How Stepper Motors Work

Stepper motors consist of a permanent magnet rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Figure 1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

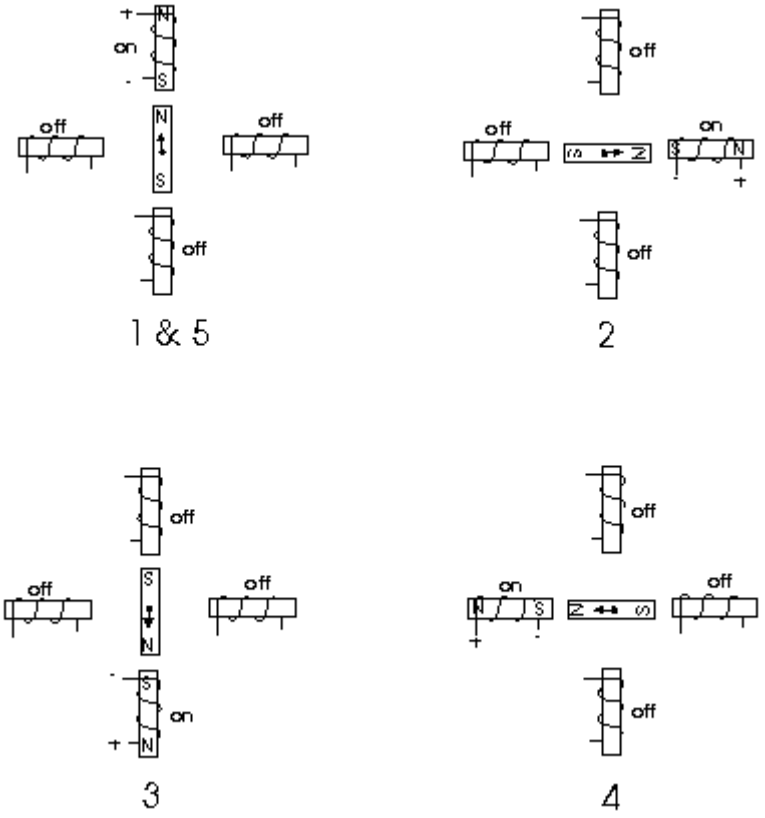


Figure 1

In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360 degree rotation.

You may double the resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half

stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in Figure 2, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top magnet is deactivated and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.

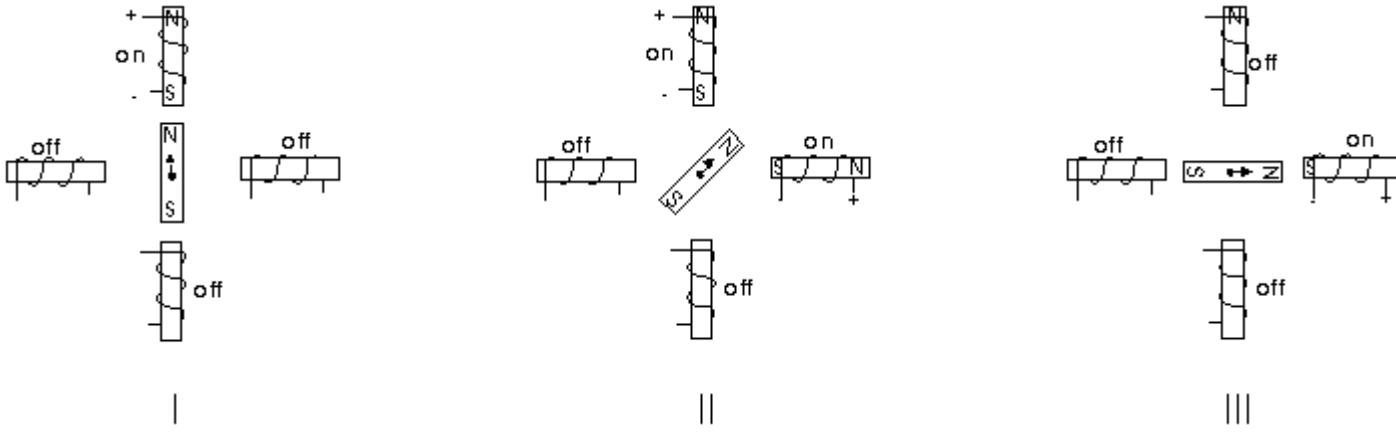


Figure 2

There are several types of stepper motors. 4-wire stepper motors contain only two electromagnets, however the operation is more complicated than those with three or four magnets, because the driving circuit must be able to reverse the current after each step. For our purposes, we will be using a 6-wire motor.

Unlike our example motors which rotated 90 degrees per step, real-world motors employ a series of mini-poles on the stator and rotor to increase resolution. Although this may seem to add more complexity to the process of driving the motors, the operation is identical to the simple 90 degree motor we used in our example. An example of a multipole motor can be seen in Figure 3. In position 1, the north pole of the rotor's perminant magnet is aligned with the south pole of the stator's electromagnet. Note that multiple positions are alligned at once. In position 2, the upper electromagnet is deactivated and the next one to its immediate left is activated, causing the rotor to rotate a precise amount of degrees. In this example, after eight steps the sequence repeats.

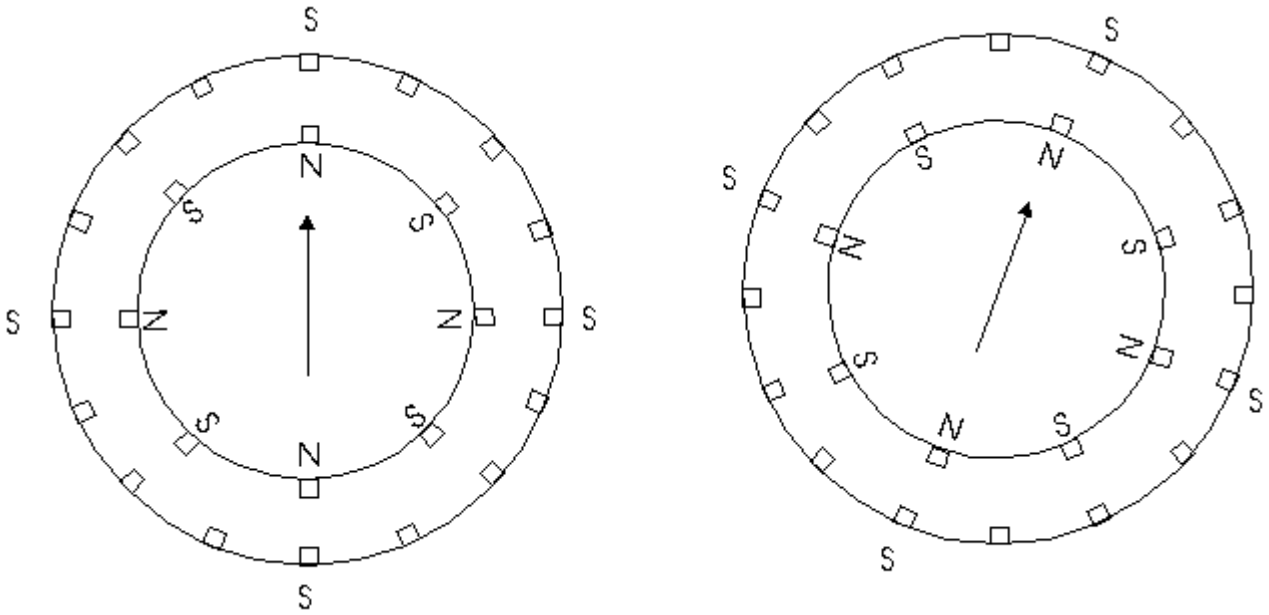


Figure 3

The specific stepper motor we are using for our experiments (ST-02: 5VDC, 5 degrees per step) has 6 wires coming out of the casing. If we follow Figure 5, the electrical equivalent of the stepper motor, we can see that 3 wires go to each half of the coils, and that the coil windings are connected in pairs. This is true for all four-phase stepper motors.

