнения WDT (-TO) в 1. Сброс WDT и его предделителя. Перевод м/контроллера в режим SLEEP и выключение тактового генератора.	SLEEP	1	-TO -PD
OPTION	Переслать содержимое регистра W в регистр OPTION. Инструкция поддерживается для совместимости программы с семейством PIC16C5x. Запись-чтение регистра OPTION можно выполнить прямой или косвенной адресацией. Не рекомендуется использовать при работе с другими (отличными от PIC16C5x) типами ПИКов.	OPTION		
TRIS	Переслать содержимое регистра W в регистр TRIS. Не рекомендуется использовать при работе с другими (отличными от PIC16C5x) типами ПИКов.	TRIS		

КОМАНДЫ АССЕМБЛЕРА ДЛЯ ПИКов

МНЕМОНИКА	ОПИСАНИЕ	ПРИМЕРЫ	Цикл	Флаги
Байт-ориентированные команды				
ADDWF f, d Сложение W и f	**Сложить содержимое регистров W и f.** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. **Косвенная адресация:** для ее выполнения необходимо обратиться к регистру **INDF**. Это вызовет действие с регистром, адрес которого указан в регистре **FSR**. Запись в регистр **INDF** не вызовет никаких действий (кроме воздействия на флаги в регистре **STATUS**). Чтение **INDF** (**FSR**=**0**) даст результат **00h**. **Изменение адреса счетчика команд PC (вычисляемый переход)** выполняется командой приращения к регистру **PCL** (**ADDWF PCL,1**). При этом необходимо следить, чтобы не произошло пересечения границы между блоками памяти программ (в блоке 256 слов). **PCL** – младший байт (8 бит <7:0>) счетчика команд (PC), доступен для чтения и записи. **PCH** – старший байт (5 бит <12:8>) счетчика команд PC, не доступен для чтения и записи. Все операции с **PCH** происходят через дополнительный регистр **PCLATH**. В случае вычисляемого перехода, при переполнении **PCL**, инкремента **PCH** не происходит.	**ADDWF REG,0** До выполнения **W=0x17** **REG=0xC2** После выполнения **W=0xD9** **REG=0xC2** Косвенная адресация **ADDWF INDF,1** До выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x20**) После выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x37**) Вычисляемый переход **ADDWF PCL,1** До выполнения **W=0x10** **PCL=0x37** После выполнения **PCL=0x47** **C=0** **ADDWF PCL,1** До выполнения **W=0x10** **PCL=0xF7** **PCH=0x08** После выполнения **PCL=0x07** **PCH=0x08** **C=1**	1	**C,DC,Z**
ANDWF f, d Побитное И W и f	**Выполняется побитное “И” содержимого регистров W и f.** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	**ANDWF REG,1** До выполнения **W=0x17** **REG=0xC2** После выполнения **W=0x17** **REG=0x02** **ANDWF REG,0** До выполнения **W=0x17** **REG=0xC2** После выполнения **W=0x02** **REG=0xC2** Косвенная адресация **ANDWF INDF,1** До выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x5A**) После выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x12**)	1	**Z**

CLRF f Очистить f	**Очистить содержимое регистра f и установить флаг Z**	**CLRF REG** До выполнения **REG=0x5A** После выполнения **REG=0x00** **Z=1** Косвенная адресация **CLRF INDF** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0xAA**) После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x00**) **Z=1**	1	**Z**
CLRW Очистить W	**Очистить содержимое регистра W и установить флаг Z**	**CLRW** До выполнения **W=0x5A** После выполнения **W=0x00** **Z=1**	1	**Z**
COMF f, d Инвертировать f	**Инвертировать все биты в регистре f** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f	**COMF REG,0** До выполнения **REG=0x13** После выполнения **REG=0x13** **W=0xEC** **COMF REG,1** До выполнения **REG=0xFF** После выполнения **REG=0x00** **Z=1** Косвенная адресация **COMF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0xAA**) После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x55**)	1	**Z**
DECF f, d Вычесть1 из f	**Декремент содержимого регистра f** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	**DECF REG,1** До выполнения **REG=0x01** **Z=0** После выполнения **REG=0x00** **Z=1** **DECF REG,0** До выполнения **REG=0x10** **W=x** **Z=0** После выполнения **REG=0x10** **W=0x0F** **Z=0** Косвенная адресация **DECF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x01**) После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x00**) **Z=1**	1	**Z**

