Лабораторные работы №1, 2

Проектирование устройства управления шаговым двигателем на основе микроконтроллера Intel 8051

1. Постановка задачи проектирования

Целью работы является создание и отладка программы управления шаговым двигателем на языке ассемблера для устройства управления на основе микроконтроллера Intel 8051.

Задание на проектирование. Необходимо спроектировать микропроцессорное устройство на основе однокристального микроконтроллера, которое управляет шаговым двигателем (ШД) с четырьмя обмотками по командам от устройства управления верхнего уровня, которое подключается к микроконтроллеру по последовательному интерфейсу.

Один из возможных вариантов схемы устройства приведен на рис. 1. Характеристики проектируемого устройства в соответствии с вариантом приведены в табл. 3.

Варианты задания различаются между собой режимами работы микроконтроллера. Во всех вариантах используется обмен с устройством верхнего уровня по последовательному каналу. Режим обмена – асинхронный 10 (десять) или 11 (одиннадцать) бит. В каждом варианте указана частота кварцевого резонатора для генератора тактовой частоты микроконтроллера.

Устройство верхнего уровня задает микроконтроллеру координату (количество шагов) и направление движения ШД в виде одного байта, где старший бит определяет направление движения: 0 – влево, 1 – вправо. Остальные семь бит определяют количество шагов. Максимальное количество шагов, которое может сделать ШД за одну команду – 128. Если требуется большее количество шагов, то посылается следующая команда микроконтроллеру. В зависимости от варианта используются две схемы включения ШД – с делением и без деления шага. Количество шагов в единицу времени фиксировано и приведено в табл. 3.

Преимущества привода на основе ШД – это отсутствие датчика обратной связи. Управление шаговым двигателем осуществляется от начального положения привода, в которое его надо переместить после включения питания устройства. Для этого используется датчик начального положения механизма – концевой выключатель. Он должен быть подключен к входу микроконтроллера, например, INT0. При включении питания двигатель осуществляет движение влево, пока не сработает концевой выключатель. Это означает, что механизм находится в начальном положении и микроконтроллер готов к приему команд управления.

Чтобы заставить двигатель вращаться, необходима последовательная коммутация обмоток в соответствии с табл. 3 – без деления шага и в соответствии с табл. 4 – с делением шага. Единица в таблице означает, что через данную обмотку протекает ток, ноль – тока в обмотке нет. Направление вращения двигателя определяется последовательностью коммутации обмоток согласно таблицам, например, сверху вниз – правое вращение, снизу вверх – левое. При составлении алгоритма и программы, рекомендуется использовать отдельные подпрограммы для движения двигателя влево и вправо на заданное число шагов.

Электрическая принципиальная схема устройства приведена на рис. 1. В качестве микроконтроллера здесь используется классический микроконтроллер Intel 8051 в корпусе с 40 выводами. Его можно заменить современным аналогом, например, AT89C51 или AT89S51, выпускаемым фирмой ATMEL. Для упрощения схемы используется внутренняя память программ. Коммутация обмоток шагового двигателя M1 осуществляется биполярными транзисторами VT1 – VT4, работающими в ключевом режиме. Они включены по схеме с общим эмиттером. Ток, проходящий через обмотки L1 – L4 шагового двигателя M1, не меняет своего направления, поэтому такой двигатель называется униполярным. Порт

P0 микроконтроллера имеет выходы с открытым коллектором, поэтому ток открытия транзисторов VT1 – VT4 формируется резисторами R4 – R7. Для закрытия транзисторов VT1 – VT4 открываются выходные транзисторы порта P0 микроконтроллера. В результате, для создания протекающего тока через обмотку шагового двигателя в соответствующий разряд порта микроконтроллера записывается единица и наоборот, для обесточивания обмотки – записывается ноль. Вместо биполярных коммутирующих транзисторов могут применяться МОП транзисторы, включенные по схеме с общим истоком. Диоды VD1 – VD4 защищают транзисторы VT1 – VT4 и микроконтроллер от ЭДС самоиндукции, которая возникает при выключении транзисторных ключей.