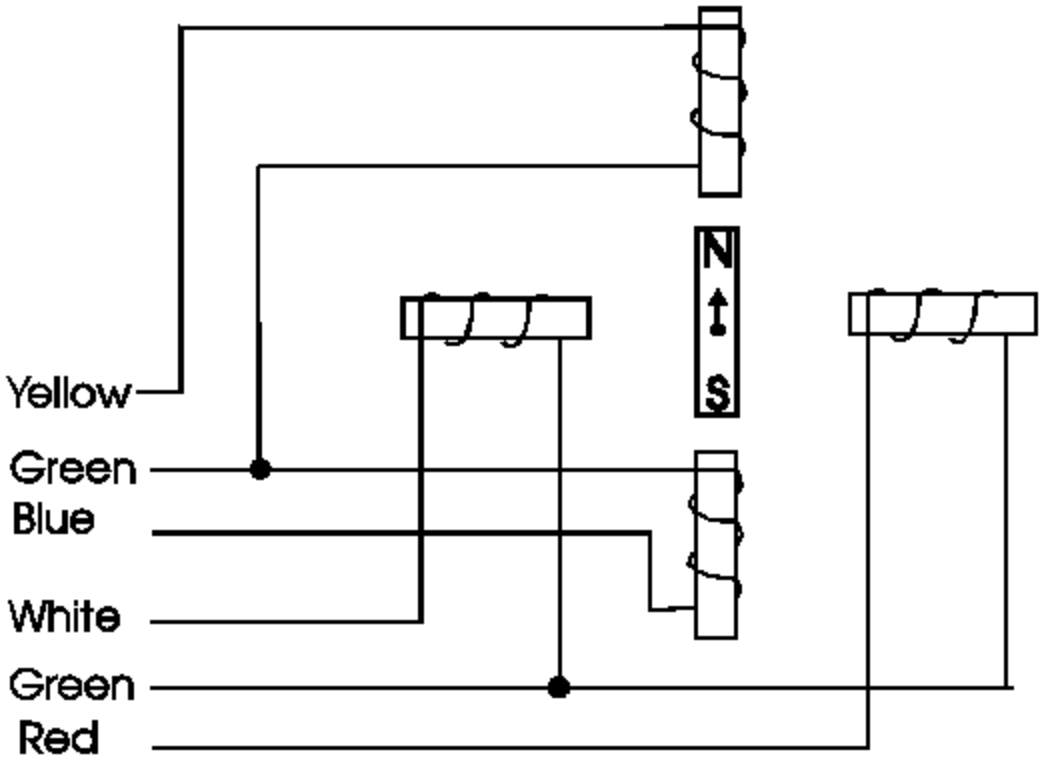


Figure 5

However, if you do not have an equivalent diagram for the motor you want to use, you can make a resistance chart to decipher the mystery connections. There is a 13 ohm reistance between the center-tap wire and each end lead, and 26 ohms between the two end leads. Wires originating from seperate coils are not connected, and therefore would not read on the ohm meter.

First Stepper Circuit

Figure 4 is the schematic of our first test circuit. The PIC's output lines are first buffered by a 4050 hex buffer chip, and are then connected to an NPN transistor. The transistor used, TIP120, is actually a NPN Darlington (it is shown as a standard NPN). The TIP120's act like switches, activating one stepper motor coil at a time.

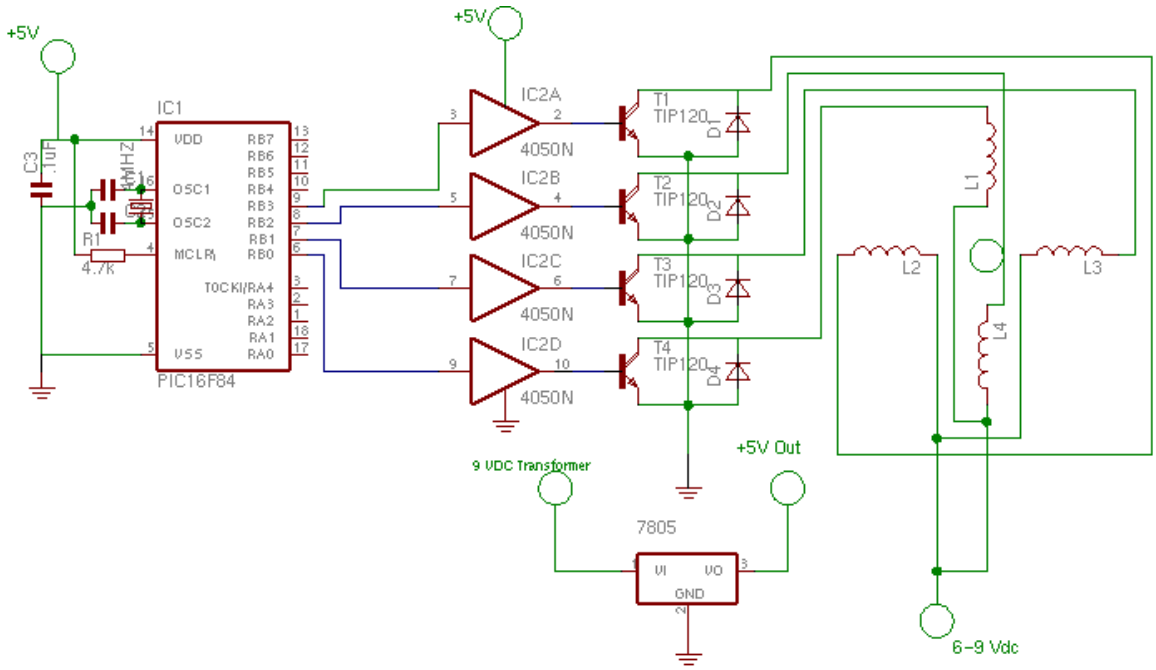


Figure 4

Due to a inductive surge created when a coil is toggled, a standard 1N4001 diode is usually placed across each transistor as shown in the figure, providing a safe way of dispersing the reverse current without damaging the transistor. Sometimes called a snubbing diode. **The TIP120 transistors do not need an external snubbing diode becasue they have a built in diode.** So the diodes shown in the drawing are the internal diodes in the TIP120 transistors.

The simplest way to operate a stepper motor with a PIC is with the full step pattern shown in Table 1. Each part of the sequence turns on only one transistor at a time, one after the other. After the sequence is completed, it repeats infinitly until power is removed.

Q1	Q2	Q3	Q4
+	-	-	-
-	+	-	-
-	-	+	-
-	-	-	+

Table 1

I purposly made this first program as small as possible, simply to demonstrate how easy it is to control a stepper motor. Also note the use of high and low commands to control the

output lines, rather than peek and poke routines. For our purposes, high and low are sufficent.

Listing 1

```
' First stepper motor controller program
' Rotates motor at set speed forever

Symbol delay = B0
delay = 25

loop:
high 0
pause delay
low 0
high 1
pause delay
low 1
high 2
pause delay
low 2
high 3
pause delay
low 3
goto loop

' use b0 as the delay variable
' set the delay to 25 ms


' turn on Q1
' wait 25 ms
' turn off Q1
' turn on Q2
' wait 25 ms
' turn off Q2
' turn on Q3
' wait 25 ms
' turn off Q3
' turn on Q4
' wait 25 ms
' turn off Q4
' forever
```

Listing 1

Second Basic Program

The first program was fine for demonstrating how to control a stepper motor with a PIC Micro, but was severely limited to the one mode of operation preprogrammed into it. In this next program, we will make the program controllable by external user input in the form of four switches: SW1 will incrementaly increase the delay variable, thereby slowing the rotation of the motor. SW2 does the opposite. SW3 halts all operation while the switch is pressed. SW4 reverses direction of the motor (CW to CCW or vice-versa) while it is closed (pressed).

listing 2

```
' Second stepper motor controller program
' Responds to user input by changing speed or direction

Symbol delay = B0
delay = 100

forward:
high 0
pause delay
low 0
high 1
pause delay
low 1
high 2
pause delay
low 2
high 3
pause delay
low 3
goto check

reverse:
high 3
pause delay
low 3
high 2
pause delay
low 2
high 1
pause delay
low 1
```

```
high 0
pause delay
low 0
goto check

check:
if pin4 = 0 then timeup
if pin5 = 0 then timedn
if pin6 = 0 then halt
if pin7 = 0 then reverse
goto forward

timeup:
delay = delay + 5
pause 50
if delay > 250 then max
if pin4 = 0 then timeup
goto check

timedn:
delay = delay - 5
pause 50
if delay <20 then min
if pin5 = 0 then timedn
goto check

halt:
if pin6 = 0 then halt
goto check

max:
delay = 245
goto check

min:
delay = 25
goto check

' use b0 as the delay variable
' delay will begin at 100 ms


' turn on Q1
' wait for delay ms
' turn off Q1
' turn on Q2
```

```
' wait for delay ms
' turn off Q2
' turn on Q3
' wait for delay ms
' turn off Q3
' turn on Q4
' wait for delay ms
' turn off Q4
' check the status of switches


' turn on Q4
' wait for delay ms
' turn off Q4
' turn on Q3
' wait for delay ms
' turn off Q3
' turn on Q2
' wait for delay ms
' turn off Q2
' turn on Q1
' wait for delay ms
' turn off Q1
' check the status of switches




' if nothing pressed, continue


' increase delay by 5 ms
' pause for 50 ms


' check switches again


' decrease delay by 5 ms
' pause for 50 ms


' check switches again
```

'check switches again

' cap the delay at 245 ms
' check switches again

' limit the delay to 25 ms
' go back and check switches

End Listing 2

As you can see by looking at the schematic (Figure 7), the "switches" used in this circuit are pieces of jumper wire on a breadboard. The inputs (B0-B3) are kept high via Vcc through a 10K resistor. Theses switches are activated by grounding them via another piece of jumper wire.