DECFSZ f, d Вычесть 1 из f с ветвлением	**Декремент содержимого регистра f с ветвлением** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. ***Если результат не = 0***, то исполняется следующая инструкция. ***Если результат = 0***, то следующая инструкция не исполняется (вместо нее исполняется “виртуальный” NOP), а команда исполняется за 2 м.ц.	**LOOP DECFSZ REG,1** **GOTO LOOP** **CONTINUE ....** 1) До выполнения **REG=0x01** После выполнения **REG=0x00** **PC=адрес CONTINUE** 2) До выполнения **REG=0x02** После выполнения **REG=0x01** **Переход на LOOP**	1(2)	
INCF f, d Прибавить 1 к f	**Инкремент содержимого регистра f** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f.	1) **INCF REG,1** До выполнения **REG=0xFF** **Z=0** После выполнения **REG=0x00** **Z=1** 2) **INCF REG,0** До выполнения **REG=0x10** **W=x** **Z=0** После выполнения **REG=0x10** **W=0x11** **Z=0** Косвенная адресация **INCF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0xFF**) **Z=0** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x00**) **Z=1**	1	**Z**
INCFSZ f, d Прибавить 1 к f с ветвлением	**Инкремент содержимого регистра f с ветвлением** Если d=0, результат сохраняется в регистре W. Если d=1, результат сохраняется в регистре f. ***Если результат не = 0***, то исполняется следующая инструкция. ***Если результат = 0***, то следующая инструкция не исполняется (вместо нее исполняется “виртуальный” NOP), а команда исполняется за 2 м.ц.	**LOOP INCFSZ REG,1** **GOTO LOOP** **CONTINUE .....** 1) До выполнения **REG=0xFF** После выполнения **REG=0x00** **PC=адрес CONTINUE** 2) До выполнения **REG=0x02** После выполнения **REG=0x03** **Переход на LOOP**	1(2)	
IORWF f,d Побитное “ИЛИ” W и f	***Побитное “ИЛИ” содержимого регистров W и f*** Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f.	1) **IORWF REG,0** До выполнения **REG=0x13** **W=0x91** После выполнения **REG=0x13** **W=0x93** **Z=0** 2) **IORWF REG,1** До выполнения **REG=0x13** **W=0x91** После выполнения **REG=0x93** **W=0x91** **Z=0**	1	**Z**

		3) **IORWF REG,1** До выполнения **REG=0x00** **W=0x00** После выполнения **REG=0x00** **W=0x00** **Z=1** Косвенная адресация **IORWF INDF,1** До выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x30**) После выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x37**) **Z=0**		
MOVF f,d Переслать f	***Содержимое регистра f пересылается в регистр адресата*** Если d=0 – значение сохраняется в регистре W Если d=1 – значение сохраняется в регистре f	**MOVF REG,0** До выполнения **W=0x00** **REG=0xC2** После выполнения **W=0xC2** **REG=0xC2** **Z=0** **MOVF REG,1** 1) До выполнения **REG=0x43** После выполнения **REG=0x43** **Z=0** 2) До выполнения **REG=0x00** После выполнения **REG=0x00** **Z=1** Косвенная адресация **MOVF INDF,1** До выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x00**) После выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x00**) **Z=1**	1	**Z**
MOVWF f Переслать W в f	***Переслать содержимое W в f***	**MOVWF REG** До выполнения **REG=0xFF** **W=0x4F** После выполнения **REG=0x4F** **W=0x4F** Косвенная адресация **MOVWF INDF** До выполнения **W=0x17** **FSR=0xC2** (по этому ад-ресу “лежит” число **0x00**) После выполнения **W=0x17** **FSR=0xC2** (по этому адресу “лежит” число **0x17**)	1	

