(Продолжение. Начало – № 4–8/2001)

Микроконтроллеры? Это же просто!

Глава 3. Регистры микроконтроллера

Одними из основных элементов архитектуры микроконтроллера являются его регистры (напомню, что регистры – это ячейки памяти внутри МК, обмен информацией между которыми осуществляется простыми и короткими командами). В предыдущих главах я уже вскользь упоминал о них, и даже воспользовался некоторыми из них при рассмотрении примеров сопряжения МК с параллельными и последовательными АЦП. Однако я пока еще не сказал ни сколько их в рассматриваемых нами микроконтроллерах, и каковы их свойства, ни каковы команды работы с ними, а также ни словом не обмолвился о тех регистрах, которые отвечают за работу находящихся внутри микроконтроллера таймеров, счетчиков, приемопередатчиков и т. д. Настоящая глава призвана частично восполнить этот пробел.

Обращаю ваше внимание на то, что хотя в полном объеме наше с вами знакомство с системой команд микроконтроллера еще не состоялось, мы регулярно анализировали и будем продолжать анализировать фрагменты программ, содержащие эти команды. Объясняется это тем, что внешние выводы и формируемые на них сигналы с одной стороны, внутренняя структура МК (память, регистры и т. д.) с другой и система команд с третьей теснейшим образом переплетены между собой. Нельзя рассказывать о чем-то одном, не касаясь другого и третьего. Поэтому я вынужден был в первых главах лишь вскользь упомянуть о регистрах, не давая более развернутого их описания — если бы я попытался это сделать, материал стал бы намного более трудным для восприятия (а он и без того нелегок). Точно также я стараюсь лишь в минимальной степени знакомить вас с командами, говоря только то, что они предписывают микроконтроллеру, опуская их формат, длину, время выполнения, разновидности и т. д. Все это у нас впереди. Но если кто-то из вас, дорогие читатели, готов воспринять эту усложняющую материал дополнительную информацию и желает поскорее с ней ознакомиться — в вашем распоряжении список литературы, опубликованный в самом начале цикла (“Схемотехника”, №4, 2001 г., стр. 14), пользуйтесь ей. Ну а мы с теми, кто не горит желанием забежать вперед, потихоньку двинем дальше.

Регистры общего назначения и слово состояния программы

Прежде, чем начать рассказ о регистрах МК семейства x51, я предлагаю совершить небольшой экскурс в историю микропроцессоров.

В середине 70-х годов прошлого века, когда микропроцессоры уже миновали этап становления, а микроконтроллеры еще не выделились из них в самостоятельный класс изделий, наиболее совершенными микросхемами были 8-разрядные процессоры второго поколения 8080, 8085 фирмы Intel, 6800 фирмы Motorola и Z80 фирмы Zilog. Технология, по которой они создавались, не позволяла разместить на кристалле более 5—7 тысяч транзисторов, поэтому при создании этих микросхем разработчикам приходилось до разумных пределов уменьшать количество входящих в них элементов и узлов.

По этой причине упомянутые микропроцессоры имели в своем составе всего по десятку регистров. Одним из них был счетчик команд PC (Programm Counter), содержимое которого увеличивалось после выполнения каждой команды. Его я поставил на первое место в списке регистров, поскольку нет такого микропроцессора или микроконтроллера, в котором бы он отсутствовал. Помимо него, все упомянутые микропроцессоры имели один или два регистра-аккумулятора, с содержимым которых можно было выполнять любые доступные микропроцессору действия по перемещению и обработке данных. Далее, 6800 имел четыре регистра, где программист мог размещать адреса ячеек памяти, с которыми ему предстояло работать. В противовес ему, остальные три микропроцессора имели по шесть так называемых регистров общего назначения (РОН), в которых могли храниться не только адреса ячеек памяти, но и константы, переменные, значения функций — словом, любые данные. И этим резервы всех микропроцессоров практически исчерпывались Исключение составлял лишь Z80, который, в сравнении с 8080 и 8085, имел двойной комплект аккумуляторов и регистров общего назначения (правда, в каждый конкретный момент времени доступным был только один аккумулятор и один комплект РОН).