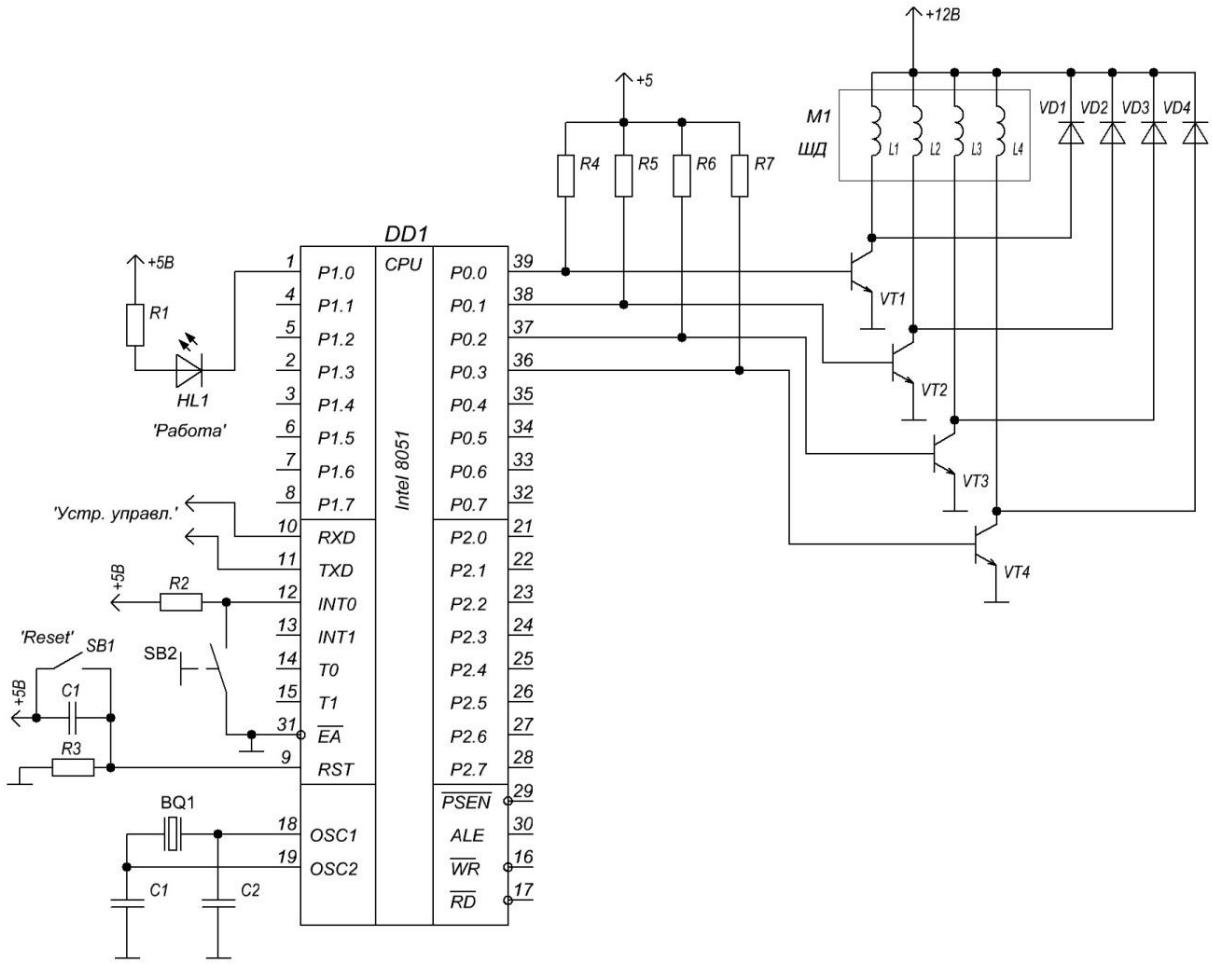

Рис. 1. Схема электрическая принципиальная устройства управления шаговым двигателем

Задание частоты синхронизации микроконтроллера происходит от внутреннего генератора, частота которого определяется внешним кварцевым резонатором BQ1. Для надежного запуска генератора используются конденсаторы C1 и C2 небольшой емкости (10 – 68 *пФ*). Для приведения всех внутренних регистров в начальное состояние при включении питания используется цепочка C1, R3. Для принудительного сброса микроконтроллера используется кнопка SB1. Для индикации работы двигателя применен светодиод HL1. Резистор R1 ограничивает ток через светодиод. Ток проходит через светодиод при записи в порт P1.0 логического нуля.

Устройство управления шаговым двигателем связано с устройством верхнего уровня по последовательному порту представленному линиями RxD и TxD. По линии RxD

производится прием информации, по линии TxD происходит передача данных. Фактически, для упрощения задания, задействована только линия RxD, которая принимает управляющий байт информации. Для задания начального положения механизма, приводимого шаговым двигателем, используется концевой выключатель SB2, срабатывающий в крайнем положении, например, крайнем левом положении, которое достигается при движении механизма влево. Крайнее левое положение индицируется появлением логического нуля на входе INT0 микроконтроллера.

В зависимости от варианта, возможна коммутация обмоток с делением и без деления шага. Алгоритм управления без деления шага более прост, но при этом движение вала шагового двигателя будет менее плавным, по сравнению с алгоритмом где присутствует деления шага.

Порядок коммутации обмоток без деления и с делением шага приведен в табл. 1 и 2 соответственно. При использовании алгоритма без деления шага последовательно коммутируются четыре обмотки, т.е. в любой момент времени протекает ток через одну обмотку двигателя.

При делении шага добавляются четыре дополнительных состояния, когда ток протекает одновременно в предыдущей и последующей обмотках. При этом переход от одного устойчивого состояния (шага) в другое происходит более плавно.

