

**microbric**TM
www.microbric.com

VIPER

The Viper Manual

Version 1.3
October 2005

1. Getting Started

- 1.1 How this manual works
- 1.2 Breaking out your pieces

2. The Components

- 2.1 The Motherboard
- 2.2 The Modules
- 2.3 The Microbric 'Brics'
- 2.4 Connecting the Batteries
- 2.5 Joining Modules to your Motherboard using the Brics

3. Programming the Microcontroller on your Motherboard

- 3.1 Installing the software
- 3.2 Making a LED flash
- 3.3 Adding a slide switch
- 3.4 Adding a buzzer
- 3.5 The push button
- 3.6 The motor forwards/backward
- 3.7 Using bump sensors
- 3.8 The infrared receiver

4. Building your Bump Robot

- 4.1 What will it do?
- 4.2 Putting it together
- 4.3 Downloading the program to drive your bump robot
- 4.4 The code

5. Building your Remote Control Robot

- 5.1 What will it do?
- 5.2 Putting it together
- 5.3 Preparing the Remote Control
- 5.4 Downloading the program
- 5.5 The code

6. What Now?

- 6.1 Cleaning up the edges
- 6.2 Variations
- 6.3 Go online

1. Getting Started

Finally! An electronics set that is really easy to build without sacrificing functionality! Welcome to Microbric, the first solderless construction set made for electronics enthusiasts.

1.1 How this manual works

This manual will take you through the various components that you have received in your kit and how to join them together. It will describe how to use your computer to program the individual parts of your project. You will then be shown how to put a basic Bump Robot together, and how to download a program to drive it and, finally, you will see a diagram of a completed Infrared controlled robot for you to build, and download the code to drive it. In addition, you can go online to view Frequently Asked Questions, New Possibilities for construction and New Code for programming as well as finding out the latest news in Microbric electronics.

If you follow the manual through from start to finish, you will learn everything you need to know to create the robots we have given you, as well as the knowledge and skills needed to build and program unique robots of your own. Additional information about programming your robots is included in the Basic Micro IDE help menu.

1.2 Breaking out your pieces

Your kit comes with a range of pieces, shown below.

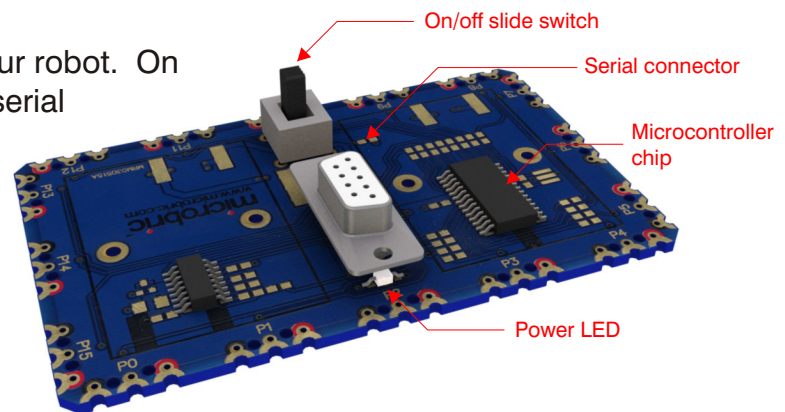
A number of your pieces will have arrived within PCBs (printed circuit boards). You need to remove these pieces before you can start building your robot. Most pieces can be pushed out with your thumbs, but if you are having difficulties, use the end of your screwdriver to gently lever them out.



2. The Component

2.1 The Motherboard

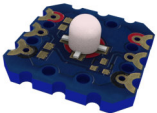
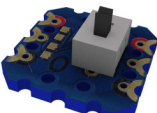
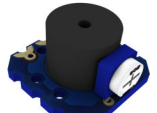
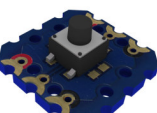
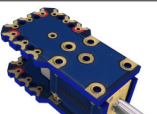
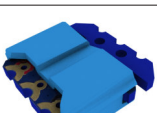
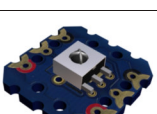
The Motherboard contains the 'brains' of your robot. On it you will find the Microcontroller chip, the serial connector (9 pin female D connector) for programming your microcontroller, an on/off slide switch, Pins (edge contacts), a Power LED, and a variety of components and integrated circuits needed to connect the circuits electronically and mechanically.



2.2 The Modules

Along with your motherboard you have received a range of little circuit boards that hold electromechanical devices. Each of these has a different function and when joined to the motherboard at the pins, you can program them to perform their functions.

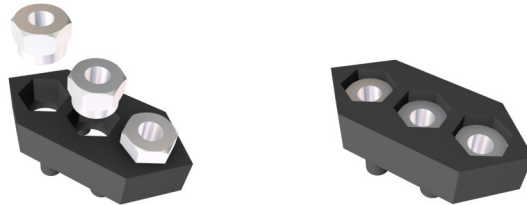
The modules consist of:

LED module	This module contains an LED (light emitting diode) and the circuitry to drive it. A LED is a tiny light, used, amongst other things, to show that an electronic device is turned on.	
Slide Switch module	This module contains a small switch and the circuitry to drive it. A switch allows us to turn an electronic device on, or 'switch' between two electronic functions.	
Buzzer module	This module contains a Buzzer and the circuitry to drive it. The buzzer is able to make sounds and even play simple tunes.	
Button module	This module contains a small push Button and the circuitry to drive it. It can be used to start or stop an electronic function.	
Motor module	This module contains a small Motor and the circuitry to drive it. The motor can be used to drive wheels forwards and backwards.	
Bump Sensor module	This module contains a Bump Sensor and the circuitry to drive it. A bump sensor can detect when it touches something, such as bumping into an object.	
Infrared Receiver module	This module contains an Infrared Receiver and the circuitry to drive it. An infrared receiver can pick up a signal sent using infrared light from a remote control device.	

2. The Component

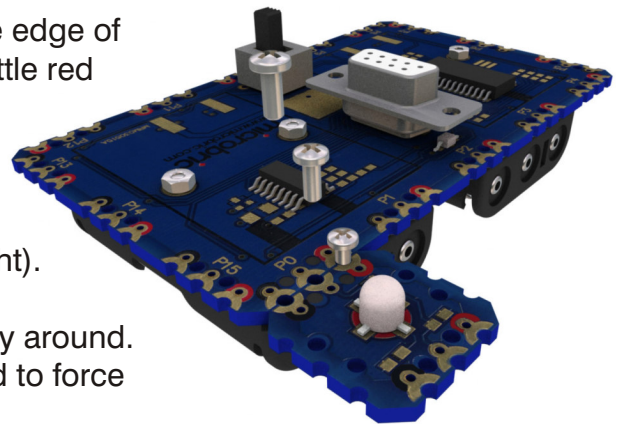
2.3 The Microbric 'Brics'

Microbric 'Brics' are tiny building blocks which can be used to hold the motherboard and modules together. These tiny black pieces fit into holes along the edge of the motherboard and the modules (and other parts) and, when screws are added, complete the circuit. You will need to assemble your Brics before you can use them. To do this, fit 3 of the tiny nuts into each bric.



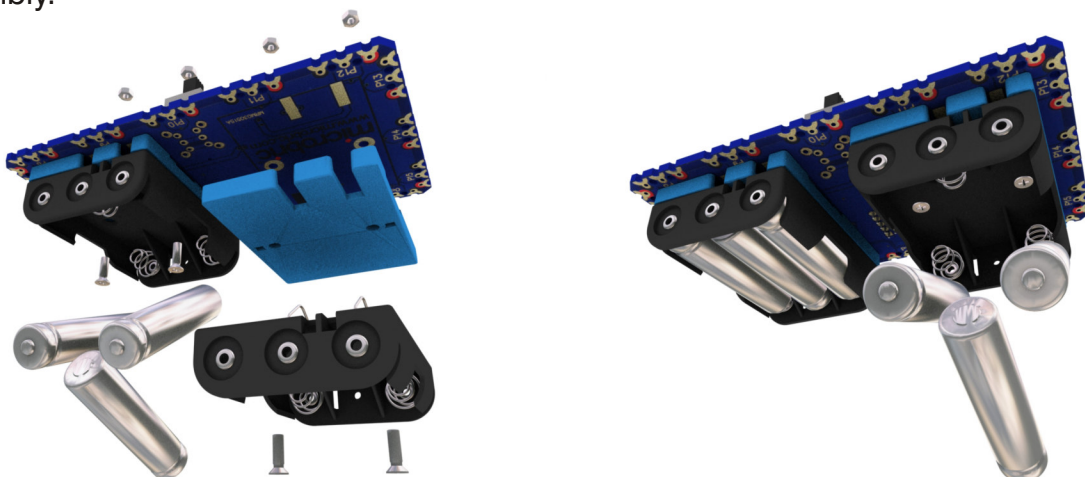
You can then use the Brics to attach the various modules (and other pieces) to the motherboard. When connecting modules to the motherboard you need to make sure of a few things:

1. Make sure that the little red dot on the edge of the module lines up exactly with the little red dots on the edge of the Motherboard (at the pins).
2. Make sure that you use all 3 screws and do them up firmly (but not too tight).
3. Make sure that the Bric is the right way around. It will only fit one way you do not need to force pieces together!



2.4 Connecting the Batteries

The motherboard is powered by batteries. You have been provided with four counter sunk screws that are different to all of the others (they have a flat, instead of a rounded, head), and 4 nuts. Use these to attach the holder to the underside of the motherboard, making sure that the gold battery contact pads are aligned with the slots in the blue battery mounts and the wires from the battery holders. (See the diagram below). The kit requires 6 AAA batteries. Put these in before flicking the switch to power up your motherboard. Once it is turned on, check that the tiny red power LED is lit up this will indicate that you have put it together correctly. If it doesn't light up, check that your batteries are all in the right way and that they are pushed in firmly before rechecking your assembly.