Зачем микропроцессору нужны были регистры типа РОН? Не проще ли было работать без них, храня все обрабатываемые данные в памяти? В общем, может в чем-то и проще. Но уж точно, практически на порядок медленнее. С числами, расположенными в регистрах, микропроцессор выполнял те или иные действия гораздо быстрее, чем с теми же данными, расположенными во внешней оперативной памяти (внутренняя была еще не по плечу существовавшей тогда технологии). Так, сложение двух чисел, находящихся в РОН, занимало несколько тактов (единицы мкс). Сложение двух чисел, расположенных в ОЗУ, требовало в 5—10 раз большего числа тактов, и выполнялось за 20—50 мкс. Поэтому алгоритмы наиболее часто выполняемых фрагментов программы составлялись таким образом, чтобы уменьшить число требуемых для их выполнения констант и переменных до того количества, которое целиком (или почти целиком) разместилось бы в имеющихся регистрах. И, надо сказать, большинство программистов успешно справлялось с этой задачей.

В отличие от упомянутых микропроцессоров, появившиеся в конце 70-х первые микроконтроллеры уже содержали на кристалле несколько десятков ячеек оперативной памяти. Часть из них была отведена под регистры, которых стало возможным сделать больше, чем у 8085 или Z80. Таким образом, программы стало писать заметно легче – большое количество регистров и дополняющая их почти столь же быстрая внутренняя оперативная память практически сняли ограничения на число используемых программой переменных и констант. Так что вам, дорогие читатели, учиться писать программы будет гораздо проще, чем тем, кто учился этому лет 20 назад.

При этом, однако, МК семейства x51, первоначально выпущенные Intel, имели (и имеют) довольно много общего с ею же разработанными микропроцессорами 8080 и 8085. Так, рассматриваемые нами микроконтроллеры содержат один аккумулятор и 8 регистров общего назначения, обозначаемых как R0, R1, R2, …, R6, R7. Правда, если быть точным, этих регистров не 8, а 32 (4 так называемых банка по 8 в каждом), но, как и в Z80, доступным в каждый момент может быть лишь один банк. Поэтому лучше считать, что x51 имеют всего 8 РОН.

Регистры общего назначения расположены в первых 32 ячейках внутренней памяти данных микроконтроллера (рис. 10). R0 первого банка располагается в ячейке с адресом 0, R1 — с адресом 1, и т. д. вплоть до R7 (с адресом 7). В ячейках с адресами с 8-го по 15-й расположены РОН второго банка (R0 — в 8-й, R1 — в 9-й, …, R7 — в 15-й). Третий банк, как вы уже наверное догадались, занимает ячейки с адресами с 16-го по 23-й (R0 — в 16-м, R7 — в 23-м). Ну а четвертый — правильно, в 24-й, 25-й и т. д., вплоть до 31-й ячейки памяти.

Как отмечалось выше, в каждый конкретный момент времени доступны регистры общего назначения только одного банка — первого, второго, третьего или четвертого. А какого? И как переключать банки РОН?

Для ответа на этот вопрос нам необходимо познакомиться с еще одним регистром — словом состояния программы

PSW (Program Status Word). Его структура приведена на рис. 11.

Сразу обращаю ваше внимание на биты RS1, RS0. Именно их состояние определяет, какой из банков РОН задействован в настоящий момент. Если RS1=0, RS0=0, то мы работаем с первым банком, расположенным в ячейках памяти данных с адресами 0—7. Комбинация RS1=0, RS0=1 означает, что мы работаем со вторым банком (ячейки с адресами 8—15). RS1=1, RS0=0 означает, что мы работаем с третьим банком (ячейки с адресами 16–23), а RS1=1, RS0=1 — что с четвертым (ячейки 24–31).

При старте МК RS1=0, RS0=0, то есть вначале мы всегда работаем с первым банком РОН. А если нам захотелось переключиться на четвертый? Для этого в программу всего-навсего нужно вставить уже знакомые нам две команды, которые изменят состояние битов RS1, RS0:

```
;
SETB   PSW.4        ;УСТАНОВКА В 1 БИТА RS1
SETB   PSW.3        ;УСТАНОВКА В 1 БИТА RS0
;
```

Естественно, установка бита RS1 или RS0 в 0 осуществляется командой CLR PSW.4 или CLR PSW.3 соответственно. Таким образом, комбинируя четыре вышеупомянутые команды, мы можем заставить МК работать с любым из банков РОН.