Таблица 1. Порядок коммутации обмоток шагового двигателя без деления шага

L1	L2	L3	L4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Таблица 2. Порядок коммутации обмоток шагового двигателя с делением шага

L1	L2	L3	L4
1	0	0	0
1	1	0	0
0	1	0	0
0	1	1	0
0	0	1	0
0	0	1	1
0	0	0	1
1	0	0	1

2. Порядок выполнения работы

Лабораторная работа рассчитана на 8 академических часов (2 лабораторных занятия).

Порядок выполнения работы следующий:

1. Получение задания в соответствии с номером варианта.
2. Изучение устройства микроконтроллера Intel 8051 и его системы команд.
3. Изучение принципиальной схемы устройства управления шаговым двигателем и ее коррекция (при необходимости) в соответствии со своим вариантом.

4. Разработка алгоритма работы программы.
5. Выполнение необходимых расчетов согласно данным варианта.
6. Ознакомление с отладчиком ассемблерных программ Pinnacle 52 или аналогичным. Выполнение пробных примеров, приведенных в разделе 2.3.
7. Составление и отладка программы на языке ассемблера.
8. Демонстрация преподавателю работы отлаженной программы в пошаговом режиме.
9. Составление отчета по лабораторной работе, включающего: данные варианта задания, принципиальную схему устройства и описание ее работы, расчеты значений управляющих регистров микроконтроллера, блок-схему алгоритма работы устройства, программный код на языке ассемблера с комментариями, подробную блок-схему алгоритма с указанием рабочих регистров микроконтроллера.

Таблица 3. Варианты заданий для выполнения лабораторной работы

Но-мер ва-риан-та	Частота кварцевого резонатора,МГц	Деление шага	Число шагов в секунду	Частота последовательного порта, бит/с	Число бит, передаваемых последователь-ным портом в асинхронном режиме
1	2	-	5	1200	10
2	**3**	**+**	**7**	**2400**	**11**
3	4	-	8	4800	10
4	5	+	9	9600	11
5	6	-	10	1200	10
6	8	+	11	2400	11
7	10	-	12	4800	10
8	11	+	13	9600	11
9	12	-	14	1200	10
10	2	+	15	2400	11
11	4	-	3	4800	10
12	5	+	4	9600	11
13	6	-	5	1200	10
14	8	+	6	2400	11
15	10	-	7	4800	10
16	11	+	8	9600	11
17	12	-	9	1200	10
18	2	+	10	2400	11
19	4	-	11	4800	10
20	5	+	12	9600	11
21	6	-	13	1200	10
22	8	+	14	2400	11
23	10	-	15	4800	10
24	11	+	14	9600	11
25	12	-	13	1200	10
26	2	+	12	2400	11

27	4	-	11	4800	10
28	5	+	10	9600	11
29	6	-	9	1200	10
30	8	+	8	2400	11
31	10	-	7	4800	10
32	11	+	6	9600	11
33	12	-	5	1200	10
34	2	+	4	2400	11
35	4	-	3	4800	10

3. Порядок работы с отладчиком ассемблерных программ Pinnacle 52

Для разработки и отладки программ для микроконтроллеров семейства mcs-51 существует множество программ для персональных компьютеров. Примером может служить программа Fd51, широко распространенная в начале 90-х годов. К ее недостаткам можно отнести интерфейс операционной среды DOS, требующий набора отдельных команд в командной строке для трансляции ассемблерной программы. Для удобства пользователя удобно применять программы отладчиков, работающие под операционной средой Windows. Одной из таких программ является Pinnacle 52.

Для запуска программы *Pinnacle 52 Professional Development System* щелкните на рабочем столе соответствующую иконку. После загрузки основного окна программы войдите в меню «Project» – «Project Options». В появившемся окне выберите тип микроконтроллера Standart 8051. С помощью меню «View» откройте следующие окна: «Registers», «Ports», «Timers», «Internal RAM».

Выбрав меню «File» – «New» откройте окно в котором предполагается ввод текста программы для отладки. Если имеется текстовый файл с программой, то воспользуйтесь меню «File» – «Open». Файл должен иметь расширение *asm*. Редактируемый файл можно сохранить, выбрав меню «File» – «Save». Для трансляции в машинный код отредактированного файла воспользуйтесь меню «Project» – «Compile&Link», либо нажмите сочетание клавиш «Ctrl+F2». Справочная информация по регистрам микроконтроллера находится в меню «Help» – «8051 Microcontroller». В меню «Help» – «Pinnacle Help» находится справочная информация как по языку программирования, так и по работе с программой.

Основным окном, необходимым для отладки программы является окно «Registers». Здесь отображаются данные хранящиеся в регистрах на текущий момент выполнения программы. Щелкнув два раза мышкой по выбранному регистру, открывается окно с обозначениями отдельных битов этого регистра. Пользователь может в процессе отладки программы менять значение каждого бита. Флаги регистра состояния программы PSW вынесены в отдельную область именуемую «Flags» для удобства отслеживания изменений в процессе отладки программы. В окне «Registers» также отображаются регистры, отвечающие за прерывания IE, IP, регистр управления мощностью потребления PCON, регистр управления и статуса последовательного порта SCON.