2. The Component

2.5 Joining Modules to your Motherboard using the Brics

The modules are joined to the motherboard using Brics. You connect each module at a pin. The pins are written around the outside edge of the motherboard as P0, P1, P2 etc.

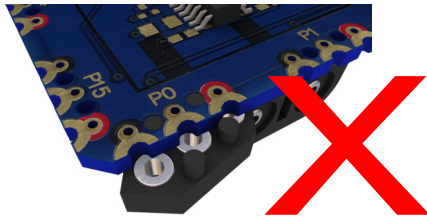
Let's try connecting a module:

You can then use the Brics to attach the various modules (and other pieces) to the motherboard. When connecting modules to the motherboard you need to make sure of a few things:

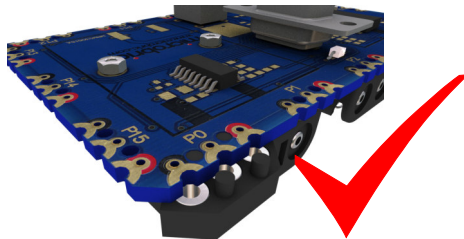
1. Find a LED module, from your module collection.
2. Line up the red dot on your LED module with the red dot on the Motherboard at P0 (Pin 0).

3. Now fit a Microbric into the holes on the edge of your LED module

NOTE: The tiny pins on the Bric fit into the holes if you put the Bric in the wrong way, the bric will sit crooked, i.e.

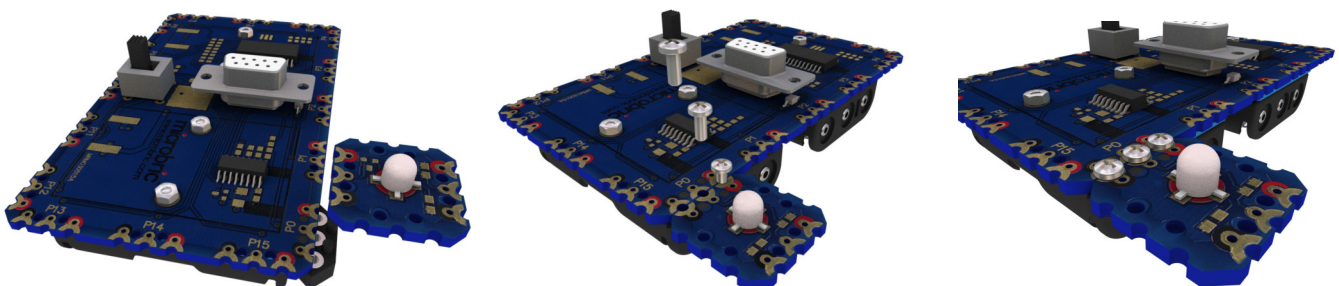


However, if you turn it up the other way it will sit straight. i.e.



4. Fit the Bric into the holes at P0 on the motherboard.
5. Now screw in three screws firmly to complete the connection.

If you've done it correctly, your Module and Motherboard should look like this:



3. Programming the Microcontroller on your Motherboard

Your motherboard has a microcontroller that can hold a range of instructions to drive various electronic components. The sets of instructions are called programs and writing the instructions is called programming.

3.1 Installing the software

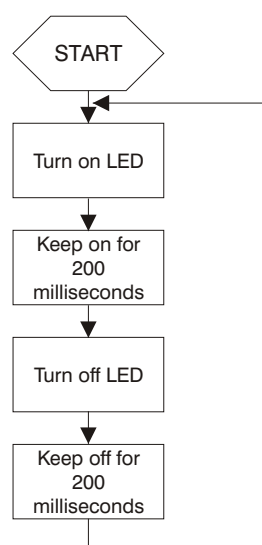
Programming computers or microcontrollers is done using programming languages. Computers which are electronic can only understand the simplest of languages which is written in binary, that is 0s and 1s (or an electrical ON or OFF). As you can imagine, programming a computer entirely in 0s and 1s would be very, very slow and tedious. So, higher level programming languages have been developed to make this easier. There are hundreds of languages available, however our Microcontroller uses a special programming language which has been written for it based on the BASIC programming language. BASIC is easy to learn and remember.

To write programs, and to get the Microcontroller to read them, you need to install a program which is provided on the CD-ROM called BMAtom.exe. You need to install this before we can do any programming to tell your motherboard, or robot what to do. To do this, double click on BMAtom.exe to run the set-up. Make sure the program installs into the folder C:\Program Files\Basic Micro ATOM IDE 2.2. It will ask you to restart your computer. Do so.

3.2 Making a LED flash

Now let's write our first program and then download it to your microcontroller.

1. Earlier we connected a LED module to pin 0 (P0) of the motherboard. If you haven't done this step, go to **2.5 Joining Modules to your Motherboard using Microbrics** and follow the instructions there.
2. Open **Basic Micro ATOM IDE 2.2**.
3. Start a new program by going to **File > New** and choose an **MBasic File**. Give the file the name **LEDprogram.bas**
4. A good thing to do before writing code for a program is to draw a flow chart of how the program will work. Here is an example of a flow chart that shows what our program will do:



3. Programming the Microcontroller on your Motherboard

5. Write the following program into the text window.

NOTE: It is important to type in exactly what is shown, because programming languages use very specific code called syntax. If the syntax is incorrect, the software, and later the hardware which is getting an interpreted version in binary, will not be able to read it. Note that the tab indents are to make it easier to reader. Writing after a semicolon are comments and don't affect the compiling or how the program operates.

```
Main
    high P0      ;make the output at pin 0 high (i.e. 5V)
    pause 200    ;hold it (at high) for 200 milliseconds
    low P0       ;make the output at pin 0 low (i.e. 0V)
    pause 200    ;hold it (at low) for 200 milliseconds
    goto Main    ;go back to Main at the top, this makes the program loop
End
```

Some things to note about the program above:

- Notice that we have used the Tab key on the keyboard to indent parts of the program. This isn't necessary, but makes it easier to read.
- Notice that the parts in grey after the semi-colons are descriptions of what the program is doing. These also aren't necessary, but help you to understand what your program is doing, and to remember what it was doing when you come back to it in a year's time (so spelling mistakes don't matter in here).

6. Make sure the serial cable is securely connected to the COM 1 serial port at the back of your computer and to the Serial connector on your Microbric motherboard. Also, make sure your Microbric motherboard has batteries and is switched on.

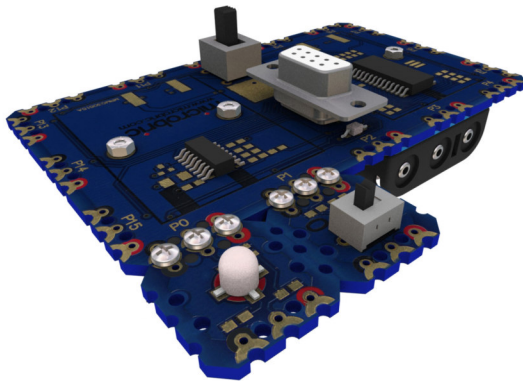
(Note, if your COM 1 port is already being used, use COM 2 and change the system setting in your BasicMicro IDE software by going to Tools/System Setup and changing the Port Setting to COM2.)

7. In the IDE menu, press **Program**. This compiles the program (changes it from your high level program to binary so that the computer can understand it) and then downloads it to the Basic ATOM chip on the Microbric board. If there are any errors (there shouldn't be any in the above program) the display window at the bottom will inform you what the errors are.
8. Once the software has finished downloading your program, you should see the light at pin 0 start to flash. You can disconnect the serial cable and it will keep going. The program stays in the chip, so you can turn the flashing LED on and off using the motherboard power switch.

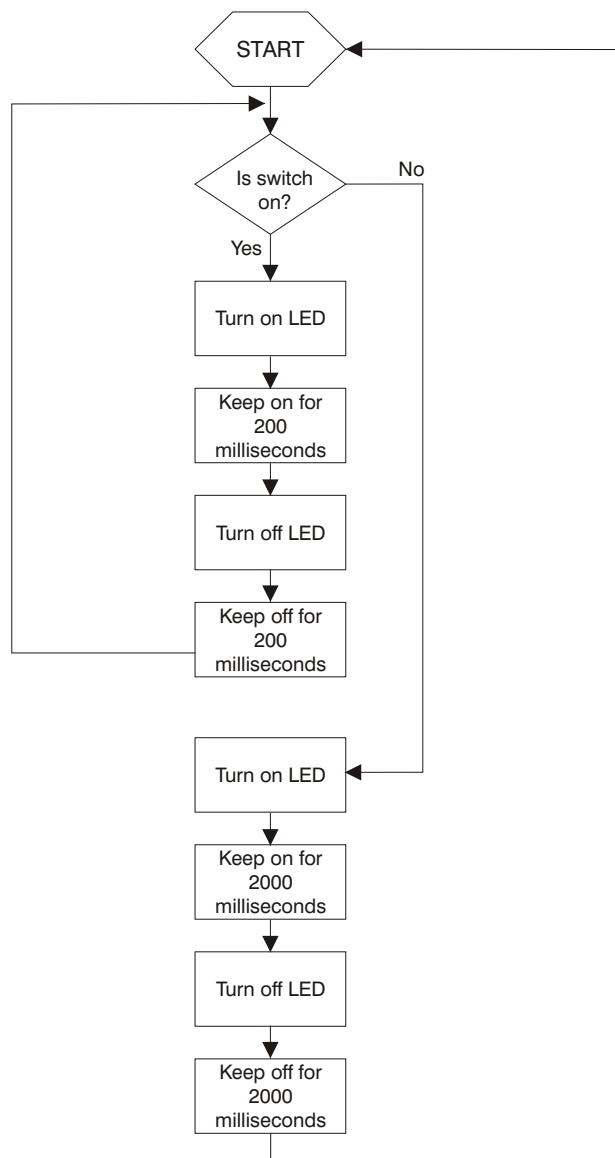
3. Programming the Microcontroller on your Motherboard