NOP	***Нет операции***	**NOP** До выполнения **PC=адрес X** После выполнения **PC=адрес X+1**	1	
RLF f, d Циклический сдвиг f влево через перенос	***Выполняется циклический сдвиг влево содержимого регистра f через бит C регистра STATUS*** Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f  	**RLF REG,0** До выполнения **REG=11100110** **W=xxxxxxxx** **C=0** После выполнения **REG=11100110** **W=11001100** **C=1** Косвенная адресация **RLF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x3A** → **00111010**) **C=1** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x75** → **01110101**) **C=0** **RLF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0xB9** → **10111001**), **C=0** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x72** → **01110010**) **C=1**	1	**C**
RRF f, d Циклический сдвиг f вправо через перенос	***Выполняется циклический сдвиг вправо содержимого регистра f черезбит C регистра STATUS*** Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f  	**RRF REG,0** До выполнения **REG=11100110** **W=xx** **C=0** После выполнения **REG=11100110** **W=01110011** **C=0** Косвенная адресация **RRF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит число **0x3A** → **00111010**), **C=1** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x9D** → **10011101**), **C=0** **RRF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x39** → **00111001**), **C=0** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x1C** → **00011100**), **C=1**	1	**C**

SUBWF f, d Вычесть W из f	***Вычесть содержимое регистра W из содержимого регистра f.*** Если d=0 - результат сохраняется в регистре W Если d=1 - результат сохраняется в регистре f.	**SUBWF REG,1** До выполнения **REG=0x03** **W=0x02** **C=x, Z=x** После выполнения **REG=0x01** **W=0x02** **C=1, Z=0** (“+” результат) **SUBWF REG,1** До выполнения **REG=0x02** **W=0x02** **C=x, Z=x** После выполнения **REG=0x00** **W=0x02** **C=1, Z=1** (“0” результат) **SUBWF REG,1** До выполнения **REG=0x01** **W=0x02** **C=x, Z=x** После выполнения **REG=0xFF** **W=0x02** **C=0, Z=0** (“-“ результат)	1	**C,DC,Z**
SWAPF f, d Поменять местами полубайты в f	***Поменять местами старший и младший полубайты регистра f.*** Если d=0 – результат сохраняется в регистр W Если d=1 – результат сохраняется в регистре f	**SWAPF REG,0** До выполнения **REG=0xA5 → 1010 0101** **W=x** После выполнения **REG=0xA5** **W=0x5A → 0101 1010** **SWAPF REG,1** До выполнения **REG=0xA5** После выполнения **REG=0x5A** Косвенная адресация **SWAPF INDF,1** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x20** **→ 0010 0000** После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x02** **→ 0000 0010**	1	
XORWF f,d Побитное "исключающее ИЛИ" W и f	***Побитное "Исключающее "ИЛИ" содержимого регистров W и f*** (проверка на одинаковость) Если d=0 – результат тохраняется в регистре W Если d=1 – результат сохраняется в регистре f	**XORWF REG,1** До выполнения **REG=0xAF** **W=0xB5** После выполнения **REG=0x1A** **W=0xB5** **XORWF REG,0** До выполнения **REG=0xAF** **W=0xB5** После выполнения **REG=0xAF** **W=0x1A**	1	**Z**

		Косвенная адресация **XORWF INDF,1** До выполнения **W=0xB5** **FSR=0xC2** (значение регистра с адресом в **FSR=0xAF**) После выполнения **W=0xB5** **FSR=0xC2** (значение регистра с адресом в **FSR=0x1A**)		
Бит - ориентированные команды (b-от 0 до 7)				
BCF f, b Установить в 0 бит b регистра f	***Установить в 0 бит b регистра f***	**BCF REG,7** До выполнения **REG=0xC7 → 11000111** После выполнения **REG=0x47 → 01000111** Косвенная адресация **BCF INDF,3** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x2F → 0010 1111**) После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x27 → 0010 0111**)	1	
BSF f, b Установить в 1 бит b регистра f	***Установить в 1 бит b регистра f***	**BSF REG,7** До выполнения **REG=0x0A → 00001010** После выполнения **REG=0x8A → 10001010** Косвенная адресация **BSF INDF,3** До выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x20 → 00100000**) После выполнения **FSR=0xC2** (по этому адресу “лежит” число **0x28 → 00101000**)	1	
BTFSC f, b Проверить бит b в регистре f, если b=0, то пропустить следующую инструкцию	***Если бит b в регистре f =1, то исполняется следующая инструкция*** ***Если бит b в регистре f =0, то следующая инструкция не исполняется (пропускается, вместо нее исполняется “виртуальный” NOP), а команда исполняется за 2 м.ц.***	**BTFSC REG,4** **GOTO LOOP** **TRUE …….** 1) До выполнения **REG=xxx0xxxx** После выполнения Т.к. REG<4>=0, исполняется **TRUE** 2) До выполнения **REG=xxx1xxxx** После выполнения Т.к. REG<4>=1, исполняется **GOTO LOOP**	1(2)	