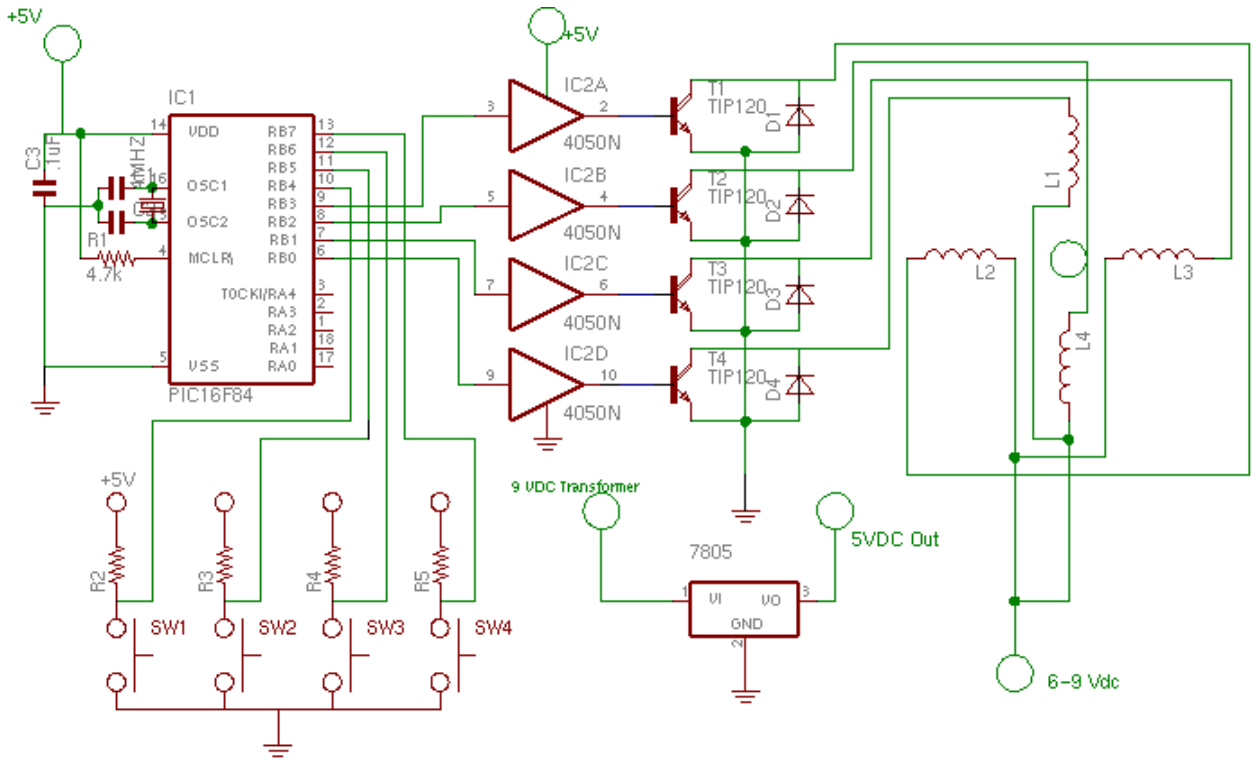


Figure 7

A note on the previous two listings: the minimum delay used in the programs was 25 ms. Depending on the clock speed of your crystal, 25 ms may be either too fast or too slow for the rotor to move. If you are having problems getting these programs to work, this is a likelyHalf Stepping

As previously stated, half-stepping doubles the resolution of the motor. In our case, we are using a motor with a 5 degree / per step resolution, requiring 72 steps to complete one rotation. By half stepping, we could double this to 2.5 degrees / pulse, requiring 144 steps to complete one rotation. Table 2 shows the step pattern.

Step	Q1	Q2	Q3	Q4
1	+	-	-	-
2	+	+	-	-
3	-	+	-	-
4	-	+	+	-
5	-	-	+	-
6	-	-	+	+
7	-	-	-	+
8	+	-	-	+

Table 2

There are two ways in which you could impliment half-stepping in a PIC. One way would be to drive it directly, using the two previous listings, but replacing the high/low commands with the pins that corrispond to each step. The other (and easier) way would be to use a UCN 5804 Stepper Motor IC in conjunction with the PIC.

culprit (further troubleshooting information).

The UCN 5804 Stepper Motor IC

By using an external stepper motor controller, such as the UCN 5804, you can simplify your programs and control as many motors as you have outputs via an array of UCN 5804's. Not only does it allow for the control of more motors, but more importantly, it simplifies the process. You now only have to output the pulse of your desired speed. Additionaly, you can switch between full and half stepping in real time via a switch on the UCN 5804 (or you may have the PIC control it), as well as reverse direction. A pinout for the UCN804 is shown in Figure 8.

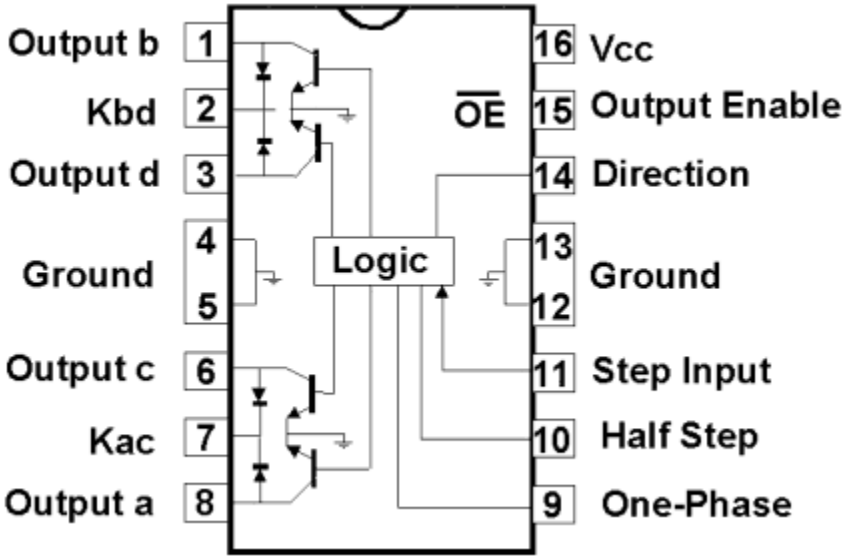


Figure 8

The schematic we will build to use this chip is shown in Figure 9. Since we are using a 5V stepper motor, we will be powering the UCN 5804 with a 9V wall transformer. You cannot use 6V, due to the draw of the motor. The UCN 5804 can support voltages up to 35V.

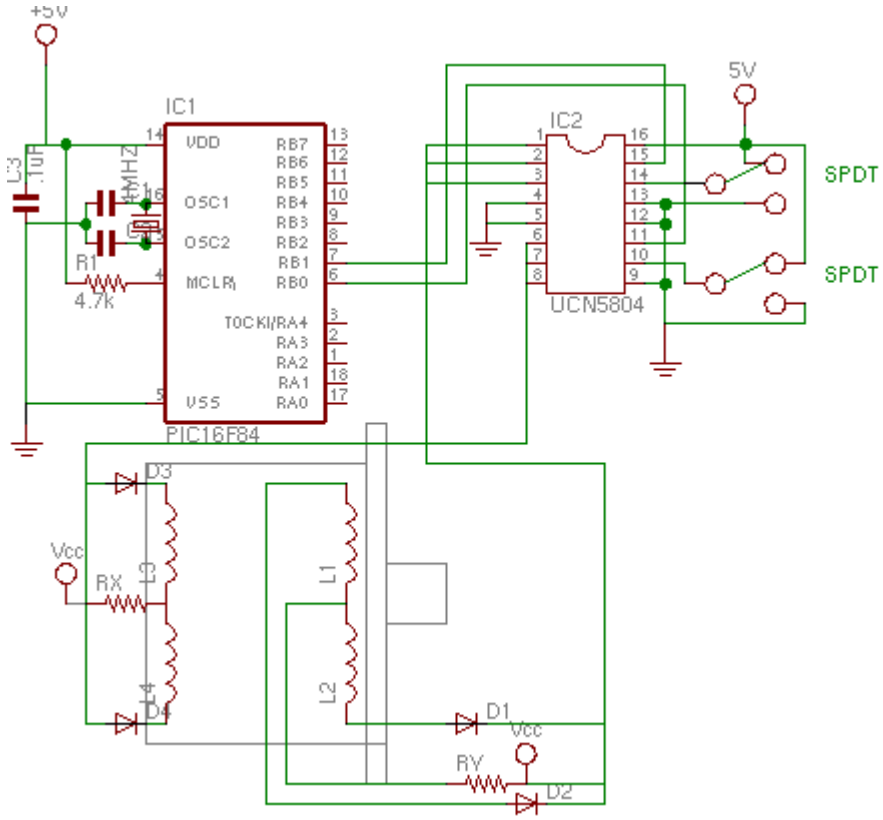


Figure 9

Notice in the schematic that two resistors, rx and ry, do not have an assigned value. This is because our motor draws only 100 mA, well under the chip's supported 1,250 mA. However, if you were to use a motor that draws the maximum or above, then you would need to use rx and ry to get the amperage under 1,250 mA. For example, a 24V motor with a phase resistance of 15 ohms would draw 1,600 mA (24/15 = 1.6). In this case, you should use _at least_ a 5 ohm resistor for both rx and ry, which would bring the current down to 1.2 A.