3.3 Adding a Slide Switch

1. Now let's attach a slide switch and use it to control the speed of flashes on the LED.
2. Attach the Slide Switch module to P0 on the edge of your Micobric motherboard next to the LED module (note the 0 and 1 written in gold on the module)



3. Again, let's draw a flow diagram before we write the program. Here is one for the program we will write.



3. Programming the Microcontroller on your Motherboard

4. Now change the program so that it is the same as below. It is just the same as above but modified with an “IF/then” statement (to set up two possible paths for the program to follow):

```
Main
if in1 = 1 then    ;tests whether the slide switch is at 0 or 1. If switch is at 1 then
                  ;goes to next line, otherwise goes to Else below
    high P0        ;make the output at pin 0 high (i.e. 5V)
    pause 200      ;hold it (at high) for 200 milliseconds
    low P0         ;make the output at pin 0 low (i.e. 0V)
    pause 200      ;hold it (at low) for 200 milliseconds
    goto Main      ;go back to Main at the top, this makes
                  ;the program loop
else
    high P0        ;make the output at pin 0 high (i.e. 5V)
    pause 2000     ;hold it (at high) for 2000 milliseconds
    low P0         ;make the output at pin 0 low (i.e. 0V)
    pause 2000     ;hold it (at low) for 2000 milliseconds
    goto Main      ;go back to Main at the top, this makes
                  ;the program loop
endif
End
```

Some things to note about the program above:

- The module at Pin 1 accepts input from the user and so it is referred to as in1. this tells the microcontroller to expect input at this pin.

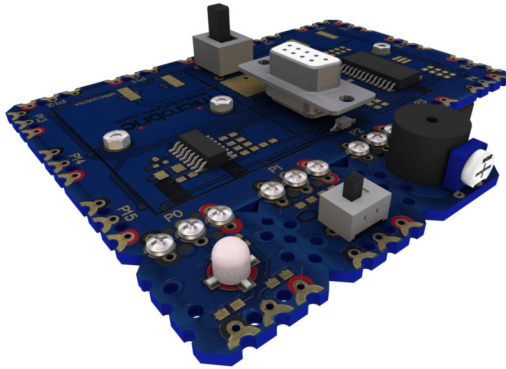
5. Save your program as **SlideSwitchprogram.bas**
6. Click on “Program” to compile and download the code onto the chip.
7. Remove the serial cable.
8. Switch the slide switch back and forth to see the difference in the LED flashing speeds.

3. Programming the Microcontroller on your Motherboard

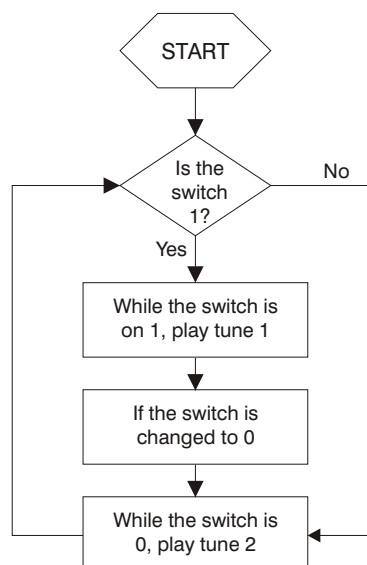
3.4 Adding a buzzer

Now let's look at how the buzzer works. We will use the `sound` command to control the buzzer.

1. Connect the buzzer module to P2.



2. Now let's write a program that plays two different "tunes" depending on whether the slide switch is at 0 or 1. The flow diagram will look like this:



3. To program the buzzer, we will use the `sound` command. Let's have a look at how the `sound` command works. The `sound` command takes the general form:

`SOUND pin,[duration1\note1,duration2\note2.....durationX\noteX]`

pin: this is the pin number that the buzzer is connected to (i.e. P2)

duration: this is a value between 1 and 65535 telling the buzzer how long to stay on the given note for (measured in milliseconds).

note: is a number between 0 and 32767 that specifies the frequency of the note.

You can put together a series of notes to make a tune by specifying note length and note frequency.

4. Open a new file in basic ATOM IDE by going to File > New and choosing an **MBasic File**. Save this file as **Buzzerprogram.bas**

3. Programming the Microcontroller on your Motherboard

5. Now, type the following into the text window:

```
Main
while in1 = 1           ;processes instructions below while slide switch (at P1) is at 1
  sound P2,[400\450,50\0,400\450,400\675,50\0,400\675,50\0,400\750,50\0,400\750,800\675]
                        ;sound a series of notes (tune 1) at Pin 2
wend                    ;end for the while loop command

while in1 = 0           ;processes instructions below while the slide switch (at P1) is at 0
  sound P2,[50\1000,50\950,50\900,50\850,50\800,50\750,50\700,50\650,50\600,50\550,50\500]
                        ;sound a series of notes (tune 2) at Pin 2
wend                    ;end for the while loop command

goto Main              ;go back to main
End                    ;tells the compiler the code has finished
```

Some things to note about the program above:

- The While command sets up a condition that while the switch is in a certain position (i.e the 1 position) a program sequence will follow. This will keep looping and will test the condition each time before it proceeds. As soon as that condition changes (i.e. The switch is moved to the 0 position) it will break out of the loop and continue through the code further down the program.
- Each **while** command has a corresponding **wend** command to show where the while loop ends.

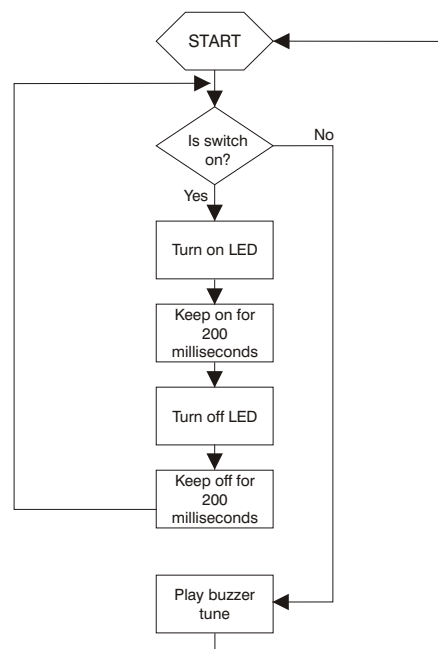
6. Save.

7. Click on “Program” to compile and download the code onto the chip.

8. Remove the serial cable.

9. Switch the slide switch back and forth to play the two different tunes.

10. Now we can modify the above program to include a LED and a buzzer and use the switch to interchange between the two.



3. Programming the Microcontroller on your Motherboard

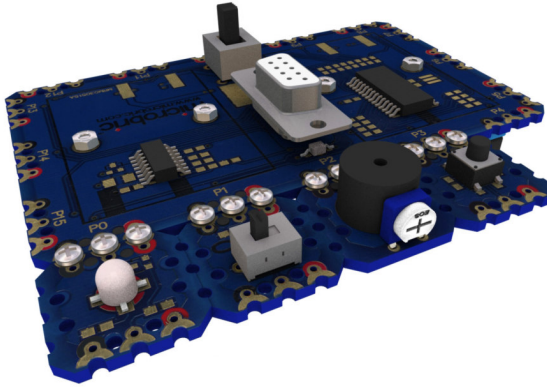
```
Main
  if in1 = 1 then      ;tests whether the slide switch (input at P1) is at 1 or 0
    high P0           ;make the output at port 0 high (eg. 5V)
    pause 200         ;hold it (at high) for 200 milliseconds
    low P0            ;make the output at port 0 high (eg. 5V)
    pause 200         ;hold it (at low) for 200 milliseconds
    goto Main         ;go back to Main; this causes the program to loop through the lines above
  else
    sound P2,[400\450,50\0,400\450,400\675,50\0,400\675,50\0,400\750,50\0,400\750,800\675]
    ;sound a series of notes (tune 1) at Pin 2
    goto Main         ;go back to Main and loop through
  endif              ;end of the if statement
End                  ;tells the compiler the code has finished
```

11. Save this program as **Switchprogram2.bas**
12. Click on “Program” to compile and download onto the Microbric motherboard.
13. Unplug the serial plug.
14. When the slide switch is equal to 1, the LED should start to flash. If the slide switch equals 0, the buzzer should start playing a tune.

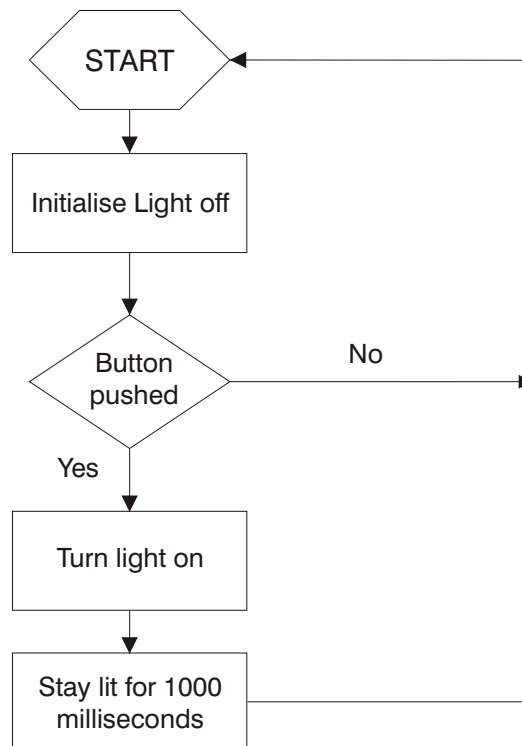
3. Programming the Microcontroller on your Motherboard

3.5 The Push Button

This section will give you an example of the use of a push button. Push buttons are extremely useful for a range of things and so the **Button** programming code has quite a number of parameters (parts to the code command that can vary).



