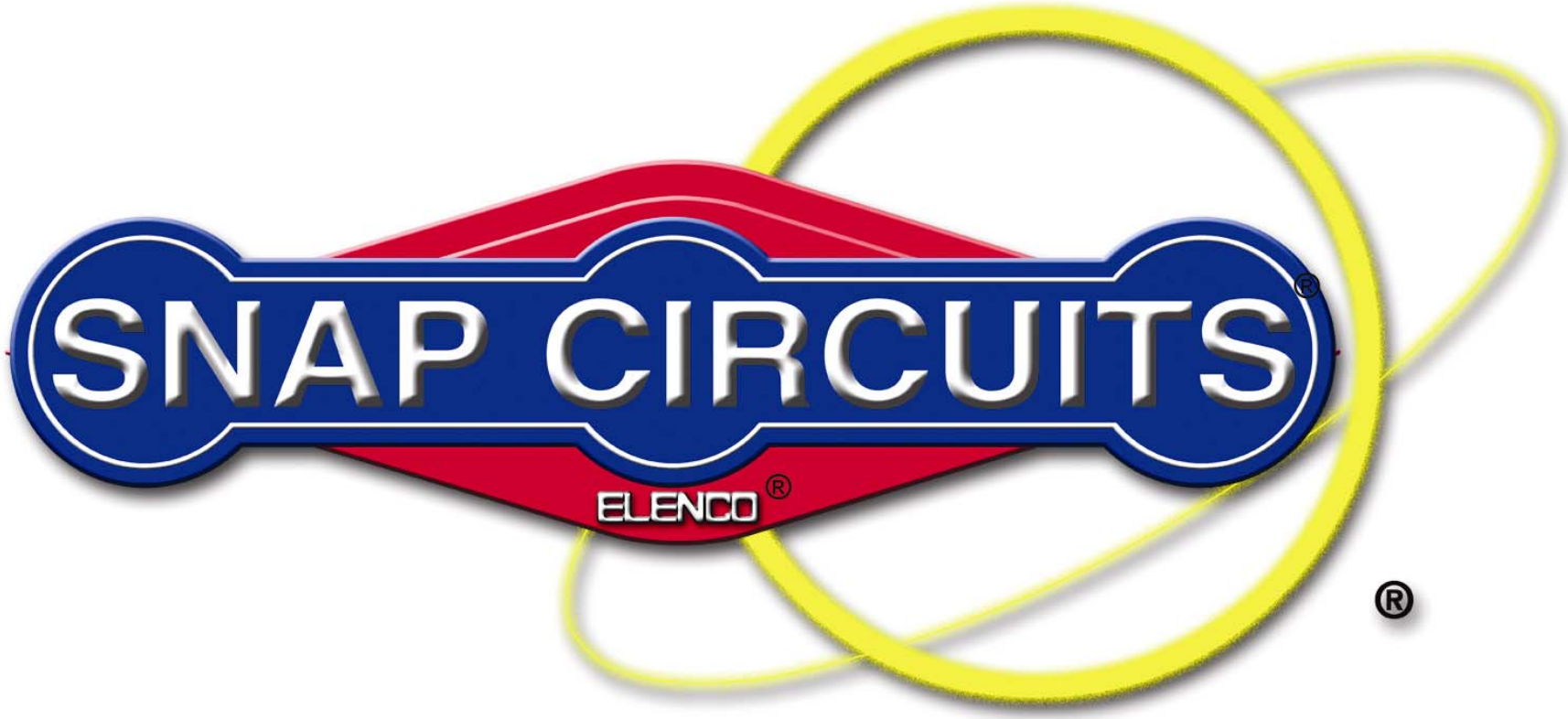


SNAP-MICRO™ I

Instruction manual

Elenco® Electronics Inc.

Rev B

Index to Pages

* Audacity is a registered trademark of Dominic M Mazzoni, South Pasadena, CA

About this manual

The Snap Circuit Micro-Controller manual is designed to quickly move the user into the world of micro-controllers without any heavy mathematics or science background. All that is required is a computer with a Windows® operating system and the Elenco® SCM400 starter kit.
The manual is divided into four separate sections:

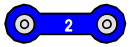
Section 1 - Getting Started (Electronic Components)
Section 2 – Flow chart programming and Snap Circuit® basics
Section 3 - Programming for Snap Circuits®.
Section 4 – Audacity® and Sound Circuits.