Окно «Ports» содержит побитное отображение состояния портов ввода-вывода. Данное окно разделено на две части: в левой части отображаются выходные данные, записанные в регистры-защелки, а в правой части отображаются сигналы, которые поступают на линии микроконтроллера. Пользователь может менять состояние выходных защелок и входных сигналов щелкнув мышью один раз на соответствующий бит.

Окно «Timers» отображает содержимое регистров управления и статуса таймер-счётчиков TCON и регистра режимов таймер-счётчиков TMOD. Щелкнув мышью по этим

регистрам, можно побитно редактировать в отдельном окне их состояние. Содержимое регистров счета TH0 и TL0, а также TH1 и TL1 отображается одновременно в шестнадцатеричном формате. Обозначенные в окне регистры T2CON, T2MOD, Timer2, RCAP2 предназначены для других модификаций микроконтроллеров. Открыв окно с описанием кода регистра можно узнать адрес этого регистра в памяти данных микроконтроллера.

Специальное окно с названием DPTR используется для отображения содержимого шестнадцатиразрядного регистра указателя внешних данных DPTR. Для обозначения состояния этого регистра выделено специальное окно из-за того, что не в каждой отлаживаемой программе используется внешняя память данных.

Особого внимания заслуживает окно Internal RAM, которое отображает внутреннюю память данных микроконтроллера (128 байт для 8051 и 256 байт для 8052). Зеленая область памяти микроконтроллера отображает выбранный в данный момент банк регистров общего назначения (РОН). Задав нужные значения флажков RS0 и RS1 можно выбрать нужный банк РОН. Четыре банка памяти РОН занимают область 00 – 1F. Регистр, помеченный красным цветом, обозначает элемент, который будет следующим помещен в стек. Адрес этого элемента является суммой содержимого регистра указателя стека плюс единица ((SP)+1).

Область, помеченная желтым цветом с адресами 20 – 2F представляет область памяти использующую побитовую адресацию. Для отображения побитного содержимого данной области может быть использовано отдельное окно «Bit Fields», которое можно открыть, выбрав меню «View» – «Bit Fields». Черным цветом выбрана память, которая недоступна для выбранного микроконтроллера.

После загрузки и трансляции программы можно закрыть окно с исходным текстом программы и открыть окно Code Window, выбрав соответственно меню: «View» – «Code Memory». Это окно показывает дизасемблированый текст программы, адрес выполняемой команды и код команды. Зеленой полосой отображается команда, которая будет выполняться следующей. Нажатием клавиши Goto в данном окне осуществляется переход к нужной области памяти программ, введя соответствующий адрес. Клавишей Set PC можно задать адрес той команды, которую необходимо выполнить. Клавишей Toggle устанавливаются точки останова в текущей программе во время ее отладки. Отладчик ассемблерных программ Pinnacle 52 имеет еще ряд дополнительных возможностей для отладки программ, с которыми можно ознакомиться в разделе помощи «Help» – «Pinnacle Help».

В качестве упражнения наберите текст следующей программы:

```
        org 0
        ljmp new
new:  mov a, #150
        mov b, #30
        div ab
        end
```

Эта программа записывает число 150 в аккумулятор, а в расширитель аккумулятора – число 30 (числа десятичные). В пятой строке эти числа делятся, и результат записывается в аккумулятор. Откомпилируйте программу, откройте окно дизассемблера и пошагово выполните ее. При этом наблюдайте изменения в регистрах a и b.

В качестве другого примера наберите текст более сложной программы:

```
      org 0
      ljmp m1
m1: mov r0, #01100010b
      mov r1, #01000011b
```

```
      mov a, r1
      cpl a
      add a, r0
      jnc m2
      inc a
m2:   mov r2, a
      end
```

Эта программа производит вычитание двух чисел, используя команду сложения, в обратном коде. Первое число больше нуля, второе – меньше нуля, результат – больше нуля. В третьей и четвертой строках вводятся числа в бинарном виде в регистры r0 и r1. В пятой и шестой строках происходит инвертирование разрядов второго числа. В седьмой строке происходит сложение чисел. В восьмой строке проверяется наличие переноса. Если есть перенос, то в девятой строке увеличивается содержимое аккумулятора на единицу. В десятой строке результат записывается в регистр r2.

В этих двух примерах вначале присутствует служебное слово org. Оно предписывает компилятору следующую команду записать по адресу, указанному после org. В данном случае эта команда ljmp перехода на начало программы.

4. Пример выполнения задания

4.1. Исходные данные и необходимые расчеты

Приведем пример выполнения задания на примере варианта 2 (табл. 5).