1. Attach the Push Button module to P3 on the edge of the motherboard.
2. Open a New file and name it **Pushbutton.bas**
3. Now, write the following program in BASIC Atom IDE:



3. Programming the Microcontroller on your Motherboard

PushButton var in3 ;Create a variable called Pushbutton that accepts input from pin 3

Main

```
low P0          ;initialise LED off
if PushButton = 1 then LightLED
                  ;if the button is pushed jump to LightLED
goto Main       ;loop back to Main
```

LightLED

```
high P0         ;light the LED at pin 0
pause 1000      ;stay lit for 1000 milliseconds
goto Main       ;loop back to Main
```

End

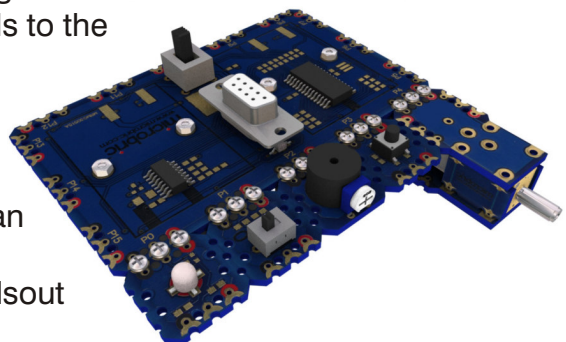
Some things to note about the program above:

- This time we have defined a variable called **PushButton** at the very beginning of the program. It is called a 'variable' because the value can vary according to input from the user. Note that the value for PushButton will be input at Pin 3.
- We have added another 'line label' besides **Main**. This allows the Main program to run a sub-program if a condition is met. The name of the line label should match what the function does, in this case **LightLED**. Line labels are used to break up a long program into smaller 'sub-programs'.

4. Click on “Program” to compile and download the code to the microcontroller on the motherboard.
5. Remove the serial cable.
6. Press the button at Pin 3 to turn the LED on and off.

3.6 The Motor

1. Attach the Motor module to P4 on the edge of the motherboard. Connect one of your wheels to the motor axle.
2. Before writing the program below, let's have a look at the “serout” and “pulsout” commands in BASIC. The serout command allows you to specify such things as the direction and speed of the motor. The pulsout sends a tiny pulse to brake the motor.



They are written in the following general formats:

SEROUT pin, baudmode,[output data]

pin: this is the pin number the motor is connected to.

baudmode: is a variable or constant that specifies serial timing and configuration in our case we use an i to say that we use an inverted configuration and 2400 as the baud rate.

3. Programming the Microcontroller on your Motherboard

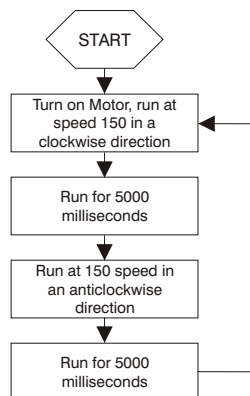
output data: is information that effects the direction and speed of our motor in our case the data includes direction (i.e. clockwise or anticlockwise) and speed which ranges from 0 to 255.

PULSOUT pin, time

pin: this is the pin number the motor is connected to.

time: the duration of the pulse, measured in microseconds.

3. Now open a new file, save it as Motorprogram.bas and then type in the following program:



;Initialsie Motor output

high P4
pause 50

Main

Serout P4,i2400,["C",150]
Pause 5000
pulsout P4,6000

Serout P4,i2400,["A",150]
Pause 5000
pulsout P4,6000

goto Main

End

;Set the output high

;Give the motor module time to get ready

;Drive motor Clockwise at speed 150

;Continue for 5000 milliseconds

;Brake Motor by sending a pulse of 6000 microseconds
;to Pin 4

;Drive motor Anticlockwise at 150 speed

;Continue for 5000 milliseconds

;Brake Motor by sending a pulse of 6000 microseconds
;to Pin 4

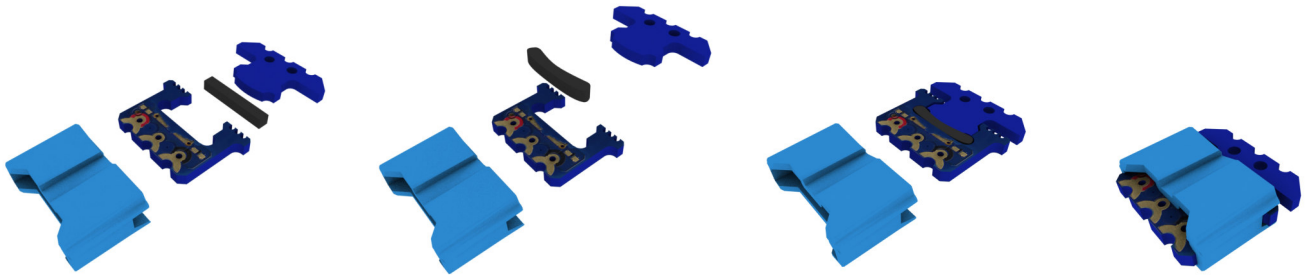
;loop to repeat the code

4. Click on "Program" to compile and download the code onto the chip. The motor should turn one way stop briefly, and then go the other way.
5. Remove the serial cable.
6. Turn the motherboard off to stop the motor.

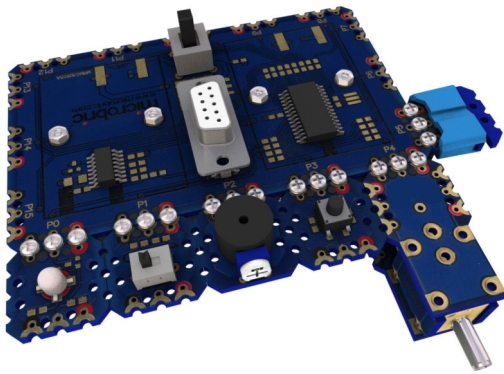
3. Programming the Microcontroller on your Motherboard

3.7 The Bump Sensor

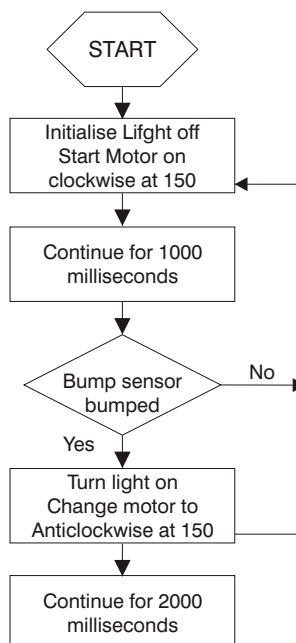
Before using the Bump Sensor module, you will need to assemble the bump sensor itself.



1. Attach the Bump Sensor module to P5 on the edge of the motherboard



2. Open a new file, save it as **BumpSensorprogram.bas** and type in the program below to see how the bump sensor works. Again, note the use of in5 instead of P5 because we are getting input from a user, and note the use of a separate function for Back when the input condition is met.



3. Programming the Microcontroller on your Motherboard

;Initialsie Motor output	
high P4	;Set the output high
pause 50	;Give the motor module time to get ready
Main	
low P0	;set the LED at P0 to low (eg. 0V)
Serout P4,i2400,["C",150]	;Drive motor Clockwise at 150
Pause 100	;Continue for 100 milliseconds
if in5 = 1 then Back	;If the bumper sensor at P5 is bumped,
	go to Back
goto Main	;Loop back to Main
Back	;Back
pulsout P4,6000	;brake the motor
high P0	;set the LED at P0 to high (eg. 5V)
Serout P4,i2400,["A",150]	;Drive the motor Anticlockwise at 150
Pause 2000	;Continue for 2000 milliseconds
goto Main	;Loop back to Main
End	

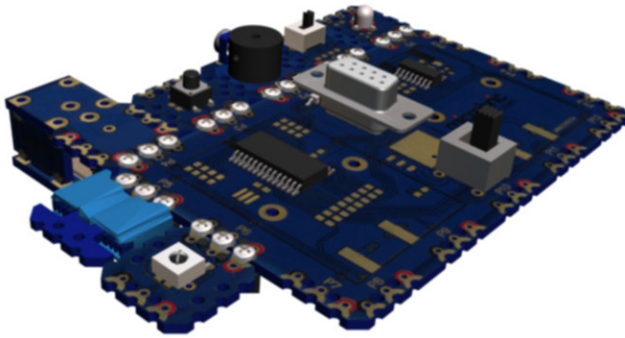
3. Click on Program to compile and download onto Microbric motherboard.
4. Unplug the serial plug.
5. The motor should start turning the wheel clockwise. When the bumper is pushed, the LED should light and the wheel should change direction to anticlockwise.

3. Programming the Microcontroller on your Motherboard

3.8 The Infrared Receiver

The Infrared Receiver module is the most complex of the kit to program. However, it is also extremely versatile.

1. Attach the Infrared Receiver module to Pin 6 on the Microbric motherboard.

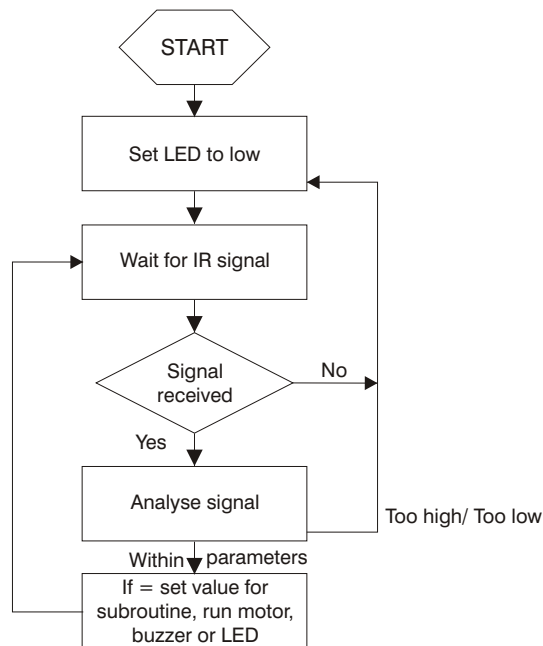


2. In order to use the remote control with the microcontroller, you will need to follow a sequence of steps to preset the remote to work with the microcontroller.
 - a. Put two AAA batteries into the remote control unit.
 - b. Simultaneously hold down the S button (in the middle of the arrows) and the B button on the remote (a red light will go on in the top left hand corner of the remote.)
 - c. Press the number sequence 0 1 3 on the remote buttons.
 - d. Press the red power button on the remote.
 - e. The remote is now configured to work with your microcontroller.