The first section provides general information for getting started with Snap Circuits® and the program editor. No prior understanding of micro-controllers is required. Most electronic components will be explained using comparisons to easy to understand water pipe systems. In Section 2, a series of easy to follow tutorials introduce the main features of both Flow Chart programming and the Snap Circuit® system. In Section 3, the programming is extended to control some clever and practical Snap Circuits®. In Section 4 an audio recording and editing program is introduced and used to gather data. This data is then used to control a light with the sound of a clap. The software used for programming the micro-controller is called the 'Programming Editor'. The software and the programs that match this manual may be found at Elenco.com/downloads

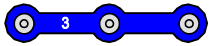
For more specific information on flow chart programming, syntax and examples of each BASIC Command please see sections 1 & 2 'Getting Started & BASIC Commands' located in the help file of the programming editor. For more advanced micro-controller circuits that do not use Snap Circuits®, and example programs, please see 'Interfacing Circuits' in the help file of the programming editor. If you have a question about any command please post a question on the forum at **www.picaxe.co.uk**
For more information on Snap Circuits® or electronic components please run the water pipe analogies included on the Snap Circuits® CD or visit the websites at **www.elenco.com.**

SECTION 1: ELECTRONIC COMPONENTS

Single Spacer (4)

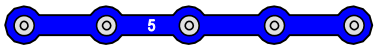


2 Space connector (9)



3 Space Connector (4)

4 Space Connector (3)



5 Space Connector (1)

6 Space Connector (1)

7 Space Connector (1)

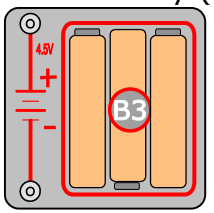
Resistor 1000 Ohms or 1k Ohms (2)

Resistor 10,000 Ohms or 10k Ohms (2)

Resistor 100 Ohms (1)

Resistor 100,000 Ohms or 100k Ohms (1)

4.5 Volt Battery (1)

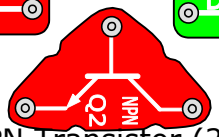


LED Color Red (1)

LED Color Green (1)

Slide Switch (1)

Pushbutton Switch (1)

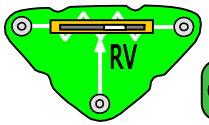


NPN Transistor (2)

Light Dependent Resistor (1)

microphone (1)

Variable Resistor (1)

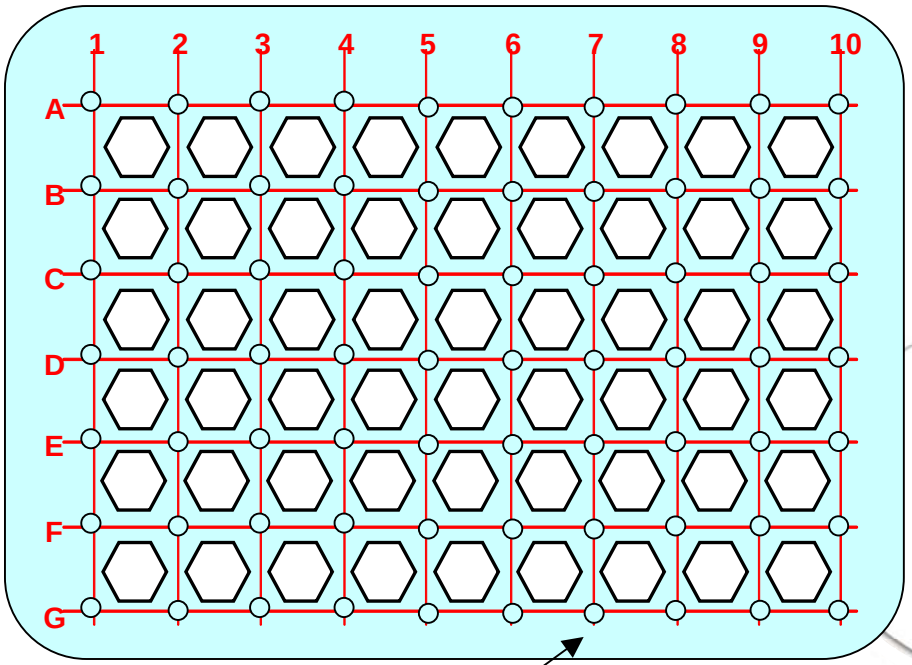


8 Ohm speaker (1)

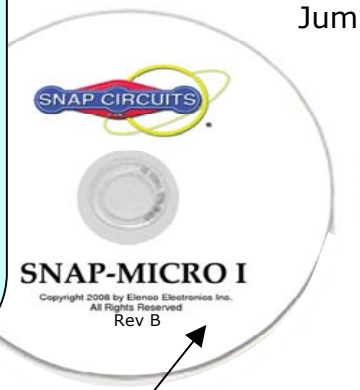
100uF Capacitor (1)