BTFSS f, b Проверить бит b в регистре f, если b=1, то пропустить следующую инструкцию	***Если бит b в регистре f=0, исполняется следующая инструкция*** ***Если бит b в регистре f=1, то следующая инструкция не исполняется (пропускается, вместо нее исполняется "виртуальный" NOP), а команда исполняется за 2 м.ц.***	**BTFSS REG,4** **GOTO LOOP** **TRUE …….** 1) До выполнения **REG=xxx0xxxx** После выполнения Т.к. REG<4>=0, исполняется **GOTO LOOP** 2) До выполнения **REG=xxx1xxxx** После выполнения Т.к. REG<4>=1, исполняется **TRUE**	1(2)	
Команды операций с константами (k – от 0 до 255)				
ADDLW k Сложить константу с W	***Содержимое регистра W складывается с 8- разрядной константой k*** Результат сохраняется в регистре W	**ADDLW 0x15** До выполнения **W=0x10** После выполнения **W=0x25** **ADDLW REG** До выполнения **W=0x10** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0x47** **ADDLW CONST** До выполнения "Прописка в шапке" программы: **CONST EQU 0x37** **W=0x10** После выполнения **W=0x47**	1	**C,DC,Z**
SUBLW k Вычесть W из константы	***Вычесть содержимое регистра W из 8-разрядной константы k*** Результат сохраняется в регистре W.	**SUBLW 0x02** До выполнения **W=0x01** **C=? Z=?** После выполнения **W=0x01** **C=1, Z=0** ("+" результат) **SUBLW 0x02** До выполнения **W=0x03** **C=? Z=?** После выполнения **W=0xFF** **C=0, Z=0** ("-" результат) **SUBLW 0x02** До выполнения **W=0x02** **C=? Z=?** После выполнения **W=0x00** **C=1, Z=1** ("0" результат) **SUBLW REG** До выполнения **W=0x10** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0x27** **C=1**	1	**C,DC,Z**

MOVLW k Переслать константу в W	***Переслать константу k в регистр W***	**MOVLW 0x5A** До выполнения **W=x** После выполнения **W=0x5A** **MOVLW REG** До выполнения **W=x** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0x37** **Z=0** **MOVLW CONST** До выполнения "Прописка в шапке" программы: **CONST EQU 0x37** **W=x** После выполнения **W=0x37**	1	
ANDLW k Побитное "И" константы и W	***Выполняется побитное "И" содержимого регистра W и 8-разрядной константы k*** Результат сохраняется в регистре W	**ANDLW 0x5F** → 01011111 До выполнения **W=0xA3 → 10100011** После выполнения **W=0x03 → 00000011** **ANDLW REG** До выполнения **W=0xA3** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0x23** **ANDLW CONST** До выполнения "Прописка в шапке" программы: **CONST EQU 0x37** **W=0xA3** После выполнения **W=0x23**	1	**Z**
IORLW k Побитное "ИЛИ" константы и W	***Выполняется побитное "ИЛИ" содержимого регистра W и 8-разрядной константы k*** Результат сохраняется в регистре W.	**IORLW 0x35** → 00110101 До выполнения **W=0x9A → 10011010** После выполнения **W=0xBF → 10111111** **Z=0** **IORLW REG** До выполнения **W=0x9A** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0xBF** **Z=0** **IORLW CONST** До выполнения **W=0x9A** "Прописка в шапке" программы: **CONST EQU 0x37** После выполнения **W=0x9F** **Z=0** **IORLW 0x00** До выполнения **W=0x00** После выполнения **W=0x00** **Z=1**	1	**Z**