Note that buttons A, C, D, E, F and G are for setting the remote control into different modes which are not required for this project. Avoid pressing these buttons as this will inadvertently set your remote into another mode. You can always return to the 'B' mode by pressing the B button

3. Open a New File and call it **IRProgram.bas**.
4. Now type in the following program. It is long and complex, so save often. Or alternatively you can find this program on your CD. Save it (**File/Save As...**) to a location on you hard drive before programming it into the microcontroller.
5. In this program you will be using the **PULSIN** command. This is a command that tells the microcontroller to wait for a pulse signal. You will note that it specifies the Pin for the input and then states what to do in the 0 state and the 1 state.

3. Programming the Microcontroller on your Motherboard



```

irdata      var    word           ;Set avariables to hold temporary data
headerlength var    word
pulselength var    word
IR_DATA_PIN con    P6             ;Infrared input is at Pin 6
  
```

Main:

```

low P0
gosub wait           ;Call the wait subroutine
if irdata <> 0 then gosub TestIRData ;If irdata is anything other than 0
                                ;run TestIRData
                                ;clear irdata
irdata = 0
goto Main
  
```

wait

```

irdata = 0           ;clear irdata
pulsin IR_DATA_PIN,0,WaitEnd,1,headerlength ;wait for a pulse, if one comes
                                ;put the length in 'headerlength'
if headerlength > 2250 then readir ;check if the pulse is greater than
                                ;2250uS. If so, goto 'readir'
  
```

WaitEnd

```

irdata = 0           ;clear irdata
return
  
```

;Start reading the pulses=====

readir:

```

if headerlength > 2750 then WaitEnd ;check if the pulse is greater than
                                ;2750uS, if so goto 'WaitEnd'
  
```

pulse1

```

pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength ;wait for the next pulse and
                                ;store the length in 'pulselength'

if pulselength < 400 then WaitEnd ;if 'pulselength' is less than
                                ;400uS then goto 'WaitEnd'
  
```

```

if pulselength < 1000 then pulse2 ;if 'pulselength' is less than
                                ;1000uS then goto 'pulse2'

irdata = irdata + %10000000000000000 ;Add a high bit 16
  
```

pulse2

```

pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength
if pulselength < 400 then WaitEnd
if pulselength < 1000 then pulse3
  
```


3. Programming the Microcontroller on your Motherboard

```

    pulsln IR_DATA_PIN,0,MotorForwardEnd,1,headerlength    ;whilst IR data is still
    goto motorforward                                       ;coming, keep in this loop
MotorForwardEnd
    pulsout P4,6000                                         ;Stop Motor
    return

motorbackward
    Serout P4,i2400,["A",255]
    pulsln IR_DATA_PIN,0,MotorBackwardEnd,1,headerlength    ;whilst IR data is still
    goto motorbackward                                       ;coming,keep in this loop
MotorBackwardEnd
    pulsout P4,6000                                         ;Stop Motor
    return

lightLED
    High P0
    Pause 2000
    return

playbuzzer
    sound P2,[400\450,50\0,400\450,400\675,50\0,400\675,50\0,400\750,50\0,400\750,800\675]
    return
end
```

Some things to note about the program above:

- This program will accept a signal from the remote control, analyse it, and then, according to the binary 16 bit number received, will either run the motor forwards, backwards, turn the LED on or play a tune.
- The 'pulse' sequence is necessary to check all the possible incoming combinations. It is tedious to type in, but gives this program its flexibility to use multiple buttons to drive it.
- Each of the buttons on your remote control has a specific 16 bit binary number (referred to as a 16 bit 'word'). You can see four of them represented in the **TestIRData** subroutine above. The 16 bit numbers for each of the buttons on your remote control are written in the table below. You can use them to program 14 separate functions.

3. Programming the Microcontroller on your Motherboard

Button	Code in Binary															
1 button	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2 button	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3 button	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4 button	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5 button	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
6 button	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
7 button	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
8 button	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
9 button	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
0 button	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
⌂ button	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
⌂ button	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
⌂ button	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
⌂ button	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
⌂ button	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
⌂ button	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0
+ button	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0
× button	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0
- button	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0
+ button	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0
⏻ button	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0

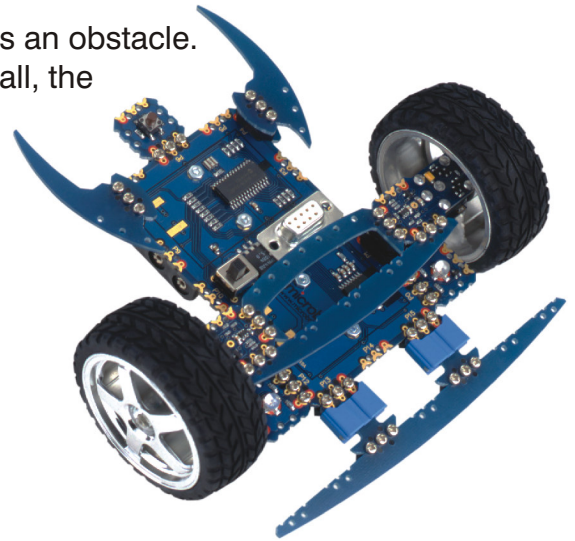
4. Building your Bump Robot

4.1 What will it do?

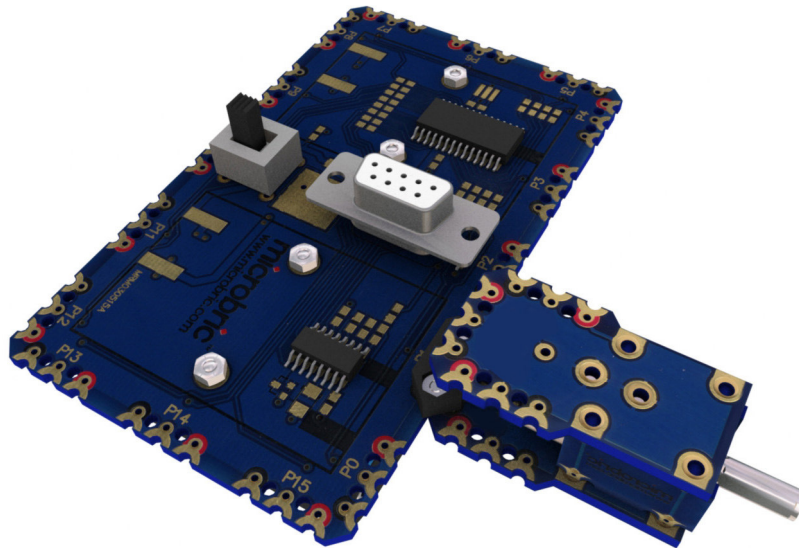
The Bump robot will drive in a straight line until it hits an obstacle. If the 'bump sensors' meet an obstacle, such as a wall, the robot will stop, reverse and try a new direction. You will press the button to start it moving.

4.2 Putting it together

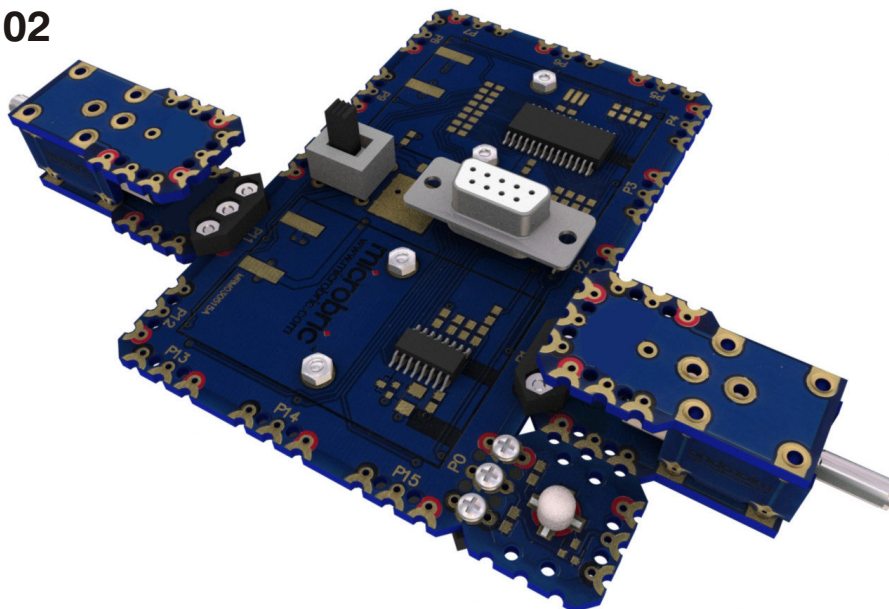
Disconnect the modules from the motherboard but leave the battery holders in place. Now follow the step by step Illustrations below to build your bump robot.