.1uF Capacitor (1)

Jumper Wires (2) Black & Red

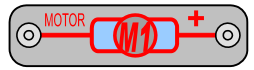


Base Grid Clear (1)



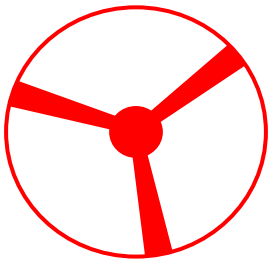
Elenco® CD Rev B (1)

USB Programming Cable (1)

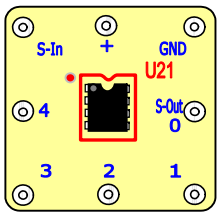


DC Motor (1)

Fan Blade (1)



Computer Interface Cable (1)



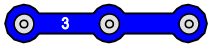
8 Pin Socket U21 with micro installed (1)

ELECTRONIC COMPONENTS

First, consider voltage to be pressure on electrons to make them move in a wire. This is similar to water pressure in a pipe to make the water move. For voltage we will use the symbol ‘V’ (volts).

Next consider the movement of electrons in a wire to be similar to the water moving in a water pipe. This movement of electrons (or water) is called current and is represented by the symbol ‘I’ and measured in units called amperes or amps.

Finally, let the friction of the wire (or water pipe) that tries to stop the current from flowing be called resistance. For resistance we will use the letter ‘R’ or the Greek symbol Ω (Ohm).

The ‘# Space Connector’ 

is really a wire or water pipe with close to zero resistance to current flow. The # represents the number of bumps on the grid will be shorted. For example, a ‘3 Space Connector’ covers 2 spaces and shorts (allows current to flow easily between) 3 bumps.

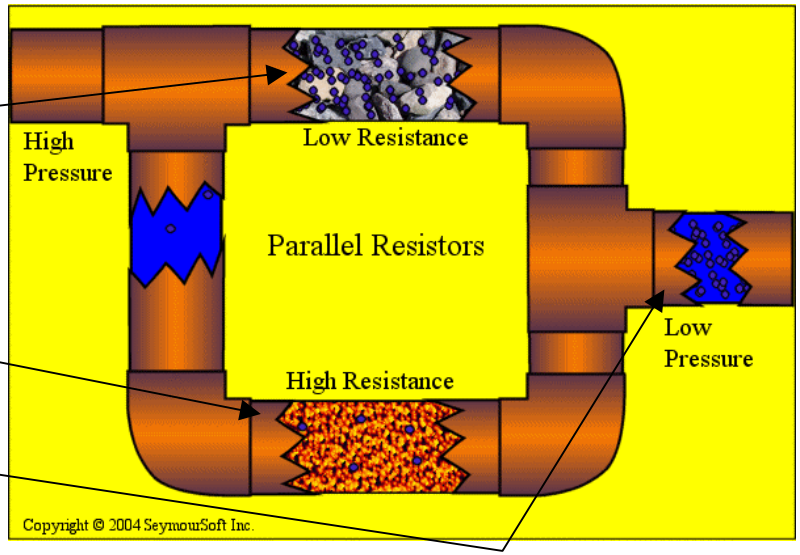
The Resistor 

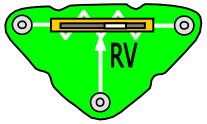
limits the flow of current. The more resistance, the less current will flow at the same pressure applied. For example, if a 10,000 Ohm or 10k Ω resistor is placed across a 4.5 volt battery less current will flow through it than if a 1k Ω was placed across the same battery. To help understand this principal, consider the following;

A water pipe filled with rocks would offer some resistance to the flow of water.

A water pipe filled with sand would offer a greater resistance to the water flow.

Water pipes filled only with water provide almost zero resistance to the flow of water.
Most of the current will take the path of least resistance as shown here.



The Variable Resistor 

'RV' is really a resistor with a wiper arm that can slide from one side of the resistor to the other side. As the wiper moves toward either end, the resistance between the wiper and that end is reduced.