Исходные данные:
Частота кварцевого резонатора – 3 МГц;
Деление шага – присутствует;
Число шагов в секунду – 7;
Частота последовательного порта – 2400 бит/с;
Число бит передаваемых последовательным портом – 11.

В составе микроконтроллера имеется два таймер-счётчика Т/С0 и Т/С1. Назначим Т/С0 отсчитывать временные интервалы между шагами для задания скорости вращения вала двигателя, а Т/С1 для задания скорости передачи последовательного порта. При этом Т/С0 будет работать в режиме 1 (шестнадцатиразрядный счетный регистр), а Т/С1 – в режиме 2 (восьмиразрядный регистр с автозагрузкой).

Поскольку таймер-счётчики считают машинные циклы, то определим длительность машинного цикла исходя из заданной частоты кварцевого генератора:

$$T_{мц} = \frac{12}{f_Г} = \frac{12}{3 \cdot 10^6 \ Гц} = 4 \cdot 10^{-6} c = 4 \ мкс .$$

Поскольку в задании присутствует деление шага, то управление происходит по полушагам. Определим длительность полушага:

$$t_{п.ш.} = \frac{1}{2 \cdot 7} = 0{,}0714 \ c = 71{,}4 \ мс .$$

Количество счетов таймер-счётчика для задания полушага:

$$N_{сч.} = \frac{t_{п.ш.}}{T_{мц}} = \frac{0{,}0714 \ c}{4 \cdot 10^{-6} c} = 17850 .$$

Поскольку таймер-счётчик T/C0 считает от начального значения N_0 до максимального значения 65535 (FFFFh), то определим N_0:

$$N_0 = 65535 - N_{сч} = 65535 - 17850 = 47685.$$

Это значение надо каждый раз записывать в шестнадцатиразрядный счетный регистр таймер-счётчика T/C0, который представлен двумя восьмиразрядными регистрами TH0 и TL0.

Представим значение N_0 в шестнадцатеричной форме: $N_0 =$ BA45h. Из этого следует, что: TH0=BAh, TL0=45h или в двоичной форме TH0=10111010b, TL0=01000101b.

Эти значения надо каждый раз записывать в счетный регистр перед запуском таймера на счет для задания времени задержки очередного полушага.

Может получиться ситуация, когда частота кварцевого резонатора велика, а время шага (полушага) велико, при этом N_0 примет отрицательное значение. В этом случае необходимо несколько раз в программе запускать таймер-счётчик, при этом уменьшить расчетное время таймера в такое же количество раз.

Рассчитаем значение регистра TH1, которое обеспечивает работу T/C1 в режиме 2 (восьмиразрядный регистр с автозагрузкой). T/C1 обеспечивает скорость работы последовательного порта в режиме 3.

Согласно формулы расчета скорости последовательного порта в режимах 1 и 3:

$$f_{1,3} = \frac{\left(2^{SMOD}/32\right)\cdot\left(f_{рез.}/12\right)}{\left(256-\left(TH1\right)\right)}.$$

Согласно варианту $f_{1,3} = 2400$.

Бит удвоения скорости последовательного порта SMOD (регистр PCON) для определенности примем равным 0.

Тогда получим уравнение:

$$2400 = \frac{1/32 \cdot 3\cdot 10^6/12}{\left(256-\left(TH1\right)\right)}.$$

Откуда находим значение TH1:

$$TH1 = 256 - \frac{3\cdot 10^6}{32\cdot 12\cdot 2400} = 252{,}74 \approx 253.$$

4.2. Блок-схема алгоритма управления шаговым двигателем

Управление шаговым двигателем происходит согласно блок-схеме, приведенной на рис. 10. Вначале производится настройка управляющих регистров, которые задают режимы работы таймер-счётчиков (TMOD), последовательного порта (SCON), режимы прерываний (IE). Затем производится калибровка устройства путем установки механизма в начальное состояние. Для этого производится движение механизма влево на один шаг и проверяется срабатывание концевого выключателя. Если концевой выключатель не сработал, то цикл повторяется и двигатель движется влево на один шаг до тех пор, пока не сработает концевой выключатель. Срабатывание концевого выключателя означает нахождение механизма в начальном положении. Далее происходит зацикливание программы и ожидание приема байта от последовательного порта. Байт последовательного порта имеет следующую организацию: старший бит отвечает за направление движения, например, ноль – влево, единица – вправо, а остальные семь бит представляют двоичный код количества шагов, которое должен отработать шаговый двигатель. В результате шаговый двигатель за один раз может повернуться максимум на 127 шагов. Далее происходит определение направления движения и количества шагов, которое должен сделать шаговый двигатель. Затем осуществляется само движение влево или вправо в зависимости от заданного направления на N шагов. Затем программа переходит в режим ожидания приема очередного байта от последовательного порта. В результате программа зацикливается. Блоки движение влево на N шагов и движение вправо на N шагов целесообразно выполнить в виде отдельных подпрограмм и передавать в них в качестве параметра количество шагов N.