Какие еще биты содержаться в регистре PSW? Старший, седьмой бит – это уже не раз упоминавшийся бит переноса CY. Кстати, программисты иногда вместо термина “бит переноса” употребляют выражение “флаг переноса”. Когда CY=1, они говорят, что флаг переноса установлен, когда CY=0 — флаг сброшен.

Шестой бит (PSW.6) — это дополнительный флаг переноса AC. Он используется при выполнении арифметических операций, и мы еще упомянем о нем в разделе, посвященном системе команд микроконтроллера. Также при выполнении арифметических операций используется и бит переполнения (или флаг переполнения) OV (PSW.2). Бит (флаг) четности P (PSW.0) обычно используется при передаче информации от одного МК к другому. Он передается вместе с передаваемым 8-битовым словом в качестве дополнительного девятого бита. Если в процессе передачи произошел сбой, то четность принятого слова может отличаться от четности переданного, и флаг четности при проверке слова принимающим микроконтроллером не совпадет с переданным в качестве девятого бита флагом четности передающего МК. В этом случае необходимо повторить пересылку ошибочного слова. Все это звучит весьма мудрено, и не расстраивайтесь, если не все понятно – мы это еще обсудим при рассмотрении приемопередатчика микроконтроллера.

Отдельно скажу о флаге пользователя F0 (бит PSW.5). Командами CLR PSW.5 или SETB PSW.5 вы из своей программы можете сбрасывать или устанавливать его при выполнении тех или иных важных для вас условий, например, если сработал какой-то из опрашиваемых датчиков, или нажата кнопка на клавиатуре. Далее, анализируя состояние этого флага, ваша программа выполняет те или иные действия, в зависимости от того, установлен ли он или сброшен. При этом отмечу, что если вы установили этот флаг, например, в единицу, его состояние останется неизменным до тех пор, пока вы самостоятельно не сбросите его командой CLR PSW.5, и никакие действия МК не приведут к его самостоятельному, без вашего вмешательства сбросу. Именно поэтому F0 называется флагом пользователя — только пользователь может изменять состояние этого бита.

Итак, мы ознакомились со всеми используемыми битами слова состояния PSW (бит PSW.1 не используется).

Аккумулятор, расширитель аккумулятора, указатель стека и механизм вызова подпрограмм

Перечисленные в этом подзаголовке три регистра, наряду с РОН, являются наиболее часто используемыми у МК семейства x51. Поэтому познакомимся с ними поближе.

Регистр-аккумулятор или просто аккумулятор мы упоминали уже не один раз. Аккумулятор — основной регистр МК x51. Это означает, что у нашего МК нет второго такого регистра, содержимое которого можно было бы сложить с содержимым *любого другого* регистра, переместить в него содержимое *любого другого* регистра или *любой* ячейки памяти. Содержимое аккумулятора можно сдвигать, побитно инвертировать (все нули заменить единицами и наоборот), анализировать, и в зависимости от его состояния выполнять те или иные фрагменты программы. Когда мы познакомимся с системой команд, вы убедитесь, что большинство из них так или иначе затрагивает аккумулятор.

Прежде, чем двинуться дальше, совершим еще одно небольшое лирическое отступление. Дело в том, что далеко не во всех МК существует один-единственный аккумулятор, играющий столь важную роль в его архитектуре. Есть микроконтроллеры с двумя равноправными аккумуляторами. Все действия, которые можно сделать с одним из них, можно сделать и с другим. Команд обработки данных таким микроконтроллерам требуется поменьше, чем есть у x51, да и сами команды покороче и выполняются быстрее. Казалось бы, это весьма существенное преимущество, и благодаря ему двуаккумуляторные МК должны были бы вытеснить одноаккумуляторные. Ан нет. Прежде, чем выполнить те или