The Switch 'S1' 

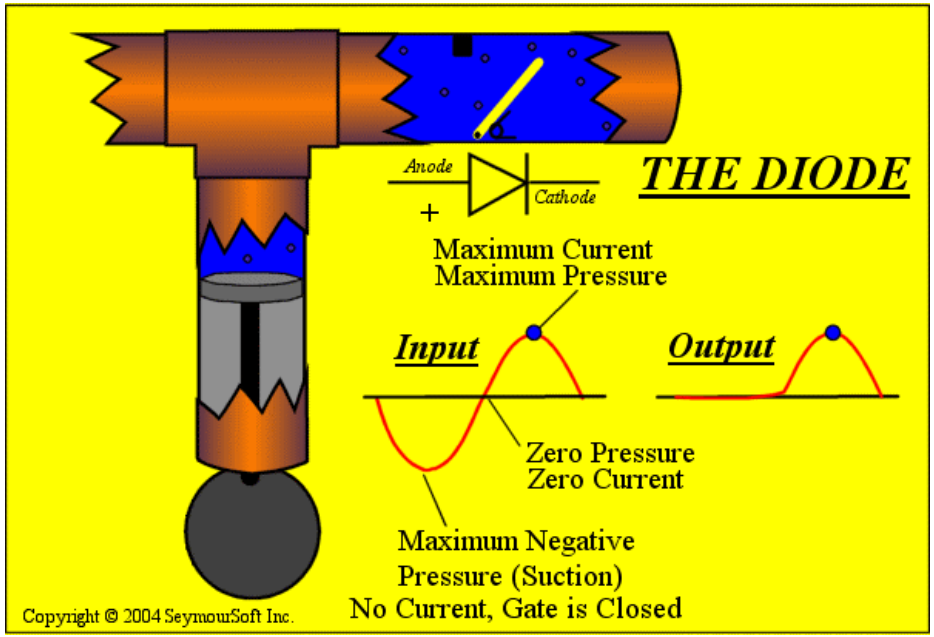
is equivalent to a zero Ohm resistor when it is ON, and an infinite Ohm resistor when it is OFF.

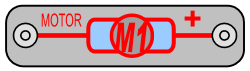
The LED (Light Emitting Diode) 'D1' 

is similar to a check valve in series with a light. Most diodes act similar to water pipe check valves and must be installed in the correct direction for current to flow.

Consider the water pipe check valve shown on the right. When the piston pushes water into the pipe the check valve opens and water flows as shown. When the piston tries to suck water from the pipe the check valve closes and no current flows through the check valve.

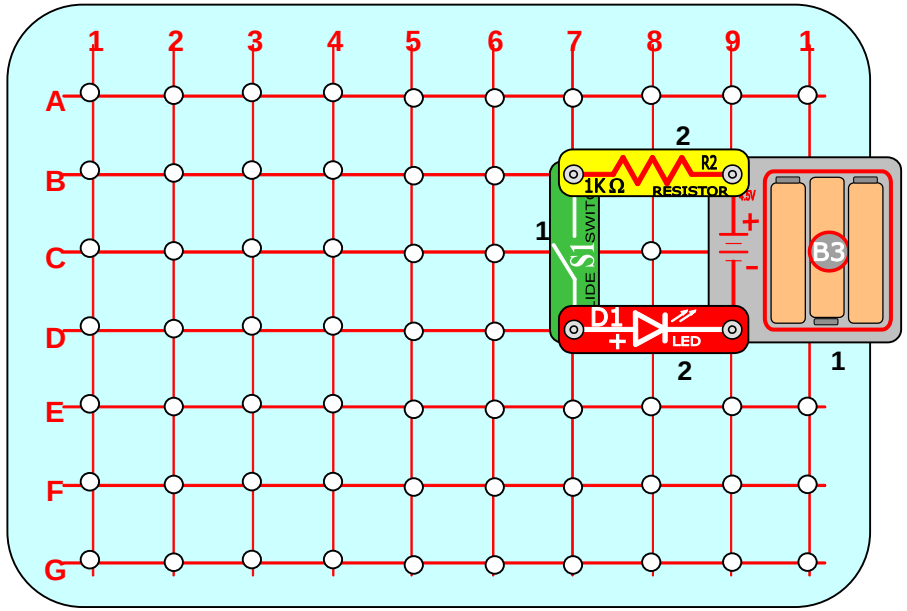
Light is produced whenever current flows through the LED check valve. The stronger the current in the LED, the brighter the light. If the LED is installed with the + symbol connected to the negative voltage or ground the current cannot flow and the LED will be off.



The DC Motor 

converts a DC voltage to a rotation. The direction of the rotation depends on the polarity of the voltage.

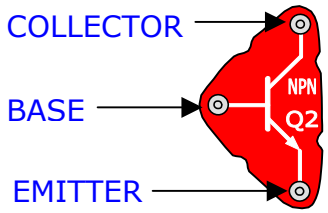
Build the circuit shown below. Snap Circuit® Boards are built one level at a time. The base grid is considered to be level 0. Parts placed directly on the base grid are said to be on level 1 and will have a small black 1 next to the part. Parts placed on level 1 parts are said to be on level 2 and will have a small black 2 next to the part. This process is continued until all parts are installed.