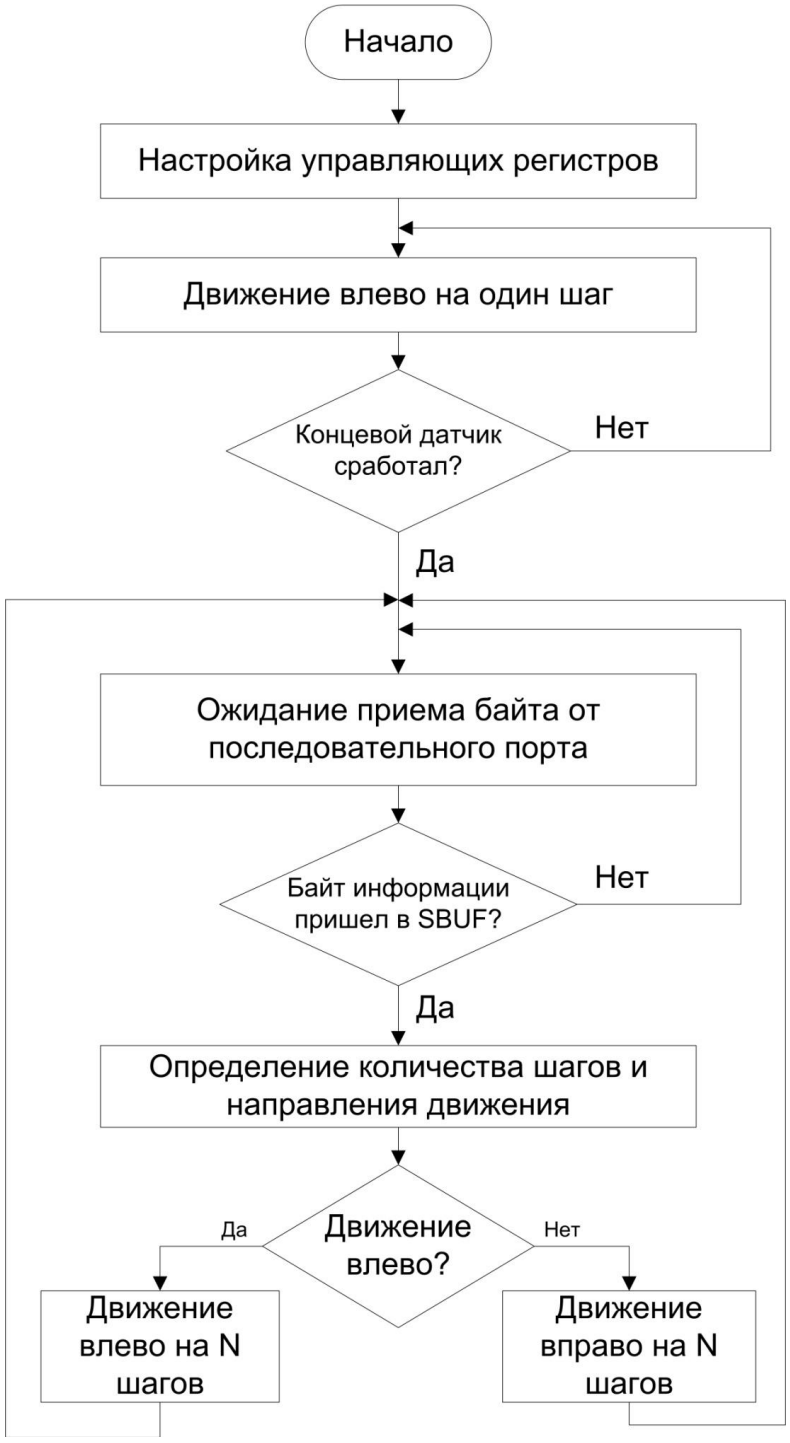

Рис. 2. Блок-схема алгоритма управления шаговым двигателем

4.3. Подпрограммы движения влево и вправо

Для обеспечения движения шагового двигателя необходимо осуществлять коммутацию его обмоток согласно табл. 3, 4. В нашем случае коммутация осуществляется согласно табл. 4, т.е. с делением шага. Переход от одного состояния (одной строки таблицы) к другому происходит при одном полушаге. Т.е. поворот на один шаг означает смену двух состояний, указанных в таблице. При этом, смена состояний в таблице сверху вниз означает одно направление движения, для определенности назовем его левое, а изменение состояний снизу вверх – другое направление движения – правое.

Эти состояния необходимо записать в память данных, чтобы в дальнейшем их оттуда извлекать. Это потребуется сделать вначале программы с использованием следующих команд:

```
mov 30h, #00001000b
mov 31h, #00001100b
mov 32h, #00000100b
mov 33h, #00000110b
mov 34h, #00000010b
mov 35h, #00000011b
mov 36h, #00000001b
mov 37h, #00001001b
```

В качестве области памяти данных выбрана область начиная с адреса 30h, поскольку с этого адреса начинается область байтовой адресации.