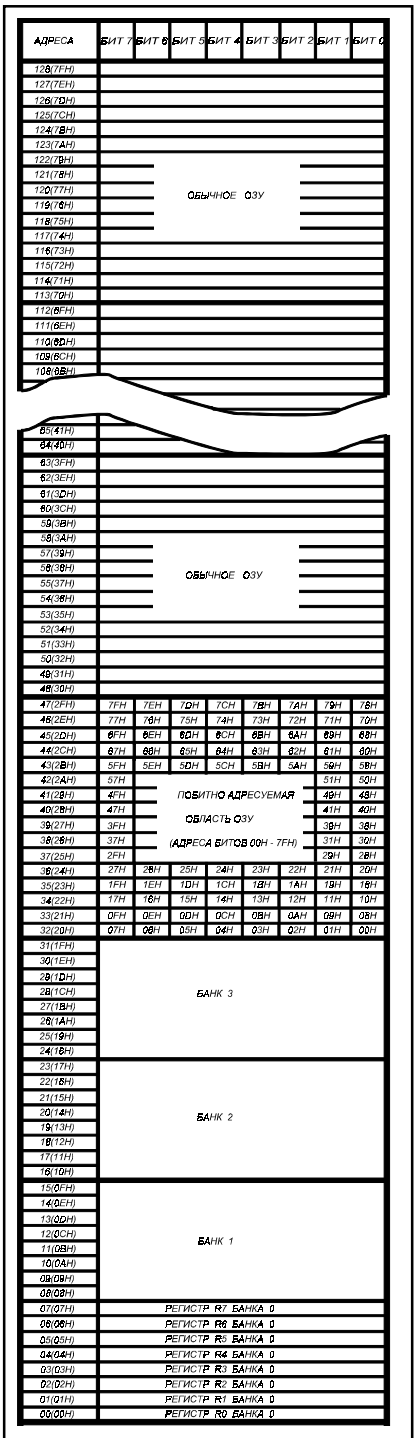

Рис. 10. Структура внутренней памяти данных МК x51

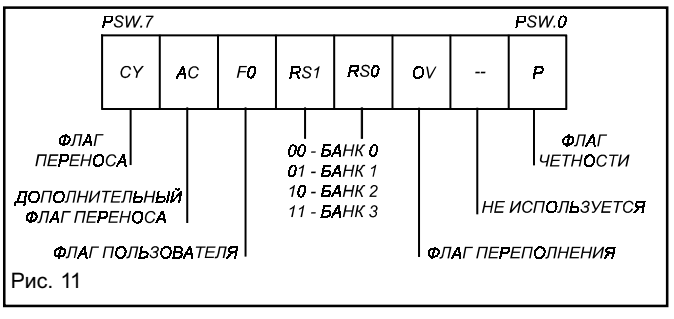

Рис. 11

иные действия над данными в обоих аккумуляторах, их нужно и в тот, и в другой занести. Кроме того, после выполнения операции необходимо также переписать данные из обоих аккумуляторов обратно в память.

В одноаккумуляторной же структуре данные из памяти нужно переносить только в один аккумулятор, ибо в качестве источника второго числа (второго слагаемого или сомножителя, вычитаемого или делителя) мы предпишем контроллеру использовать ту ячейку памяти, где это число хранится, и его не потребуется куда-либо переносить. Так что одноаккумуляторная структура, хотя и уступает двухаккумуляторной в скорости обработки данных, но превосходит ее в скорости подготовки данных для обработки. Таким образом, обе они оказываются практически эквивалентными как с точки зрения быстродействия, так и по функциональным возможностям. Именно поэтому ни та, ни другая архитектура не смогла вытеснить конкурирующую. Так что имейте ввиду, что в иных, отличных от x51 микроконтроллерах, аккумуляторов может быть и 2, и даже 32, пусть это вас не смущает.