Step 01

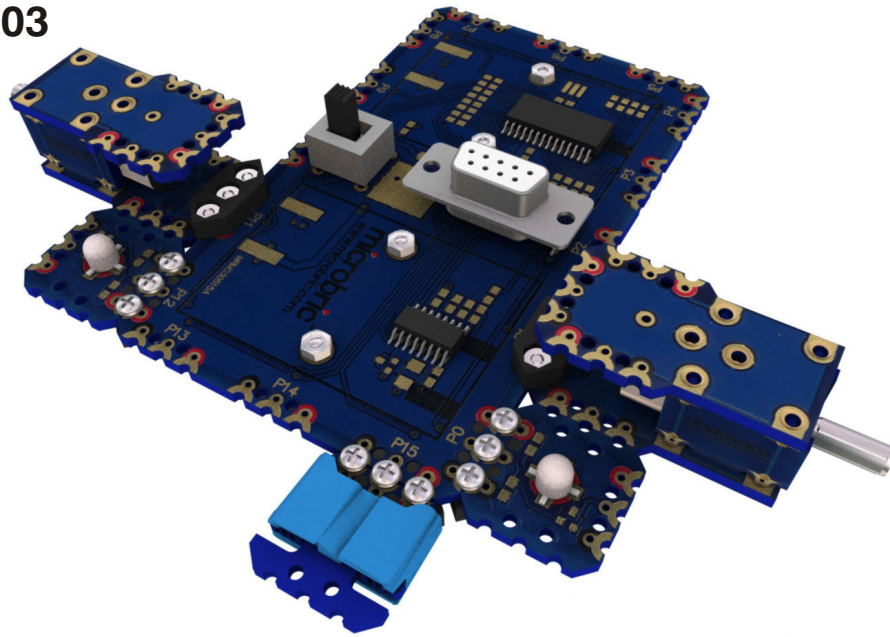


Step 02

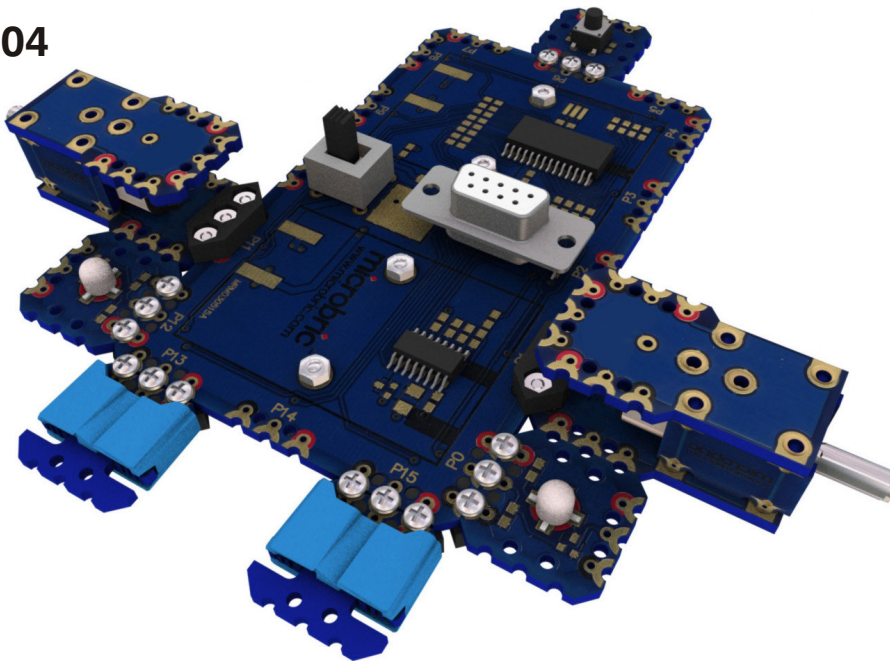


4. Building your Bump Robot

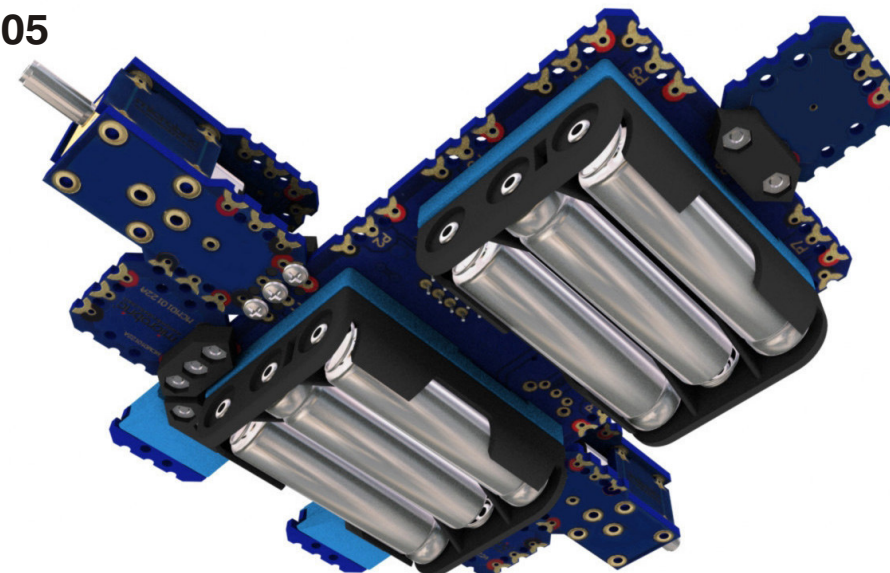
Step 03



Step 04

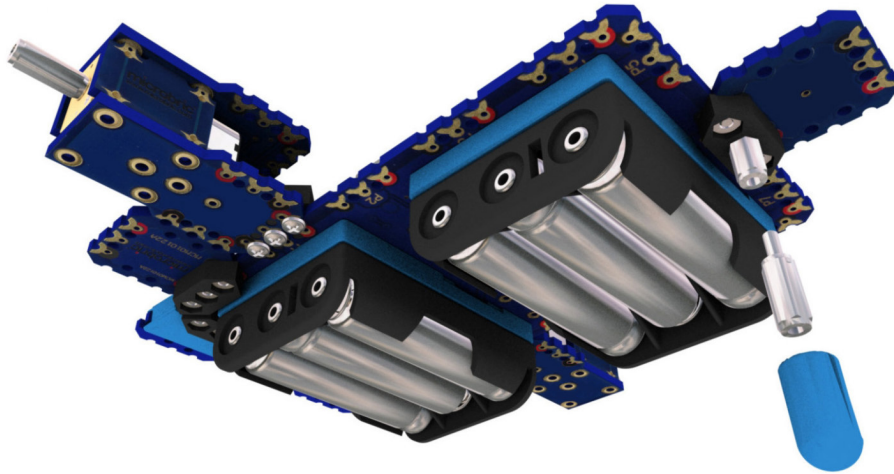


Step 05

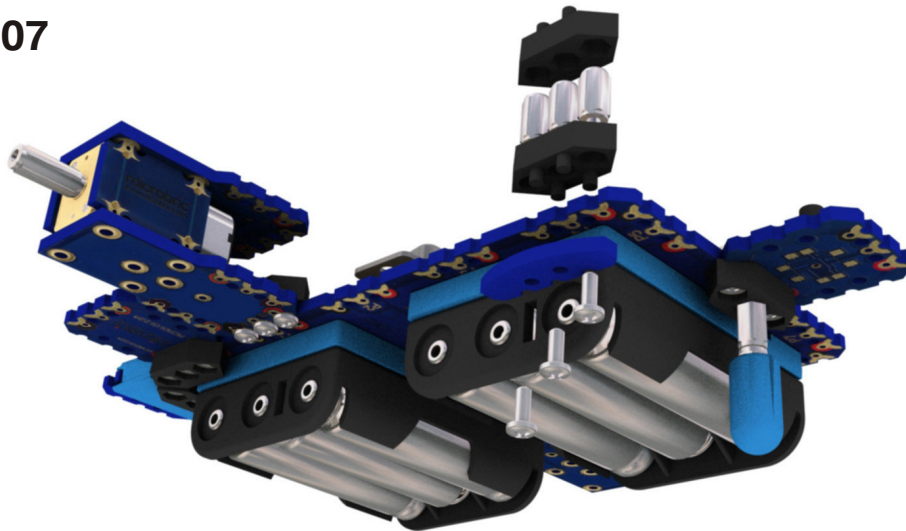


4. Building your Bump Robot

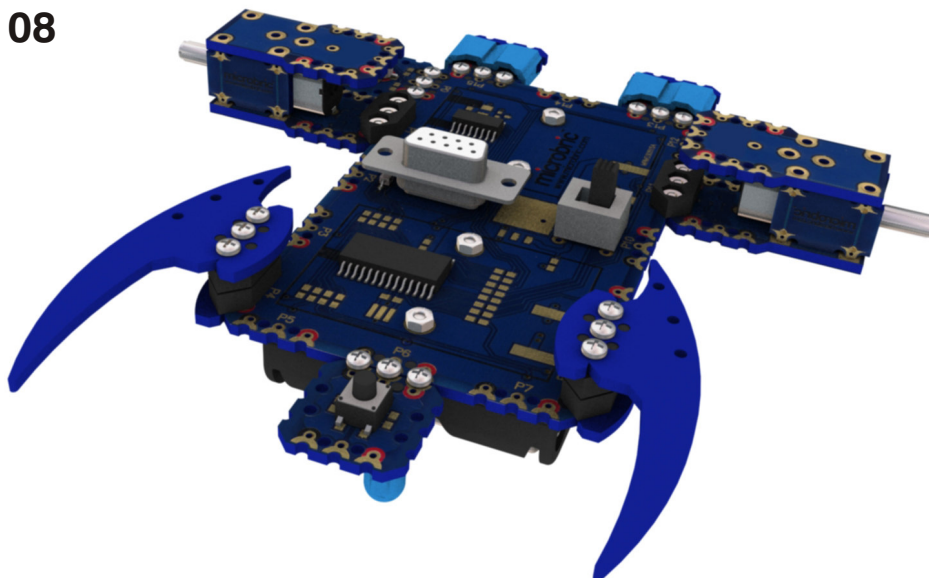
Step 06



Step 07

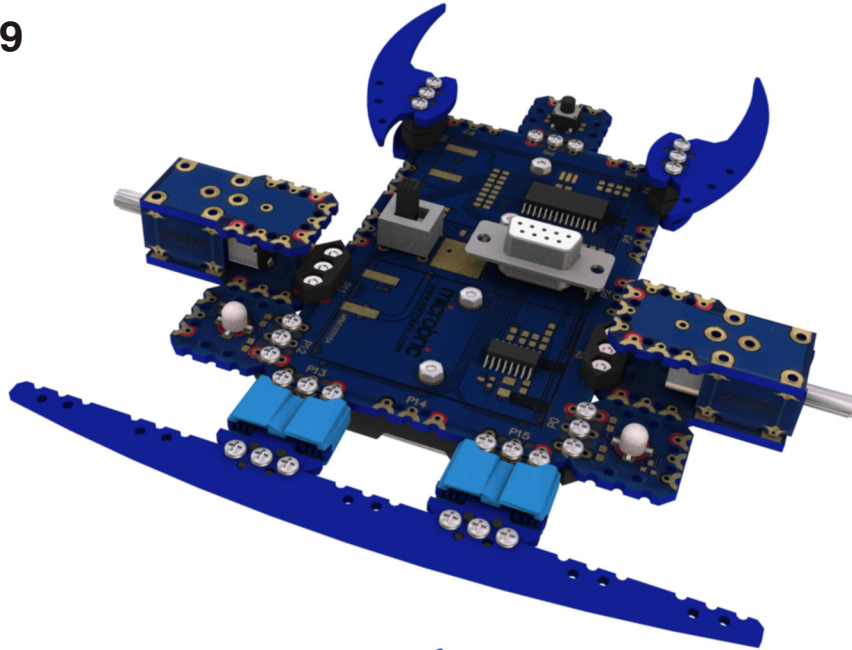


Step 08

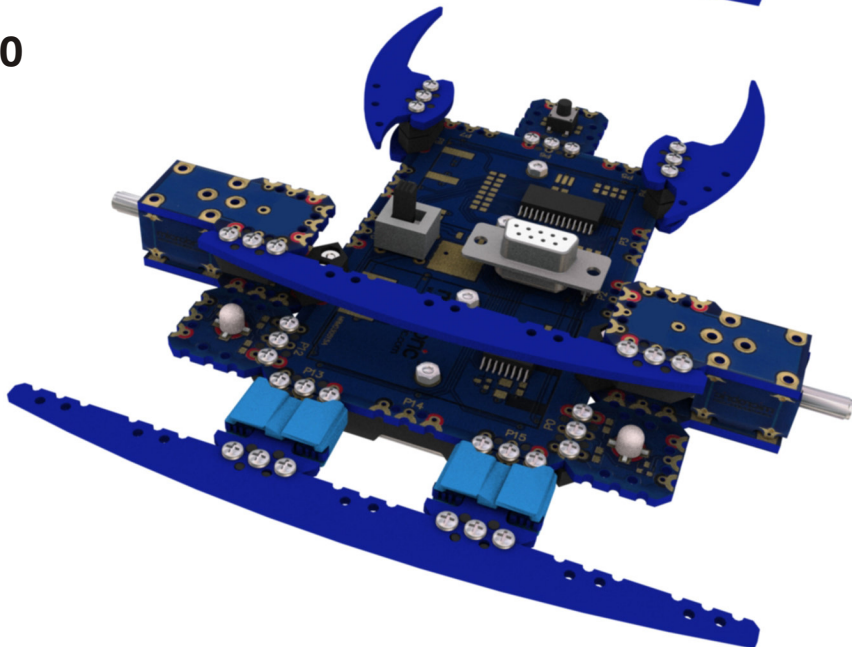


4. Building your Bump Robot

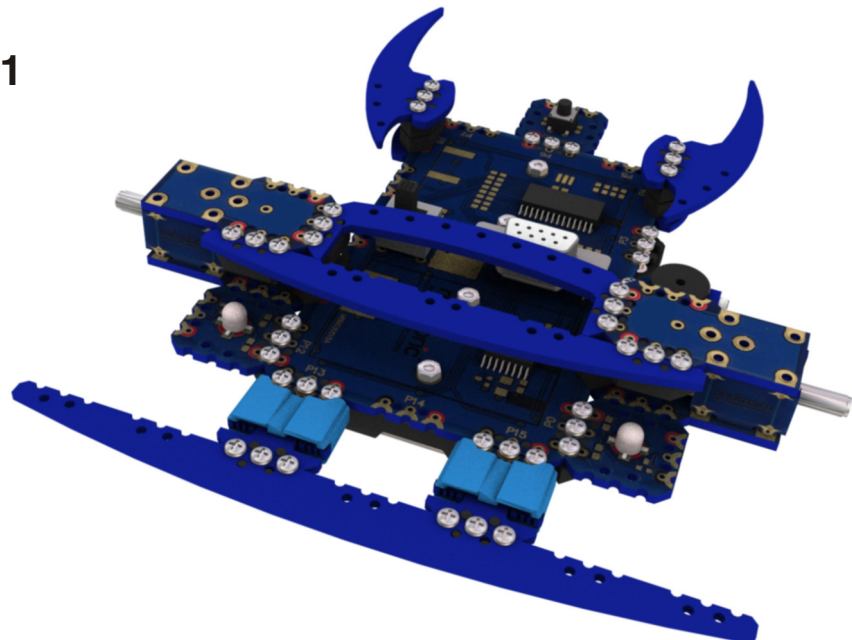
Step 09



Step 10

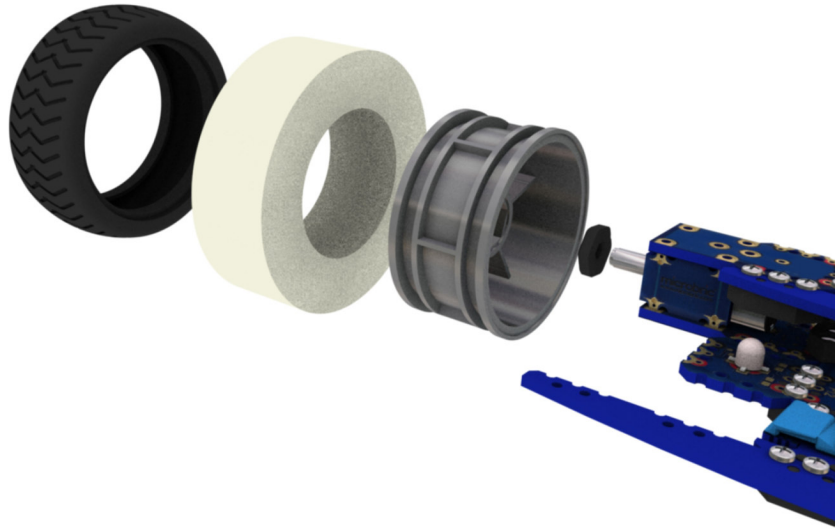


Step 11