Приведем непосредственно текст подпрограммы движения влево:

```
left:   mov p1, @r0
        dec r0
        mov th0, #10111010b
        mov tl0, #01000101b
        setb tr0
m1:     jnb tf0, m1
        clr tf0
        cjne r0, #2fh, m2
        mov r0, 37h
m2:     djnz r1, left
        ret
```

В этой подпрограмме регистр r1 содержит количество полушагов, которое должен сделать шаговый двигатель. После выполнения подпрограммы r1 содержит ноль. Т.е. подпрограмма строится на основе цикла, организованного командой djnz. В первой строке подпрограммы в порт p1 выводится строка состояния обмоток шагового двигателя из памяти данных, на которую указывает указатель r0. Фактически r0 перебирает адреса ячеек памяти данных, в которых содержатся состояния обмоток шагового двигателя. Таким образом, для нашего варианта r0 может принимать значения от 30h до 37h.

Строка 2 осуществляет уменьшение содержимого указателя r0 на единицу.

В строках 3 и 4 прописываются значения счетного регистра T/C0, которые были получены ранее. В строке 5 происходит запуск T/C0. Строка 6 содержит пустой цикл, до тех пор, пока T/C0 не завершит свою работу. Окончанием цикла служит установка флага tf0 в единицу. В седьмой строке флаг tf0 сбрасывается. В строке 8 проверяется, не вышел ли указатель r0 за нижний предел 30h, если вышел, то ему присваивается максимальное значение 37h, которое в дальнейшем будет уменьшаться.

Подпрограмма движения вправо строится аналогично:

```
right:  mov p1, @r0
        inc r0
        mov th0, #10111010b
        mov tl0, #01000101b
        setb tr0
m3:     jnb tf0, m3
```

```
        clr tf0
        cjne r0, #38h, m4
        mov r0, 30h
m4:     djnz r1, left
        ret
```

Отличия от подпрограммы движения влево проявляются лишь в том, что во второй строке происходит увеличение указателя на единицу, и проверка указателя происходит по максимальному значению 37h, если значение стало 38h, то ему присваивается начальное значение 30h.

4.4. Настройки микроконтроллера

Приведенные выше подпрограммы движения влево и вправо помещаются в конце программы. Вначале программы необходимо реализовать последовательность действий согласно алгоритму на рис. 10.

Сформируем управляющий код регистра настройки таймер-счётчиков TMOD.

TMOD.7	TMOD.6	TMOD.5	TMOD.4	TMOD.3	TMOD.2	TMOD.1	TMOD.0
GATE 1	**$C/\overline{T1}$**	**M11**	**M10**	**GATE0**	**$C/\overline{T0}$**	**M01**	**M00**

GATE0, GATE1=0 (не требуется управление блокировкой от входов INT0, INT1);
$C/\overline{T0}$ =0, $C/\overline{T1}$ =0 (работа устройств в качестве таймеров);
M01=0, M00=1 (режим 1 T/C0);
M11=1, M10=0 (режим 2 T/C1).
В результате требуется выполнить следующую команду:
mov tmod, #00100001b.

Сформируем управляющий код регистра управления таймер-счётчиков TCON.

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**

TF0=0, TF1=0 (биты завершения работы принимают начальные нулевые значения);
TR0=0, TR1=0 (не требуется запуск таймер-счётчиков);
IE0=0, IE1=0 (начальная установка флагов внешнего прерывания);
IT0=0, IT1=0 (установка внешних прерываний по уровню).

В результате требуется выполнить следующую команду:
mov tcon, #00000000b

Сформируем управляющий код регистра управления/статуса последовательного порта.

SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

SM0=1, SM1=1 (режим 3 работы последовательного порта);
SM2=0 (многопроцессорный режим отсутствует);
TB8=0, RB8=0 (8-ой передаваемый и принимаемый биты);
TI=0, RI=0 (флаги прерываний от передатчика и приемника).
В результате требуется выполнить следующую команду:

mov scon, #11000000b.

Сформируем управляющий код регистра управления мощностью.

PCON.7	PCON.6	PCON.5	PCON.4	PCON.3	PCON.2	PCON.1	PCON.0
SMOD	-	-	-	**GF1**	**GF0**	**PD**	**IDL**

SMOD=0 (согласно расчетам скорости последовательного порта);
GF0=0, GF1=0 (биты пользователя, не используются);
PD=0, IDL=0 (энергосберегающие режимы, не используются).
В результате требуется выполнить следующую команду:
mov pcon, #00000000b.
Обратите внимание, что из-за аппаратных особенностей микроконтроллера к регистру PCON нельзя обращаться побитно. Для изменения значения бита в программе требуется записать байт полностью.

Сформируем управляющий код регистра масок прерываний.

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	**-**	**-**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**

EA=0, ES=0, ET1=0, EX1=0, ET0=0, EX0=0 (прерывания в программе не используются).
В результате необходимо выполнить следующую команду:
mov ie, #00000000b.
Сформируем управляющий код регистра приоритетов прерываний.