Since the input of the UCN 5804 is CMOS and TTL compatable, we can connect the outputs from the PIC directly into the UCN 5804. Two outputs are needed; one to control the step input (pin 11), and one to control the output enable (pin 15) which enables the stepper motor while high and disables the stepper motor while low.

Pins 9 and 10 control the stepping method (half or full steps). Pin 14 controls the step direction. Both of these controls can be manipulated by the PIC, but it is easier to control them directly through the use of jumpers acting as switches.

All the program has to do, is output a pulse and set the output enable low. This can be accomplished with a very simple program such as the following:

Listing 3

```
' Stepper motor control with a UCN 5804

Symbol delay = B0
low 1
delay = 10
delay = delay * 1000

loop: pulsout 0,delay
goto loop

' make a delay variable
' set the output enable low
' set a pulsewidth of 10 ms
' turn delay into microseconds

' send pulsewidth of delay to UCN5804
' repeat forever
```

End Listing 3

Notice that I added an extra step; taking the defined "delay" value and multiplying it by 1,000. This is neccisary because the pulsout command requires a pulsewidth in micro-seconds, not milli-seconds. You can make the code even smaller by removing lines 1, 3, and 4 and replacing "delay" in line 5 with a set number in microseconds. However, I prefer to the method shown in the listing, because it makes it easier to change the delay paramater in the more familiar milliseconds without having to convert it to microseconds.

Speed Control via Potentiometers

The PIC Basic language has a command that allows you to read the value of potentiometers on a given scale. This is the "pot" command, and is used in the following manner:

pot pin,scale,var

Where pin is the pin the potentiometer is connected to (pins B0-7), scale is the maximum value the command will return (in this case 245, being the maximum amount of ms we want in between pulses), and var is the variable the value will be placed in.

Using this command, we can determine the value of a potentiometer (we will be using a 50K ohm pot) and then use it to control the pulsewidth outputted to the UCN 5804. A schematic is shown in Figure 10.

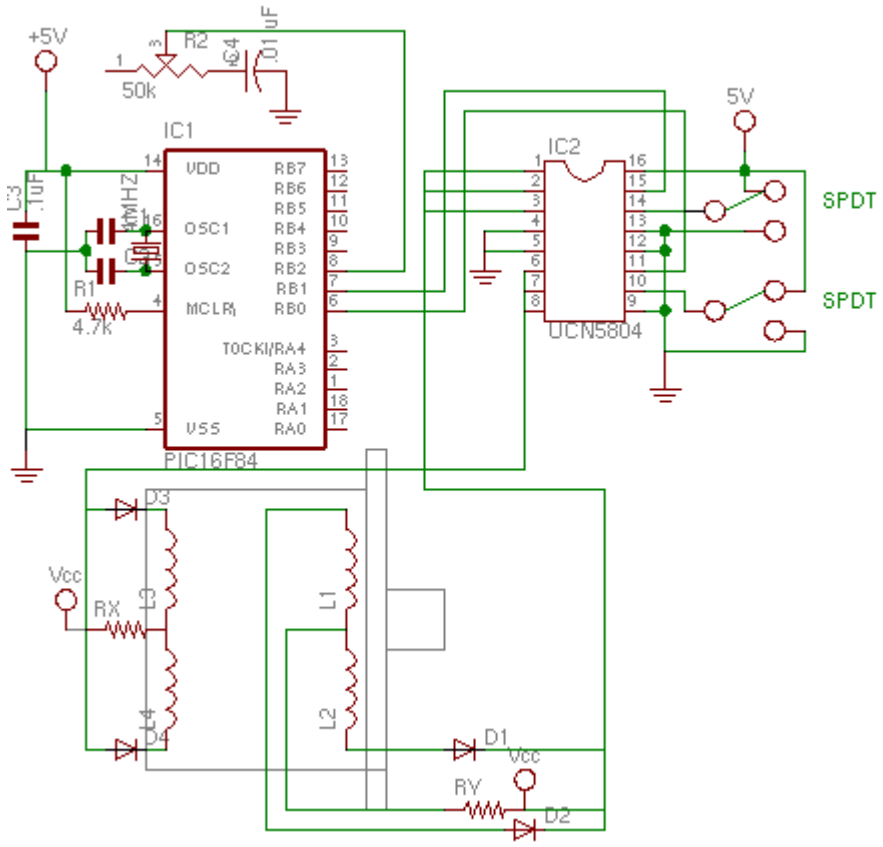


Figure 10

The program is shown in listing 4. First, it reads the value of pin B2 (the potentiometer), then places it on a scale of 245. That value is then multiplied by 1,000 (converting it into microseconds) and set to the UCN 5804.

Listing 4

```
' Controlling a stepper motor with the UCN 5804 and a potentiometer

Symbol delay = B0
low 1

start:
pot 2,245,delay
if delay < 25 min
goto turn
```

```
turn:
delay = delay * 1000
pulsout 0,delay
goto start

min:
delay = 25
goto turn

' make a delay variable
' set the output enable low


' read pot at pin B2
' set min delay to 25
' if delay is over 25, start moving


' turn delay into microseconds
' send to UCN 5804
' go back to start


' bottom out the delay at 25
' start the motor
```

End Listing 4

Not quite as complicated as you may have expected (thanks to the use of the UCN 5804). Notice again, the minimum delay set at 25 ms. If you would like to make it a lower number (thereby making the motor turn faster), simply change all occurances of "25" to the number desired.

Controlling Stepper Motors with a PIC Microcontroller

If your motor does not move at all when the program is active, then the most likely problem is that the diodes are facing the wrong direction. Check the schematic for proper wiring.

If the motor is moving, but does so only sporadicaly or jiggles back and forth, then there can be a number of possible causes:

1. There is not enough power going to the motor. This is often the case when using batteries which cannot support the drain of the motor. It is reccomended that you use a wall transformer to power the motor in combination with a 7805 voltage regulator to power the circuitry.

2. You are not using TIP 120 transistors, or a variation that is not able to support the load of the stepper motor.

3. The stepper motor is not wired to the correct transistor. Check the coil resistance with an ohm meter and rewire properly. For motors purchased from us, the pinouts are available here.

4. The pulse frequency is higher than that which the motor can react to, causing a malfunction. The programs in this article use the delay variable to control pulse frequency. Try increasing the value to decrease pulse frequency.

Stepper Motors:

Most of the motors we are familiar with spin when provided with electricity. The small DC motors found in toys have two wires coming out of them-- and batteries have two sides (+ and -), so it's pretty easy to figure out how to hook them up. Connect the battery to the motor one way, and it spins clockwise. Reverse the connection and you get counter clockwise. We can certainly use our parallel port bit wires to control the electricity feeding a DC motor, but because the motor spins freely, it is difficult to precisely control exactly how far it spins. If we want the motor to turn *exactly* a certain amount, we either need to measure how far the motor has turned, and feed this information back to the computer so it can make corrections in the controlling signal (this is how servo motors work), or we need a different kind of motor. And here is where *stepper* motors come in! **Steppers don't spin, they step**-- meaning their shaft rotates a certain amount with each command received from the controlling electronics (stepper motor driver). Steppers can do two things-- take a step clockwise, or take a step counter clockwise. Because steppers break rotation into discrete steps, we can think of them as digitizing motion, with each step being a motion pixel. This is ideal for connecting to the computer.

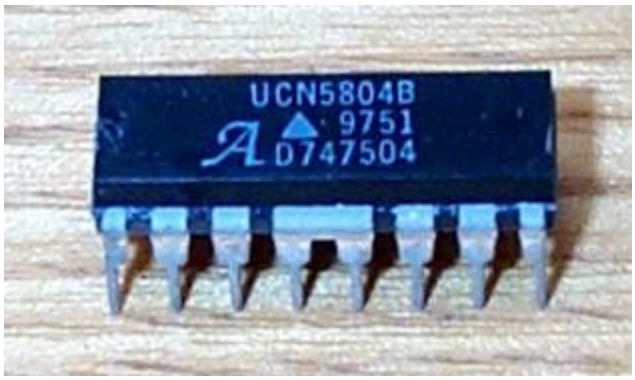
The down-side to steppers is that they are more complicated to hook up than simple DC motors. Steppers have 4,5,6 or 8 wires coming out of them! Electricity must be fed to just the right wires, in just the right amount and at just the right time for the motor to take a step. Fortunately this is a common need, and there are inexpensive electronic devices (integrated circuits or IC's) that are made just for this task. The job of the **stepper motor driver** is to convert two bits of information (we will use two of our bit wires again) into the proper pulses of electrical flow that will make the stepper motor work.