Как я уже сказал, в аккумулятор мы можем перенести данные из любого регистра общего назначения командой MOV A,Rn, где n=0...7 или из любой ячейки памяти. Последнее осуществляется командой MOV A,addr, где addr — принимающий значения от 0 до 255 адрес ячейки памяти. Мы можем также занести в аккумулятор любое целое число из диапазона 0–255 (командой MOV A,#data, где data=0–255; не забудьте про знак #, без него МК воспримет число, записанное после символа A, не как данные, а как адрес ячейки памяти). Естественно, из аккумулятора можно вернуть данные в любой регистр или в любую ячейку памяти командами MOV Rn, A и MOV addr, A соответственно. К нему можно прибавить содержимое любого РОН, любого 8-битного числа или любой ячейки памяти командами ADD A,Rn; ADD A,#data и ADD A,addr соответственно. С тем же успехом при помощи соответствующей команды SUBB из него можно вычесть содержимое любого РОН, любого 8-битного числа или любой ячейки памяти. И это далеко не полный перечень того, что можно делать с содержимым аккумулятора x51. Ни с каким другим регистром микроконтроллера мы не в состоянии проделать всего этого.

О регистре B мы также уже упоминали, чаще всего он используется при умножении и делении. Команда MUL AB осуществляет быстрое (за 4 мкс при тактовой частоте 12 МГц) перемножение чисел, хранящихся в аккумуляторе и в регистре B. После завершения умножения в B хранятся старшие 8 бит результата, а в аккумуляторе – младшие 8 бит. Командой DIV AB осуществляется деление содержимого аккумулятора на содержимое регистра B, результат деления – в аккумуляторе, остаток от деления — в B.

Отмечу, что последняя команда — довольно бестолковая, ибо делимое в ней должно быть не более 255. А как быть, если вам нужно разделить, например, результат измерения, полученный при помощи уже упоминавшихся 12-разрядных АЦП (он может быть в пределах 0–4095), например, на 10 или на 100? Командой DIV AB здесь не воспользуешься. Для таких случаев надо оставить соответствующие подпрограммы, которые позволят выполнить подобные действия. Но об этом – чуть позже.

Еще один важный регистр — SP (Stack Pointer), или указатель стека. Термин этот многим незнаком, поэтому я постараюсь поподробнее объяснить, что такое стек, и какова функция регистра SP. Но прежде нам нужно понять, что такое подпрограммы, и зачем они нужны.

В предыдущей главе было рассмотрено сопряжение МК с параллельными и последовательными АЦП и проанализированы программы, которые обеспечивали считывание результата преобразования в регистры R4, R5. Но если вдуматься, ценность этих программ почти нулевая – ну считали, а дальше-то что? Нужно хотя бы отобразить считанное. И потом, сами по себе коды считанного результата часто неинформативны — нам нужно измерять ток, напряжение, температуру и т. д. А для получения этих величин результат считывания необходимо преобразовать — на что-то умножить, с чем-то сложить. Иногда нужно провести предварительное измерение какого-то параметра, а последующее преобразование осуществлять с учетом этого результата. Например, пусть мы используем наш МК в системе, измеряющей температуру чего-то с помощью термопары (две сваренных в одной точке проволочки из различных материалов). Термопара имеет такое свойство, что прежде, чем с ее помощью что-то измерить, нужно знать температуру тех концов входящих в нее проволочек, которые соединены с измерительным прибором (у нас — с АЦП). Следовательно, перед измерением с помощью термопары мы должны каким-то датчиком измерить температуру упомянутых концов, а затем по известным специалистам формулам внести соответствующую поправку в результат последующего измерения, выполняемого при помощи самой термопары.

Вот мы и добрались до главного. Нам для получения результата нужно делать не одно, а два измерения — сначала измерить температуру концов, а затем — сигнал с термопары. (Я сейчас не вдаюсь в подробности аппаратной реализации такой задачи — ясно, что в состав системы должен входить мультиплексор, переключающий АЦП с термопары на датчик и наоборот, какие-то усилители и т. д.). Важно то, что в ходе выполнения неведомой нам пока программы термопарного измерения нам как минимум дважды нужно запустить АЦП на преобразование и считать его результат. То есть получается, что содержимое программы par_adc.a51 (или ser_adc.a51) в этой программе должно повторяться минимум дважды! А может и трижды, или четырежды — зависит от того, что мы захотели от нашего МК и как мы это программно реализовали.

Давайте теперь вспомним, что память программ у микроконтроллеров не бездонная – всего несколько тысяч ячеек. И ее всегда не хватает. А тут получается, что нам приходится часто повторять в нашей программе одни и те же фрагменты. Возникает законный вопрос — нельзя ли как-то лишь однажды написать эти фрагменты, а затем просто обращаться к ним по мере необходимости?