XORLW k Побитное "Исключающее ИЛИ" константы и W	***Выполняется побитное "Исключающее ИЛИ" содержимого регистра W и 8-разрядной константы k*** (проверка на "одинаковость") Результат сохраняется в регистре W.	**XORLW 0xAF** → 10101111 До выполнения **W=0xB5 → 10110101** После выполнения **W=0x1A → 00011010** **Z=0** **XORLW REG** До выполнения **W=0xAF** **REG=0x37** (адрес регистра, а не его содержимое) После выполнения **W=0x98** **Z=0** **XORLW CONST** До выполнения **W=0xAF** "Прописка в шапке" программы: **CONST EQU 0x37** После выполнения **W=0x18** **Z=0**	1	**Z**
Команды управления				
CALL Условный переход (переход по стеку)	***Выполнить условный переход*** Адрес следующей инструкции (PC+1) загружается в вершину стека. 11 бит адреса загружаются, в счетчик команд, из кода команды. 2 старших бита загружаются, в счетчик команд, из регистра PCLATH.	**HERE CALL ABC** До выполнения **PC=адрес HERE** После выполнения **PC=адрес ABC** **В вершине стека, адрес HERE+1**	2	
GOTO k Безусловный переход (стек не задействован)	***Выполнить безусловный переход*** 11 бит адреса загружаются, в счетчик команд, из кода команды. 2 старших бита загружаются, в счетчик команд, из регистра PCLATH.	**GOTO ABC** После выполнения **PC= адрес ABC**	2	
RETURN Возврат по стеку	***Возврат из подпрограммы*** Содержимое вершины стека "выгружается" в счетчик команд PC.	**RETURN** После выполнения **PC= адресу, который "выгружен" из вершины стека.**	2	
RETLW k Возврат по стеку с загрузкой константы в W	***Возврат из подпрограммы с установленной константой*** В регистр W загружается 8-разрядная константа Содержимое вершины стека "выгружается" в счетчик команд PC.	**CALL TABLE** **.....** **.....** **TABLE ADDWF PCL,1** **RETLW k1** **RETLW k2** **……** **……** **RETLW kn** После выполнения **PC= адресу, который "выгружен" из вершины стека.** **W = k**	2	

RETFIE Возврат по стеку из ПП обработки прерываний	***Возврат из подпрограммы обработки прерываний*** Содержимое вершины стека "выгружается" в счетчик команд PC. Осуществляется глобальное разрешение прерываний (бит GIE устанавливается в 1).	**RETFIE** После выполнения **PC= адресу, который "выгружен" из вершины стека.** **GIE=1**	2	
CLRWDT Сброс WDT (сторожевого таймера)	***Сброс WDT и предделителя*** (если он подключен к WDT). Биты -TO и -PD устанавливаются в 1.	**CLRWDT** До выполнения **WDT и предделитель не сброшены** После выполнения **WDT и предделитель сброшены** **-TO=1** **-PD=1**	1	**-TO** **-PD**
SLEEP Переход в режим SLEEP	***Переход в режим пониженного энергопотребления (в "спящий режим")*** Сброс флага включения питания (-PD) в 0. Установка флага переполнения WDT (-TO) в 1. Сброс WDT и предделителя. Выключение тактового генератора и перевод м/контроллера в режим SLEEP.	**SLEEP**	1	**-TO** **-PD**
OPTION	***Переслать содержимое регистра W в регистр OPTION.*** Инструкция поддерживается для совместимости программы с семейством PIC16C5x. Запись-чтение регистра OPTION можно выполнить прямой или косвенной адресацией. Не рекомендуется использовать при работе с другими (отличными от PIC16C5x) типами ПИКов.	**OPTION**		
TRIS	***Переслать содержимое регистра W в регистр TRIS.*** Не рекомендуется использовать при работе с другими (отличными от PIC16C5x) типами ПИКов.	**TRIS**		