And what do these two bits do?
The first is **DIRECTION**-- this bit determines which direction (CW or CCW) steps will go.
The second is **STEP**-- when the driver receives this bit, the motor takes a step in the direction set by the **DIRECTION** bit.

Stepper motors are in lots of commonly used computer equipment-- they move the paper (up and down) and the print head (across) in your printer. They move the read/write head across the surface of your hard and floppy disk drives. Because of this, they are relatively easy to find as junk! Which is good, because when they are brand new *they can be very expensive*.

Steppers come in a wide range of sizes-- generally the bigger they are, the more turning force (torque) they have. The ones we will be experimenting with are pretty small, and similar to those you might find if you took apart an old floppy drive. They are new, but were cheap because they were surplus. Our stepper motors have 200 steps/revolution (1.8 degrees per step) which is a very common value-- but other motors may have fewer (or sometimes more) steps/rev.

If you want to know more about steppers and how to run them, there is a wealth of information on the Web. I have listed some of my favorite sites in the reference section.



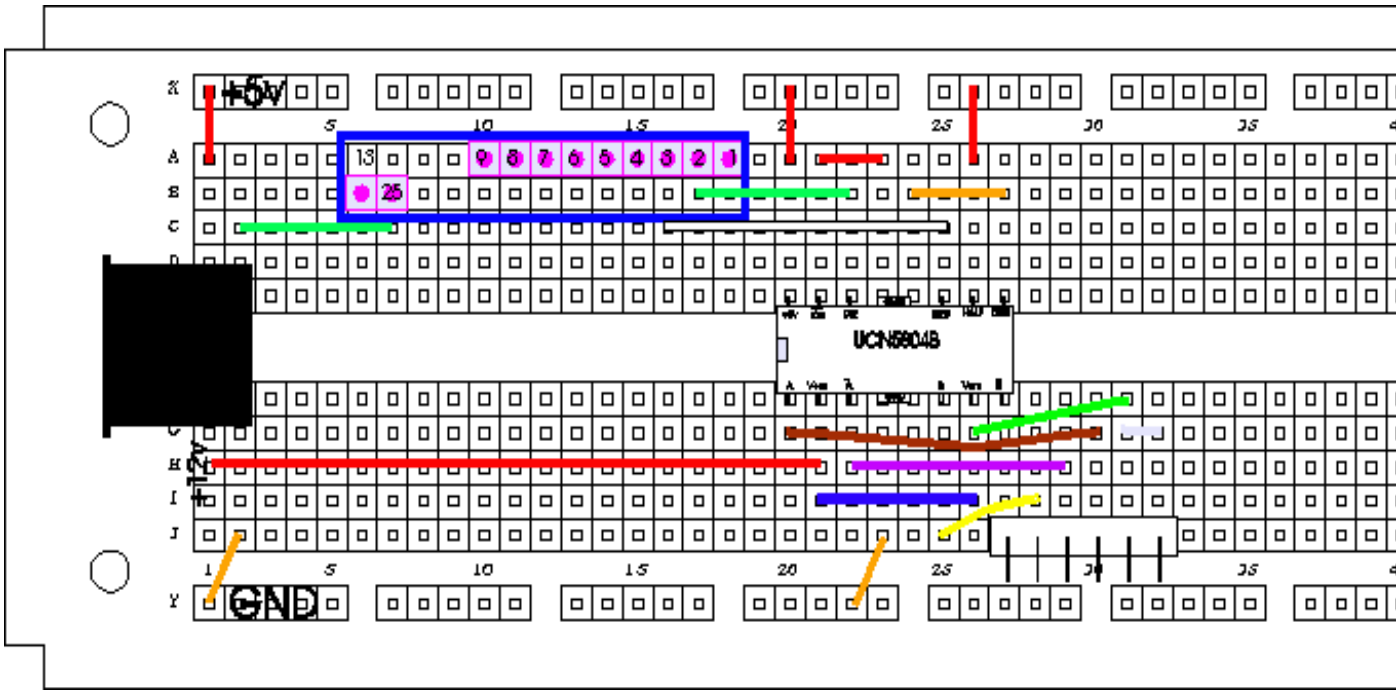
Stepper motor drivers also come in a wide variety of types and sizes. It is not that hard to build your own drive electronics from commonly available parts, but for small motors (like ours) it is far easier (and just as cheap) to use a ready-made IC. We will be using the UCN5804 chip. It combines my favorite two features: cheap and easy!

OK, enough talk-- let's do it...

Journey of a 1000 miles begins with a single step:

We will start by setting up our breadboard, similar to the way we used it with the Light-byte. We will leave the 2+9 pins which mate with our 26-hole connector in exactly the same place. But now we will be adding some new stuff-- in particular, we will be using power that does not come from the parallel port itself. It is important to remember that the bit wire electricity is extremely low power-- enough to light an LED, but certainly not enough to run a motor! In order to decrease the likelihood of making a mistake and damaging our computers, I have found some surplus power supplies and we will be using mating connectors that ensure that power gets hooked up in the right places. I like **P**ower **S**upplies because I hate changing batteries. I found the PS we will use as surplus-- it has just the voltages (5 and 12) that we need, and can handle the amount of current we will use (more about this later). **It is possible to damage the control computer if you are not careful with where these voltages are connected!** Fortunately, it is easy with the breadboard setup to prevent this. But, as an extra precaution, **we will ALWAYS check our circuit BEFORE connection to the computer**.

Here is the breadboard map of our single stepper motor driver circuit:

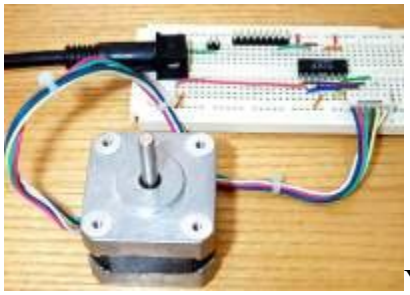


And here is a picture of the breadboard up to this point.

Placement of the PS connector is crucial-- the four pins must go into the first and second column of holes (E1,E2,F1,F2). This will ensure that 5+ is on the A1-D1 column, GND is on both the A2-D2 and F2-J2 columns, and 12+ is on the F1-J1 column. We will put a drop of Superglue on the bottom of this connector so that it does not come out every time we plug the PS into it.

After carefully checking to make sure that we have all the parts and jumpers properly in place, we will connect our stepper to the 6-prong connector (at J27-J32). Make sure the connection is made so that the motor's white and yellow leads are connected to the right side (J31,J32). Turn the motor shaft-- it should feel no different than before you made the connection.

Plug the PS's AC cord into a wall outlet, and then insert the PS's plug into the PS connector that has been glued to the breadboard. It only goes in one way (the arrow will be visible on top). Now try to turn the motor's shaft again-- **it should be stuck tight!** You may be able to overpower it into turning (this causes no harm), but there is an obvious difference. If the shaft turns easily, then something is wrong. Don't feel bad-- I rarely put together a circuit which works the first time! Disconnect the PS plug and carefully double check every connection. You *will* find the error if you are patient.


You will notice that after a short while your motor will get very warm. This is normal for stepper motors, but may surprise you if you don't know this.

OK-- if everything checks out (the motor shaft is stuck), we are now ready to connect to the computer. But first, it is a good idea to disconnect out PS from the breadboard, in case we put the 26-hole connector on wrong. So, after removing power from the breadboard, connect the 26-hole connector to the 2+9 pins (just as with the Light-byte). Now reconnect to the PS. Again the motor will be frozen in place.

Now we need to write a program to send step and direction bits to our motor:

Start QBASIC, and type (in the program window):

```
OUT 888,2
OUT 888,0
```

Put your finger on the motor shaft and then push Sh-F5-- *you have just taken your first step!*
Yes, it is rather small-- do it a couple times to convince yourself that the shaft is really moving.

Now, let's use a loop to make lots of steps (steps1.bas):

```
 CLS
INPUT "enter count value for speed control: ", speed
FOR counter = 1 TO 400

OUT 888,2
OUT 888,0
FOR zz = 1 TO speed: NEXT

NEXT counter
```

Run the program (Sh-F5) and try a count value of 500. You should see the shaft take 1 complete rotation. Try it a few times, and pay careful attention to the starting and final position of the flat spot on the shaft (they should be exactly the same). Try different count values-- the lower, the faster it goes (remember, the computer must count up to this number in between every step). You will find a value that is too low-- the motor just can't step that fast. This causes no harm to the motor or the computer, but of course this "misstepping" causes position errors and we will want to avoid it in the future when we are building robots.

How can we make it go in the other direction? Think about that direction bit. It is assigned to bit #0 (STEP is on bit #1). In our program, we have left it off the whole time-- *OUT 888,2* can be thought of as *OUT 888, 0+2* . But what if we try *OUT 888,1+2* -- go ahead, modify the program and run it. The shaft should complete 1 rotation in the opposite direction now.