Радуйтесь, можно. Вы пишите этот фрагмент (программисты его называют подпрограммой) один раз, присваиваете ему какое-то понятное вам имя и дальше вызываете его по мере необходимости командой LCALL. Например, программу (пардон, теперь уже подпрограмму) измерения при помощи параллельного АЦП вы назвали IZMPAR и собираетесь дважды использовать в своем алгоритме. Тогда ваша программа будет выглядеть следующим образом:

```
;           НАЧАЛО ПРОГРАММЫ ДО МОМЕНТА ПЕРВОГО
;           ВЫЗОВА ПОДПРОГРАММЫ ИЗМЕРЕНИЯ С
;           ПАРАЛЛЕЛЬНЫМ АЦП МЫ ЗДЕСЬ НЕ ПРИВОДИМ
;
  MOV R6,A                  ;ЭТО КОМАНДА ИЗ ПРЕДШЕСТВУЮ-
ЩЕГО
    ;ФРАГМЕНТА,
      LCALL IZMPAR          ;А ВОТ ЭТО – ПЕРВЫЙ ВЫЗОВ
; ПОДПРОГРАММЫ ИЗМЕРЕНИЯ
      MOV   A,R4            ;ЭТО КОМАНДА СЛЕДУЮ-
ЩЕГО
  ;ФРАГМЕНТА – ОБРАБАТЫВАЕМ РЕЗУЛЬТАТ
;
;           СЛЕДУЮЩИЙ КУСОК ПРОГРАММЫ ДО МОМЕНТА
;           ВТОРОГО ВЫЗОВА ПОДПРОГРАММЫ ИЗМЕРЕНИЯ С
;           ПАРАЛЛЕЛЬНЫМ АЦП МЫ ОПЯТЬ ОПУСКАЕМ
;
  ADD R2,#6                 ;ЭТО КОМАНДА ИЗ ПРЕДШЕСТВУЮ-
ЩЕГО
    ;ФРАГМЕНТА,
      LCALL IZMPAR          ;А ВОТ ЭТО – ВТОРОЙ ВЫЗОВ
; ПОДПРОГРАММЫ ИЗМЕРЕНИЯ
      MOV   A,R5            ;ЭТО КОМАНДА СЛЕДУЮ-
ЩЕГО
  ;ФРАГМЕНТА – ОПЯТЬ ОБРАБАТЫВАЕМ РЕЗ.
;
;           ИДУЩАЯ ДАЛЕЕ ЧАСТЬ ПРОГРАММЫ, СВЯЗАННАЯ
;           С ОБРАБОТКОЙ И ОТОБРАЖЕНИЕМ, МЫ ЕЕ ТАКЖЕ
ОПУСКАЕМ
;
  LJMP START                ;ЭТО КОМАНДА – ОКОНЧАНИЕ ОС-
НОВНОЙ
  ;ЧАСТИ ПРОГРАММЫ И ВОЗВРАТ К НАЧАЛУ
```

```
;
;               А ВОТ ТЕПЕРЬ ПОШЛИ ПОДПРОГРАММЫ,
;               И ПЕРВАЯ ИЗ НИХ – IZMPAR
;
IZMPAR:                                 ;ЕЕ ИМЯ С ОБЯЗАТЕЛЬНЫМ
;ДВОЕТОЧИЕМ НА КОНЦЕ
      MOV    P1,#11111111B  ;НАЧАЛЬНАЯ УСТАНОВКА
MOV    P3,#11111111B
;
L7880:                                  ;СОБСТВЕННО ЧТЕНИЕ
;
      CLR    CONVST        ;ИМПУЛЬС СТАРТА ПРЕОБРАЗОВАНИЯ
SETB   CONVST
;
NOP                  ;ЗАДЕРЖКА НА ВРЕМЯ ПРЕОБРАЗОВАНИЯ
NOP
;
;               ИДУЩАЯ ДАЛЕЕ ЧАСТЬ ПОДПРОГРАММЫ
;               ПРОПУЩЕНА ДЛЯ ЭКОНОМИИ ЖУРНАЛЬНОЙ ПЛО –
ЩАДИ
;
MOV    R4,A          ;СОХРАНЯЕМ ИХ В R4
;
MOV    A,P3          ;ЧИТАЕМ ИЗ ПОРТА P3 СТ. ТЕТРАДУ
MOV    R5,A          ;В R5R4 – РЕЗУЛЬТАТ
;
SETB   RD            ;УСТАНОВКА RD В 1
SETB   CS            ;УСТАНОВКА CS В 1
;
RET                             ;КОМАНДА ЗАВЕРШЕНИЯ ПОДПРОГ –
РАММЫ
;
;
;               ДАЛЕЕ ИДУТ ДРУГИЕ ПОДПРОГРАММЫ, СВЯЗАННЫЕ
;               С ОБРАБОТКОЙ И ОТОБРАЖЕНИЕМ
;
```