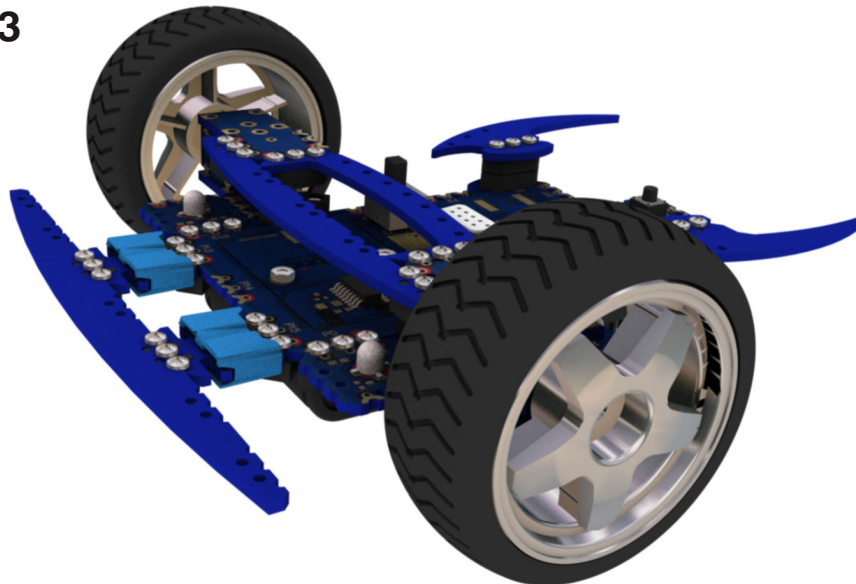


4. Building your Bump Robot

Step 12



Step 13



4. Building your Bump Robot

4.3 Downloading the program to drive your bump robot

Once your bump robot is built:

1. Open the program, **BumpRobot.bas** in BasicMicro IDE from your CD-ROM
2. Save the program (**File/Save As...**) to a location on you hard drive.
3. Connect the serial connector to the motherboard's serial port.
4. Click on Program to download the program onto your Bump Robot microcontroller on the motherboard.
5. Disconnect the serial connector.
6. Press the button on the Button Module to start your Bump Robot.