Finally, let's do both directions, 10 times (steps2.bas):

```
CLS
INPUT "enter count value for speed control: ", speed

FOR repetition = 1 to 10

FOR counter = 1 TO 400
OUT 888,2
OUT 888,0
FOR zz = 1 TO speed: NEXT
NEXT counter

FOR counter = 1 TO 400
```

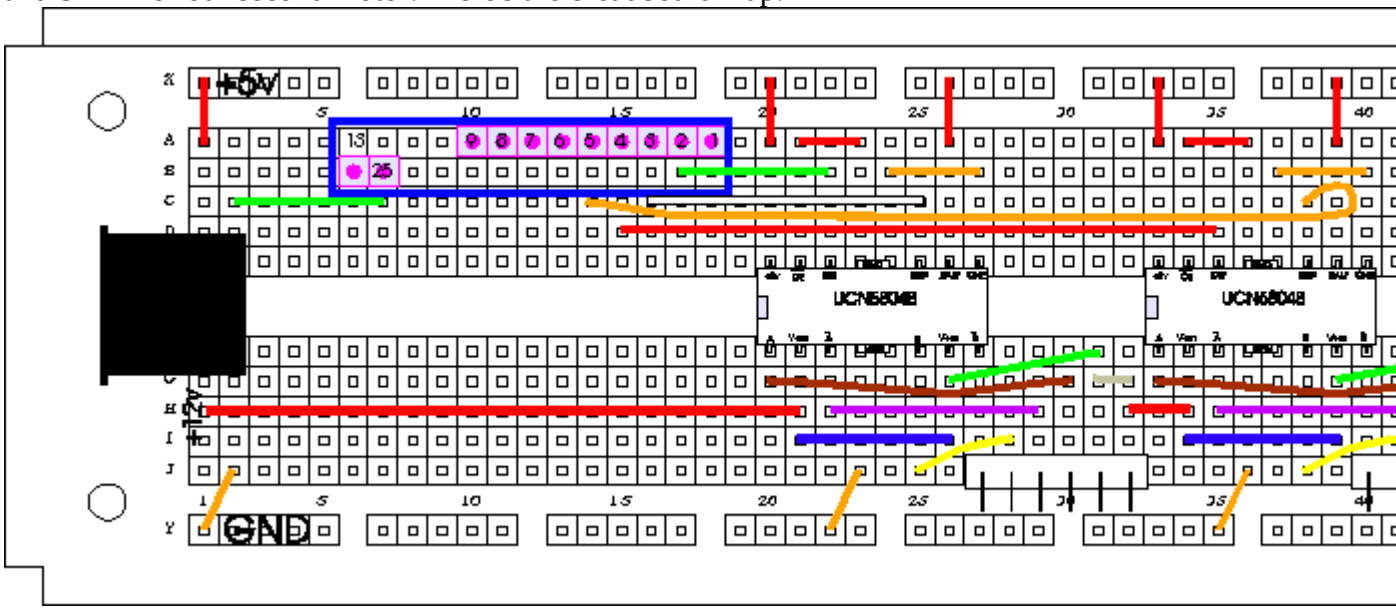
OUT 888,3
OUT 888,0
FOR zz = 1 TO speed: NEXT
NEXT counter

NEXT repetition

Notice that (if you haven't chosen a counter value too low) the shaft position at the start and end of the run is the same. This is because we have asked the motor to make full rotations only. But if you look at the program you will notice that our program commanded the motor to take 400 steps (not 200). The reason is that we are running in "half-step" mode-- an option of the 5804b chip. Although our motor is designed to take a step of 1.8 degrees (200 steps/rev), this applies only when it is run in "full-step" mode. The 5804b chip can run in this mode, but I always choose half-step because it is smoother (results in smaller motion pixels). Even though we will actually be taking half-steps, I will refer to these simply as steps.

Dual Stepping:

From controlling a single stepper (using two of our bit wires), to controlling two steppers is as easy as adding another chunk of driver circuitry to our breadboard, and connecting it to two more bits-- bits 0 and 1 are taken, so we'll use #2 and #3 to control DIRECTION and STEP for our second motor. Here's the breadboard map:



Notice that all we've done is to add an identical copy of the UCN5804 chip and its jumpers, except for the connection to bits 2 and 3 (wire #4 and #5 of the parallel port cable).

Pic of breadboard up to this point. (I have added an LED + resistor across +5 and GND as a "power on" indicator)

Just as with the single motor circuit, attach a motor to the new circuitry's 6-prong connector and then apply power. As before, the shaft should be frozen (disconnect power and troubleshoot if this is not so). If the new circuit checks out, attach your other motor to the first 6-prong connector. Shaft frozen? Good.

Disconnect power. Attach to the parallel port via the 26-hole connector (carefully), and reapply power.

Now let's modify steps2.bas to move our second motor:

```
 CLS
INPUT "enter count value for speed control: ", speed

FOR repetition = 1 to 10

FOR counter = 1 TO 400
OUT 888,8
OUT 888,0
FOR zz = 1 TO speed: NEXT
NEXT counter

FOR counter = 1 TO 400
OUT 888,12
OUT 888,0
FOR zz = 1 TO speed: NEXT
NEXT counter

NEXT repetition
```

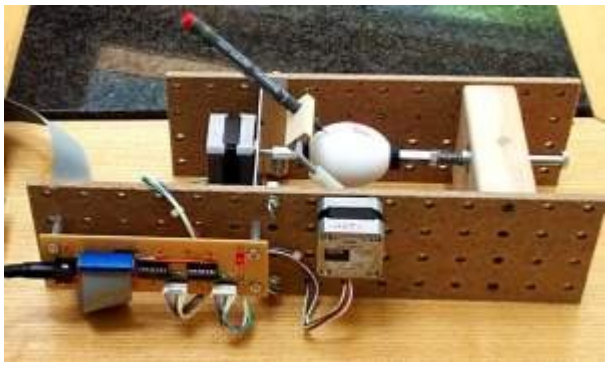
Run it-- motor #2 should now be swiveling merrily back and forth! But what about motor #1-- can we have them both moving at the same time? Sure-- it's as simple as addition: change *OUT 888, 8* to *OUT 888, 2+8*, and *OUT 888, 12* to *OUT 888, 3+12*. Running this new modification should result in both motors moving identically. Look at what we're doing-- simply controlling DIRECTION and STEP bits, no different than any other bit-controlled device. Just like with the Light-byte, we can have any combination

of DIR and STP bits we want. And we have room to add *two more* stepper drivers and motors!

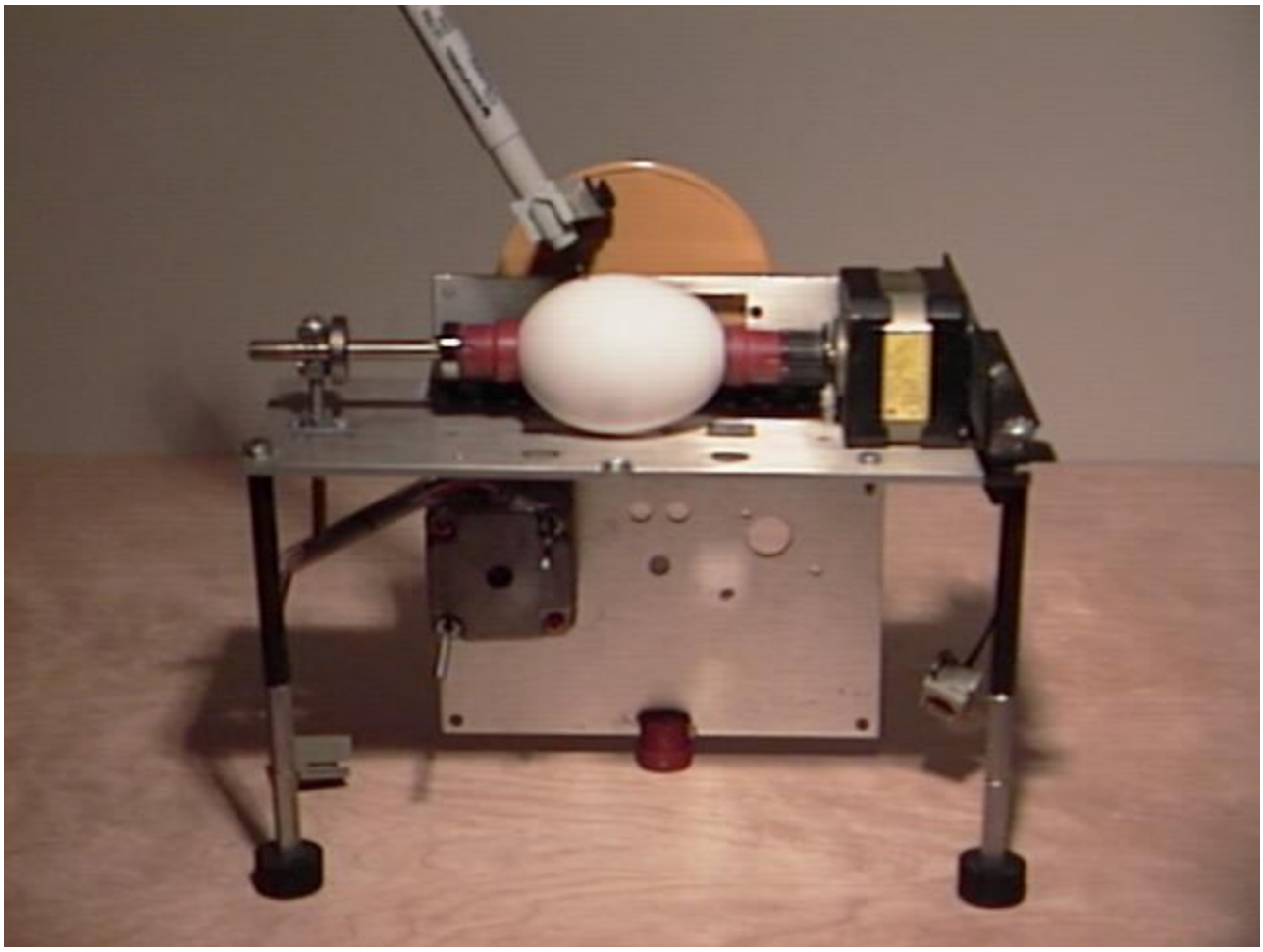
Egg-bot:

About 10 years ago I was asking myself what I could do with the two stepper motors that I had learned to control. No matter how excited ***I*** was over the potential of being able to precisely control motion, others remained relatively unimpressed at the sight of two motor shafts swiveling back and forth. Easter was coming, and my kids were talking about egg-coloring-- suddenly I had an application! My first motion-control device was built from junk parts and a few stolen pieces from my kids' toy chest.

We will be building essentially the same device, with a slightly improved design.



To begin, lets look at our assortment of parts:



In April of 1990 I had just learned how to connect two stepper motors to an old PC and was pondering an application. Easter was approaching, and my kids' discussion of egg coloring prompted an idea: construction of an "egg-plotter" using the two motors in an X and Y configuration-- one to roll the egg along its long axis, the other to move a pen from "pole to pole." My construction skills were pretty shabby, but I managed to throw it together in an afternoon. After writing a simple BASIC routine, I sat back and observed the creation of my first piece of robotic art.

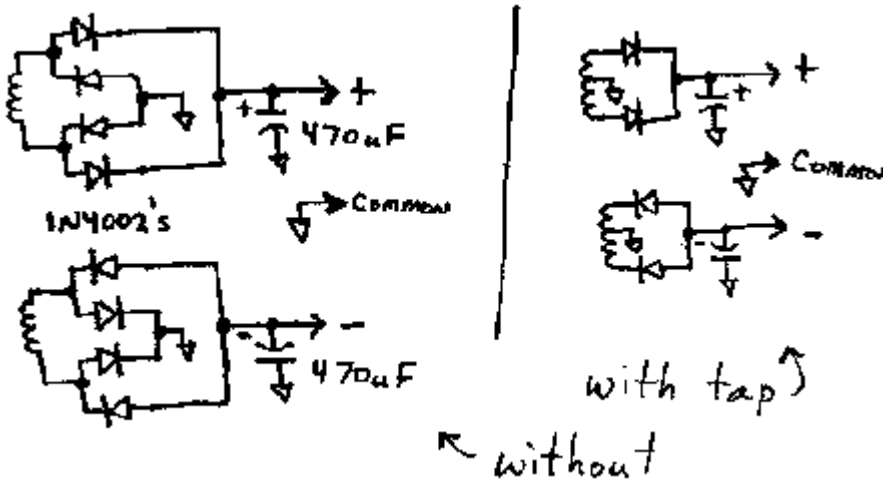
Stepper Motor Experiments

Stepper motors are most commonly controlled by microprocessors or custom controller ICs and the current is often switched by stepper motor driver ICs or power transistors. Precise motion is possible but the complexity usually lands the hobbyist's stepper motors in the "maybe someday" parts bin. But steppers may be used for a variety of applications without complex circuitry or programming. At first glance the stepper motor looks

a bit intimidating since there are at least four wires and often there are six. Most steppers have two independent windings and some are center-tapped, hence the four or six wires. A quick ohmmeter check will determine which wires belong together and the center-tap may be identified by measuring the resistance between the wires; the center-tap will measure 1/2 the total winding resistance to either end of the coil. Tie the wires that belong together in a knot and tie another knot in the center-tap wire for easy identification later. Stepper motors have become quite abundant and are available in all shapes and sizes from many surplus dealers. Experimenters can also salvage excellent steppers from old office and computer equipment.

Steppers move in small increments usually indicated on the label in degrees. To make a stepper motor spin in one direction current is passed through one winding, then the other, then through the first winding with the opposite polarity, then the second with flipped polarity, too. This sequence is repeated for continuous rotation. The direction of rotation depends upon which winding is the "leader" and which is the "follower". The rotation will reverse if either winding is reversed. The center-tapped versions simplify the reversal of current since the center-tap may be tied to Vcc and each end of the coil may be alternately pulled to ground. Non-tapped motors require a bipolar drive voltage or a bit more switching circuitry. If current is applied to both windings, the stepper will settle between two steps (this is often called a "half-step"). Taking the half-step idea to the extreme, one could apply two quadrature sinewaves to the windings and get very smooth rotation. This technique would not be particularly efficient since the controller would be dissipating at least as much power as the motor but, if smooth motion is required, it might be worth a try! Or, for those who don't mind complexity, the sinewaves could be efficiently approximated by using variable duty-cycle pulses. But the purpose here is to get those motors out of the junk box, not to think of more reasons to leave them alone! So here are some simple things to try.

Steppers make excellent low power generators and surprisingly efficient low power motors for low RPM applications. As a starting point, try connecting the windings of two steppers together. Pick steppers that turn freely so that internal friction doesn't spoil the experiment. When you spin one motor shaft, the other will follow. Admittedly, there is little torque. But it does illustrate that steppers may be used to generate electricity. Here are a couple of sketches showing how to connect stepper motors as generators:



The AC windings are not directly connected since the voltages are 90 degrees out of phase and the resulting voltage would be somewhat lower. Dual isolated outputs are possible if the grounds for each winding are not connected together. Here are a few application ideas:

- Add a hand crank and bulb for kids' science demonstration. A 24 volt stepper will light a 120 volt nightlight or Christmas tree light.
- Make a wind-powered anemometer.
- Generate a higher voltage by turning a high voltage stepper with a low voltage motor.
- Run various gadgets including radios, calculators, multimeters, flashers, garage door openers, remote controls, LED flashlights, bike flashers.
- Achieve extreme isolation - use a long plastic shaft to turn the generator for high voltage isolation or use a line-powered motor to spin a stepper for very high line isolation.

A large capacitor may be charged to run low current devices for some time after a few quick spins of the crank. A voltage regulator may be required for some devices but use an efficient series regulator to conserve power if the affair is hand-powered.

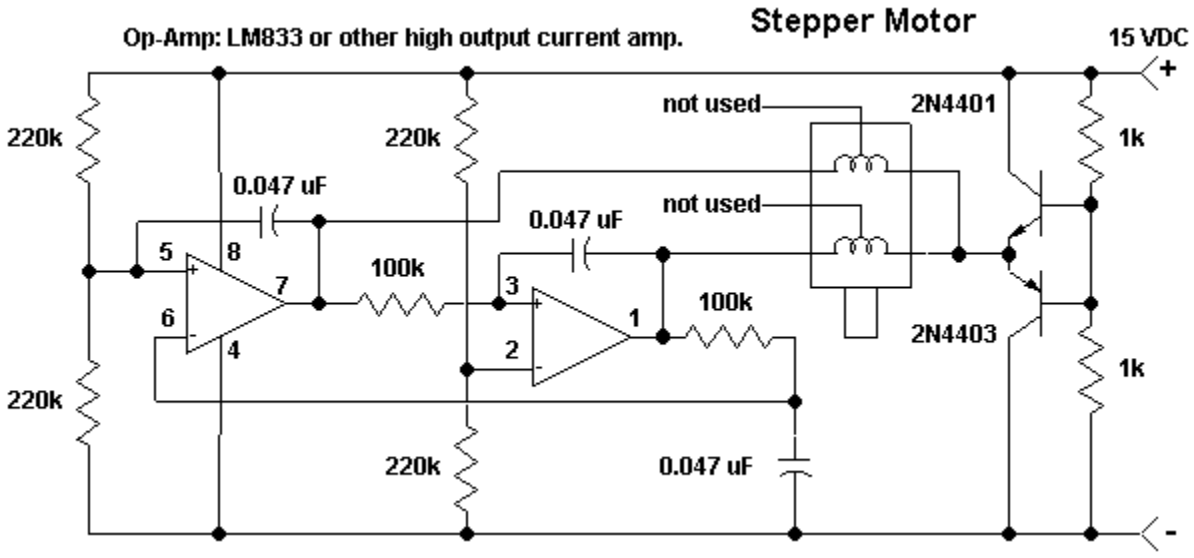
Many steppers make excellent low power motors despite the markings on the label. For example, a stepper rated at 16.8 volts, 280 mA consumed only 20 mA when unloaded and driven with bipolar 5 volt squarewaves. Such low currents may be directly extracted from many op-amps and logic devices without any additional drivers! Obviously the mechanical load must be light and the speed will be low or the motor will stall but many useful applications exhibit little load. Here are some examples:

- Use vanes to mechanically chop light, electrostatic fields or other

radiation.