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
-	**-**	**-**	**PS**	**PT1**	**PX1**	**PT0**	**PX0**

PS=0, PT1=0, PX1=0, PT0=0, PX0=0 (приоритеты установлены низкие, прерывания в программе не используются).
В результате необходимо выполнить следующую команду:
mov ip, #00000000b

Выше были рассмотрены управляющие регистры, которые настраивают и управляют работой всех внутренних устройств микроконтроллера. Следует заметить, что если управляющий код регистров принимает значение ноль, то необязательно его прописывать в начальной установке программы, поскольку все внутренние регистры автоматически устанавливаются в ноль после сброса микроконтроллера после включения питания.

4.5. Вывод механизма в начальное положение

После настройки управляющих регистров следует выполнить вывод механизма в начальное положение. Для этого двигатель делает один шаг влево и проверяется наличие срабатывания концевого датчика SB2 (рис. 9). Цикл проверки длится до тех пор, пока датчик не сработает (рис. 10). Это можно реализовать с помощью следующих команд:

```
m3: mov r1, #02
    lcall left
    jb p3.2, m3
```

В первой строке в регистр r1 записывается количество полушагов – два полушага, что соответствует одному шагу. Во второй строке вызывается подпрограмма движения влево. В третьей строке если второй бит порта 3 установлен (на рис. 9 обозначен INT0), то переход на метку m3, иначе выполняется следующая команда.

Далее согласно блок-схеме алгоритма на рис. 10 требуется принять управляющий бит, определить направление движения и количество шагов (полушагов) и вызвать подпрограмму движения влево или вправо. Соответствующий фрагмент программы будет иметь следующий вид:

```
      setb ren
      setb tr1
m4: jnb ri, m4
      clr ri
      mov a, sbuf
      anl a, #01111111b
      rl a
      mov r1, a
      mov a, sbuf
      anl a, #10000000b
      jz m5
      lcall right
      ljmp m4
m5: lcall left
      ljmp m4
```

В первой строке устанавливается единичное значение бита REN. Этим разрешается работа приемника последовательного порта. При нулевом значении этого бита прием не осуществляется. В следующей строке устанавливается бит TR1, который запускает работу таймер-счётчика 1, осуществляющего тактирование последовательного порта. В третьей строке происходит зацикливание микроконтроллера до тех пор, пока не будет установлен бит RI в единицу, означающий, что приемник последовательного порта завершил прием байта. После приема байта в четвертой строке бит RI сбрасывается. В строках 5, 6 из принятого бита в SBUF выделяется информация о количестве шагов (младшие семь бит) и помещается в аккумулятор. В строках 7, 8 количество шагов умножается на два путем сдвига содержимого аккумулятора влево, и результат помещается в регистр R1. Это необходимо, поскольку управление происходит с делением шага. Затем, в строках 9, 10 в аккумулятор записывается старший бит принятого байта. Он отвечает за направления движения. В строке 11 происходит ветвление: если в аккумуляторе ноль – то переход по метке m5 на строку14 и вызов подпрограммы движения left, если в аккумуляторе не ноль – то вызов подпрограмм движения вправо right в строке 12. После завершения движений происходит безусловный переход по метке m4, где снова ожидается приход байта в последовательный порт.

4.6. Программа управления шаговым двигателем

Далее полностью составим программу управления шаговым двигателем и отладим ее в отладчике:

```
        org 0
        ljmp start
start:  mov tmod, #00100001b
```

```
        mov tcon, #00000000b
        mov scon, #11000000b
        mov pcon, #00000000b
        mov ie, #00000000b
        mov ip, #00000000b
m3:     mov r1, #02
        lcall left
        jb p3.2, m3
        setb ren
        setb tr1
m4:     jnb ri, m4
        clr ri
        mov a, sbuf
        anl a, #01111111b
        rl a
        mov r1, a
        mov a, sbuf
        anl a, #10000000b
        jz m5
        lcall right
        ljmp m4
m5:     lcall left
        ljmp m4
left:   clr p1.0
        mov p1, @r0
        dec r0
        mov th0, #10111010b
        mov tl0, #01000101b
        setb tr0
m1:     jnb tf0, m1
        clr tf0
        cjne r0, #2fh, m2
        mov r0, 37h
m2:     djnz r1, left
        setb p1.0
        ret
right:  clr p1.0
        mov p1, @r0
        inc r0
        mov th0, #10111010b
        mov tl0, #01000101b
        setb tr0
m3:     jnb tf0, m3
        clr tf0
        cjne r0, #38h, m4
        mov r0, 30h
m4:     djnz r1, left
        setb p1.0
        ret
        end
```

Обратите внимание на то, что программы на ассемблере всегда начинаются со служебного слова org и заканчиваются служебным словом end. Кроме того, в программном коде появились строки clr p1.0 и setb p1.0, которые включают и выключают индикацию движения двигателя в подпрограммах left и right.