4.4 The Code

We have included the code here so that you can investigate the program used:

```
**Variables**
temp          var    word
PushButton    var    in6
LeftBumper    var    in15
RightBumper   var    in13

**Constants**
LeftLED       con    P0
RightLED      con    P12
RightMotor    con    P11
LeftMotor     con    P1

;Initialise Motor outputs
high RightMotor
high LeftMotor
pause 50

StopRobot:
  gosub StopWheels          ;Call the subroutine StopWheels
  low LeftLED               ;Turn LeftLED off
  low RightLED              ;Turn RightLED off
  pause 250                 ;Pause - Do nothing for 250mS
  if PushButton = 1 then StopRobot

Loop:
  if PushButton = 1 then Main ;If Push Button is pressed goto the
                              ;main program
  pause 10                   ;Pause - Do nothing for 10mS
  goto Loop                  ;Loop around to Loop

main:
  toggle LeftLED             ;Swap the state of Left LED
  toggle RightLED            ;Swap the state of Right LED
  gosub Forward              ;Call the subroutine "Forward"
  pause 200                  ;Pause - Do nothing for 200mS
  if LeftBumper = 1 then BackwardSpinRight
                              ;If the right bumper sensor is bumped
                              ;BackwardSpinRight
  if RightBumper = 1 then BackwardSpinLeft
                              ;If the right bumper sensor is bumped
                              ;BackwardSpinLeft
  if PushButton = 1 then StopRobot ;If Push Button is pressed goto StopRobot
  goto main                  ;Loop around to Main
```

4. Building your Bump Robot

BackwardSpinRight:

<code>gosub StopWheels</code>	<code>;Call the subroutine StopWheels</code>
<code>gosub Backward</code>	<code>;Call the subroutine Backward</code>
<code>gosub SpinRight</code>	<code>;Call the subroutine SpinRight</code>
<code>goto main</code>	<code>;Loop around to Main</code>

BackwardSpinLeft:

<code>gosub StopWheels</code>	<code>;Call the subroutine StopWheels</code>
<code>gosub Backward</code>	<code>;Call the subroutine Backward</code>
<code>gosub SpinLeft</code>	<code>;Call the subroutine SpinLeft</code>
<code>goto main</code>	<code>;Loop around to Main</code>

Forward:

<code>Serout LeftMotor,i2400,["A",150]</code>	<code>;Drive motor Anticlockwise at 150</code>
<code>Serout RightMotor,i2400,["C",150]</code>	<code>;Drive motor Clockwise at 150</code>
<code>Return</code>	

Backward:

<code>pause 50</code>	
<code>Serout LeftMotor,i2400,["C",150]</code>	<code>;Drive motor Clockwise at 150</code>
<code>Serout RightMotor,i2400,["A",150]</code>	<code>;Drive motor Anticlockwise at 150</code>
<code>for temp = 1 to 25</code>	
<code> pause 7</code>	
<code> if PushButton = 1 then StopRobot</code>	<code>;If Push Button is pressed goto StopRobot</code>
<code> next</code>	
<code>gosub StopWheels</code>	
<code>pause 50</code>	
<code>Return</code>	

SpinRight:

<code>Serout LeftMotor,i2400,["A",150]</code>	<code>;Drive motor Anticlockwise at 150</code>
<code>Serout RightMotor,i2400,["A",150]</code>	<code>;Drive motor Anticlockwise at 150</code>
<code>for temp = 1 to 25</code>	
<code> pause 15</code>	
<code> if PushButton = 1 then StopRobot</code>	<code>;If Push Button is pressed goto StopRobot</code>
<code> next</code>	
<code>gosub StopWheels</code>	
<code>pause 50</code>	
<code>return</code>	

SpinLeft:

<code>Serout LeftMotor,i2400,["C",150]</code>	<code>;Drive motor Clockwise at 150</code>
<code>Serout RightMotor,i2400,["C",150]</code>	<code>;Drive motor Clockwise at 150</code>
<code>for temp = 1 to 25</code>	
<code> pause 15</code>	
<code> if PushButton = 1 then StopRobot</code>	<code>;If Push Button is pressed goto StopRobot</code>
<code> next</code>	
<code>gosub StopWheels</code>	
<code>pause 50</code>	
<code>return</code>	

StopWheels:

<code>pulsout LeftMotor,6000</code>	<code>;Brake Left Motor</code>
<code>pulsout RightMotor,6000</code>	<code>;Brake Right Motor</code>
<code>Return</code>	

`end`

5. Building your Infrared Viper Robot

5.1 What will it do?

Your infrared Viper can be driven using your remote control. Having built the “Bump Robot”, now challenge yourself by building variations of the Viper!



5.2 Preparing the Remote Control

In order for the Microbric microcontroller to receive signals from the remote control unit that has come with your kit, you will need to work through a preset routine.

1. Put two AAA batteries into the remote control.
2. Simultaneously press the S button (in the middle of the arrows) and the B button (at the top of the remote) until the little red light goes on in the top left hand corner.
3. Now type in the numbers 0 1 3
4. Press the red power button
5. The remote is now ready to be used with your robot.

Note that buttons A, C, D, E, F and G are for setting the remote control into different modes which are not required for this project. Avoid pressing these buttons as this will inadvertently set your remote into another mode. You can always return to the 'B' mode by pressing the B button.

5.3 Downloading the program

Once your Infrared Viper robot is built:

1. Open the program **IRControlledRobot.bas** in Basic Micro IDE from your CD-ROM.
2. Save the program (**File/Save As...**) to a location on you hard drive.
3. Connect the serial connector to the motherboard's serial port
4. Click on Program to download the program onto your Viper's microcontroller on the motherboard.
5. Disconnect the serial connector.

Press the direction arrows on your remote control to drive your Viper.

5. Building your Infrared Viper Robot

```
pulse5:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse6
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse6'
;Add a high bit 12

pulse6:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse7
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse7'
;Add a high bit 11

pulse7:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse8
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse8'
;Add a high bit 10

pulse8:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse9
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse9'
;Add a high bit 9

pulse9:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse10
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse10'
;Add a high bit 8

pulse10:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse11
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse11'
;Add a high bit 7

pulse11:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then pulse12
  irdata = irdata + %100000000000
;As above
;If less than 1000uS go to 'pulse12'
;Add a high bit 6

pulse12:
  pulsins IR_DATA_PIN,0,WaitEnd,1,pulselength
  if pulselength < 400 then WaitEnd
  if pulselength < 1000 then endreadir
  irdata = irdata + %100000000000
  endreadir:
  return
;As above
;If less than 1000uS end
;Add a high bit 5

TestIRData:
  High LED_Pin
  if irdata = %0000100100000000 then forward
  if irdata = %1000100100000000 then backward
  if irdata = %0100100100000000 then twistright
  if irdata = %1100100100000000 then twistleft
  if irdata = %0010100100000000 then still
  return
;Set LED_Pin high

forward:
  Serout LeftMotor,i2400,["A",255]
  Serout RightMotor,i2400,["C",255]
  pulsins IR_DATA_PIN,0,ForwardEnd,1,headerlength
  goto Forward
;whilst IR data is still coming in
;keep in this loop

ForwardEnd:
  gosub Still
  return

backward:
  Serout LeftMotor,i2400,["C",255]
  Serout RightMotor,i2400,["A",255]
  pulsins IR_DATA_PIN,0,BackwardEnd,1,headerlength
  goto Backward
;As above

BackwardEnd:
  gosub Still
  return
```

5. Building your Infrared Viper Robot

5.4 The Code

```

; **VARIABLES**
temp                var                word
irdata              var                word
headerlength        var                word
pulselength         var                word
leftspeed           var                word
rightspeed          var                word

; **CONSTANTS**
IR_DATA_PIN         con                P14
LeftMotor           con                P13
RightMotor          con                P15
LED_Pin             con                P6

; **Initialise Motor Outputs**
High LeftMotor
High RightMotor
pause 500

Main:
low LED_Pin
gosub wait                    ;Call the wait subroutine
if irdata <> 0 then gosub TestIRData
irdata = 0
goto Main

wait:
irdata = 0                    ;clear irdata
pulsin IR_DATA_PIN,0,WaitEnd,1,headerlength ;wait for a pulse, if one comes
                                           ;put the length in 'headerlength'
if headerlength > 2250 then readir ;check if the pulse is greater
                                           ;than 2250uS if so goto 'readir'

WaitEnd:
irdata = 0                    ;clear irdata
return

;Start reading the pulses=====
readir:
if headerlength > 2750 then WaitEnd ;check if the pulse is less than
                                           ;2750uS, if so goto 'WaitEnd'

pulse1:
pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength ;wait for the next pulse and
                                           ;store the length in 'pulselength'
if pulselength < 400 then WaitEnd ;if 'pulselength' is less than
                                           ;400uS then goto 'WaitEnd'
if pulselength < 1000 then pulse2 ;if 'pulselength' is less than
                                           ;1000uS then goto 'pulse2'
irdata = irdata + %1000000000000000 ;Add a high bit 16

pulse2:
pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength ;As above
if pulselength < 400 then WaitEnd
if pulselength < 1000 then pulse3 ;If less than 1000uS go to 'pulse3'
irdata = irdata + %1000000000000000 ;Add a high bit 15

pulse3:
pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength ;As above
if pulselength < 400 then WaitEnd
if pulselength < 1000 then pulse4 ;If less than 1000uS go to 'pulse4'
irdata = irdata + %1000000000000000 ;Add a high bit 14

pulse4:
pulsin IR_DATA_PIN,0,WaitEnd,1,pulselength ;As above
if pulselength < 400 then WaitEnd
if pulselength < 1000 then pulse5 ;If less than 1000uS go to 'pulse5'
irdata = irdata + %1000000000000000 ;Add a high bit 13
```

5. Building your Infrared Viper Robot

twistright:

```
Serout LeftMotor,i2400,["A",255]
Serout RightMotor,i2400,["A",255]
pulsin IR_DATA_PIN,0, TwistRightEnd,1,headerlength ;As above
goto TwistRight
```

TwistRightEnd:

```
gosub Still
return
```

twistleft:

```
Serout LeftMotor,i2400,["C",255]
Serout RightMotor,i2400,["C",255]
pulsin IR_DATA_PIN,0, TwistLeftEnd,1,headerlength ;As above
goto TwistLeft
```

TwistLeftEnd:

```
gosub Still
return
```

still:

```
pulsout LeftMotor,6000 ;Brake Left Motor
pulsout RightMotor,6000 ;Brake Right Motor
pause 50
return

end
```

6. What now?

6.1 More Programming Information

There is so much more to the BASIC ATOM than what has been cover here. See the full ATOM programming manual under the 'Help' menu in the BASIC MICRO IDE.

6.2 Cleaning up the edges

You may find that the pieces that have been pushed out of the sheets have rough edges. Find a small file (a metal nail file will do) and gently file the edges smooth to perfect your robots..

6.3 Variations

You now know enough about programming to make your own variations of the models provided. Experiment with a range of robot designs and write or adapt the programs to drive your new robots..

6.3 Variations

Want more? Go online to www.microbric.com to find out the latest news in Microbric Robotics! Join an online forum. Find out the newest models and designs. Buy additional pieces or replace missing ones. Check out the FAQs to troubleshoot problems, or contact the Microbric team.