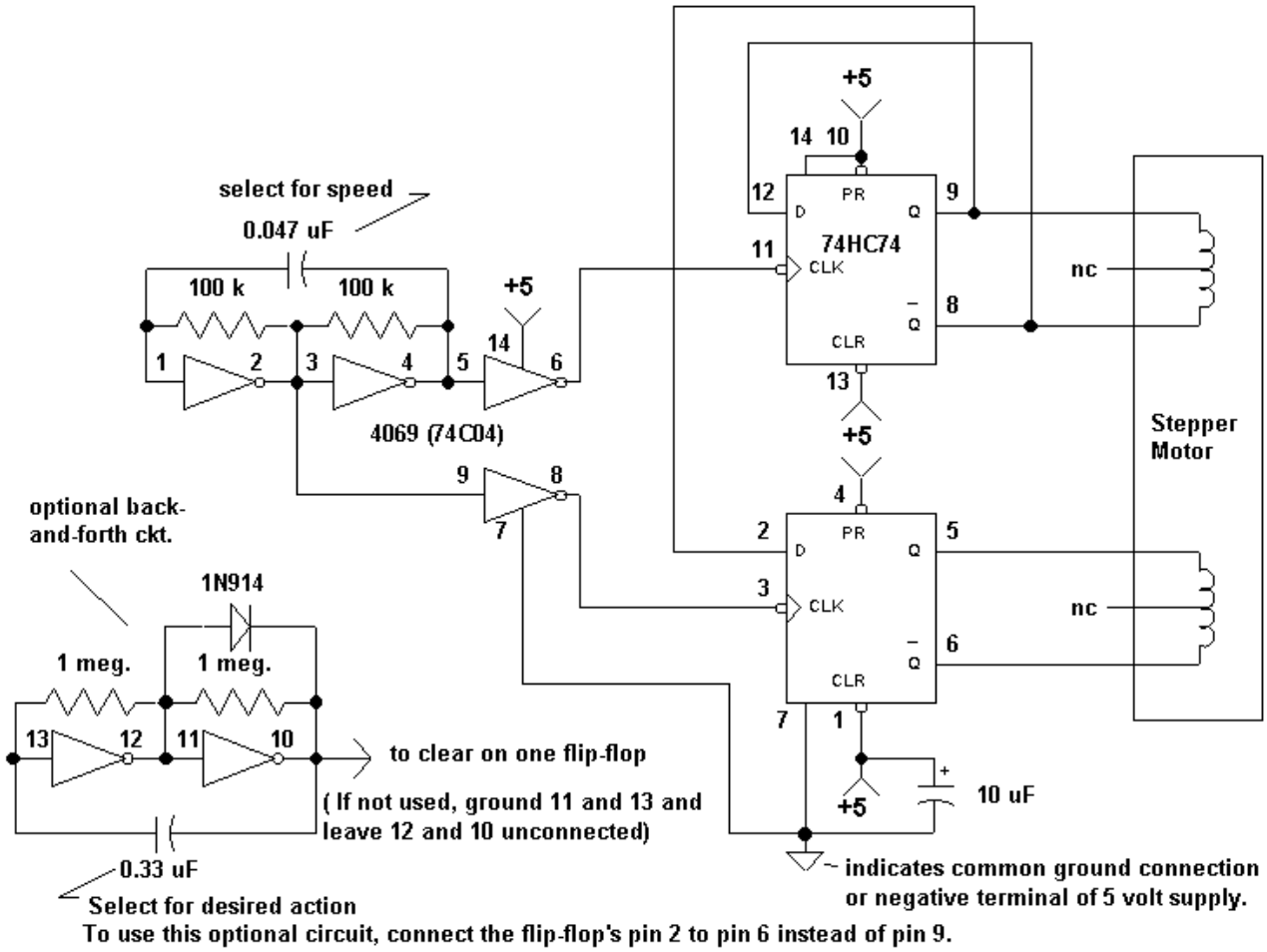
- Spin a sign or attention-getting symbol on an advertising display.
- Spin a toy radar antenna or make a highly efficient electric toy car or train.
- Make a spinning Christmas tree ornament.
- Make a laborator stirrer.

The following circuit is commonly called a quadrature oscillator or a sine - cosine oscillator and may be built using almost any op-amp with reasonably high output current. An LM358 actually worked in this circuit, but only barely! A better choice would be an LM833 or any one of many higher current op-amps.



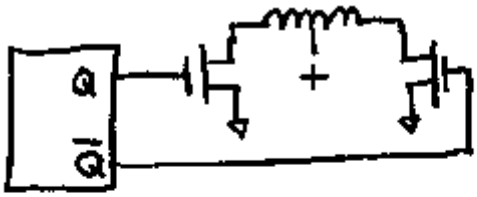
The two transistors generate Vcc/2 so that the current in the windings will reverse when the op-amps go high and low. They are not needed if a dual polarity power supply is used. Simply ground the windings that go to the emitters. A motor rated at 16.8 volts and 280 mA consumed only 30mA in the above circuit when unloaded. Try whatever stepper motor is available and adjust the power supply voltage for proper operation. Don't expect to get much torque from this circuit! For more drive capability connect two transistors to the op-amp outputs in the same manner as the Vcc/2 circuit above. Leave out the resistors and simply connect the two bases to the op-amp output. The two emitters connect together and to the motor winding.

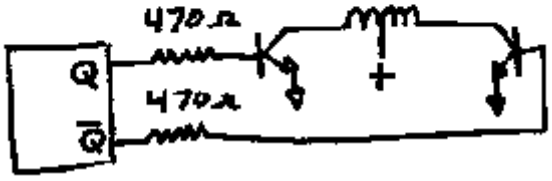
The above circuit is all that is needed for many applications but the following circuit has a bit more flexibility.



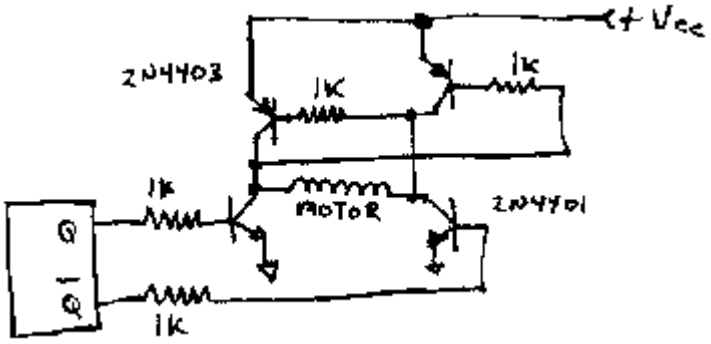
In this circuit the 74HC74 directly drives the stepper motor for low power applications. The two flip-flops are alternately clocked to give the desired "follow-the-leader" pulse train. The 16.8 volt motor (1.8 degree step size) described above draws only 20 mA in this circuit and a tiny 15 degree step size, 12 volt motor only draws 30 mA. The unused inverters are wired to form a slow pulse generator which may be used to randomly change the direction of rotation. Actually, the change of direction will often synchronize with the spin oscillator giving a back-and-forth action unless the oscillator frequency is just right. This change of direction will add interest to moving displays.

The circuit generates four control signals and adding circuitry for high power operation is fairly easy. If a center-tapped motor is used then the following connection will work:





You will need one of these circuits for each motor winding. If the motor doesn't have a center-tap then try the circuit below. The values are only representative and may vary depending upon the motor and transistor gain. No provisions are shown to protect against inductive kickback but small motors don't seem to generate much. However, some care may be required if larger motors and higher currents are used. Consider adding diodes from each motor winding to Vcc (the cathode goes to Vcc). Also, the 2N440Xs are only good for a few hundred milliamps so choose heftier transistors and use lower value resistors if higher motor currents are expected.



The flip-flop shown is an HC device and its power supply should be limited to 5 or 6 volts but the flip-flop could be a 4000 series device if driver transistors are added. The 4000 series devices cannot supply much current but they will run on +15 volts. Increase the resistors to 10k and use darlington transistors (or mosfets such as the VN10KM) in place of the NPNs shown.

Here is a note sent in by Garin :

I appreciate your experimenters internet home site for new electronics nuts. A good source of a ready made "robot" stepper motor system is an old dot matrix computer printer. Not only does it contain several motors, but it also has in it the electronics to drive them which can be interfaced to your control computer via the parallel port. Old manuals list control codes which can be translated into your application much faster and cheaper than designing your own system. In fact with the advent of color inkjet printers, many old dot matrix PC computer printer can be had for free. Hopefully this note will be of help.