В приведенном фрагменте пропущены куски программы, содержание которых для нас в данный момент не важно. Важными являются лишь следующие аспекты:

- в тех местах, где нам нужно вызвать подпрограмму (в данном случае IZMPAR), мы ставим команду LCALL IZMPAR;
- непосредственно перед первой командой, входящей в состав подпрограммы, мы ставим ее имя, оканчивающееся двоеточием. Как мы помним из предыдущего материала, двоеточие ставится в конце метки. Так что имя подпрограммы для нашей программы является меткой, куда нужно перейти для того, чтобы подпрограмму выполнить;
- в конце подпрограммы стоит пока еще неизвестная нам команда RET.

А теперь — внимание! Микроконтроллер, выполняя программу, дошел до того места, где нужно вызывать подпрограмму измерения IZMPAR. Здесь он встретил команду LCALL IZMPAR, имя которой является меткой, куда ему нужно перейти, чтобы найти коды этой так необходимой нам подпрограммы. Перейдя туда, он начал выполнять команды подпрограммы, вплоть до самого ее конца. А затем… Да, как вы думаете, что будет затем?

Если кто еще не догадался — подскажу. После того, как подпрограмма выполнена, микроконтроллер должен выполнять команду, которая стоит в программе следующей после команды вызова подпрограммы. Так, в программе на рис.15 после того, как первый вызов подпрограммы будет завершен, МК должен будет выполнить команду MOV A,R4., а после второго — MOV A,R5. А как, завершив подпрограмму, МК перейдет к выполнению этих команд, ведь перед ними нет никаких меток, которые подсказали бы ему, куда переходить?

Вот мы наконец и добрались до стека. Стек — это какое-то количество ячеек памяти, в которые микроконтроллер перед переходом на исполнение подпрограммы заносит адрес той самой следующей команды, которую ему предстоит выполнять после завершения подпрограммы. Как видите, и здесь все просто. Наткнувшись на команду вызова подпрограммы LCALL, МК сохраняет в стеке адрес следующей за ней команды и отправляется выполнять подпрограмму. А в конце любой подпрограммы – запомните это! — должна стоять команда возврата RET. Как только МК доберется до нее, для него это послужит сигналом, что подпрограмма завершена, и он, прочитав из стека сохраненный в нем адрес следующей команды, перейдет на ее выполнение. Таким образом, вы можете вызывать из вашей программы интересующую вас подпрограмму хоть сто раз, и это не приведет к безумному раздуванию объема программы за счет сотни повторяющихся одинаковых кусков. Подпрограмма будет написана вами всего однажды, а везде, где нужно ее выполнить, вы поставите всего-навсего одну команду — вызова этой подпрограммы.

Стек располагается в оперативной памяти микропроцессора или микроконтроллера. А на конкретную ячейку, где хранится адрес команды, той самой, следующей за вызовом подпрограммы, указывает именно регистр SP. Да, забыл сказать, адрес этой команды, следующей за вызовом подпрограммы, обычно называют адресом возврата.

Забавно — самому регистру SP посвящен всего один абзац. Но прежде, чем о нем упомянуть, понадобилось десять абзацев с описанием того, что такое подпрограммы, и каков механизм их вызова и возврата из них.

Александр Фрунзе
alex.fru@mtu-net.ru

Продолжение следует