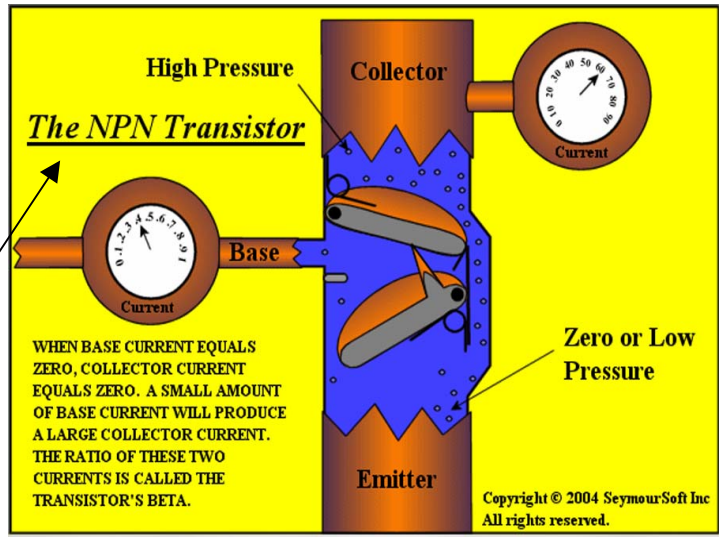
1. Turn Switch S1 to ON and the LED should glow red. Turn S1 to OFF.
2. Turn D1 around so the + is on the switch side and repeat step 1. In this case the LED should be dark and never glow. Turn Switch S1 to OFF and replace LED 'D1' in the original position.
3. Replace the 1k Ω resistor 'R2' with a 100 Ω resistor 'R1'. The LED 'D1' should be brighter when Switch 'S1' is turned ON. Lower resistance produces more current, and more current makes LED's glow brighter.
4. Use this circuit to test LED's, resistors, and switches S1 and S2.

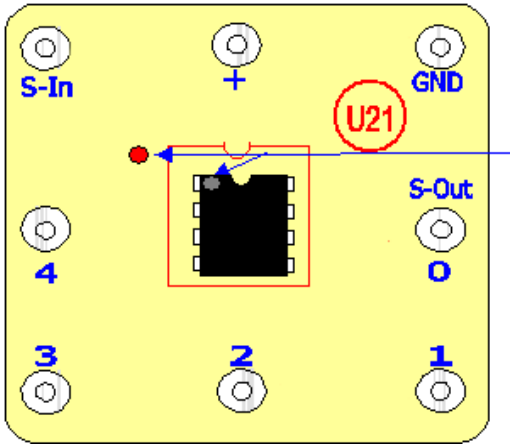
The SPEAKER 'SP' is actually an 8 Ω speaker that can be used to produce audible tones.

The NPN Transistor 'Q2' is a device that amplifies current. For example, a small current from base to emitter will produce a much larger current from collector to emitter. The NPN connections are labeled collector, base, and emitter as shown here.



Similar device in water pipe system





The U21 Snap Circuit® module is designed to accept any 8 pin integrated circuit. If not already installed, remove the 8 pin integrated circuit (micro-controller chip) from its package and carefully install it as shown here

Make sure the dot on the integrated circuit and the red dot on the socket are in the same corner.

The blue numbers printed on the socket platform are the pin numbers used by the programming editor. In all situations we will use these numbers to reference the output or input pin.

WHAT IS A MICRO-CONTROLLER?

A micro-controller is a 'computer-on-a-chip'. It's an integrated circuit that contains memory, logic, processing, and input/output circuitry. Micro-controllers are programmed with specific instructions to control many different devices. Once programmed the micro-controller is built into a product to make the product more intelligent and easier to use.

For example, a microwave oven uses a single micro-controller to process information from the keypad, display user information on a display, and control the turntable motor, light, bell and cooking time.

One micro-controller can often replace a number of separate parts, or even complete electronic circuits.

Applications that use micro-controllers include household appliances, alarm systems, medical equipment, vehicle subsystems, musical instruments, and electronic instrumentation. Some modern cars contain many micro-controllers used for engine management and remote locking.

SECTION 2: PROGRAMMING & SNAP CIRCUIT BASICS

INSTALLING SOFTWARE AND PROGRAMMING CABLE

To install the Programming Editor software you must have an internet connection. Click on the link shown here and then follow instructions. http://www.elenco.com/downloads/SCM400dlinst.pdf

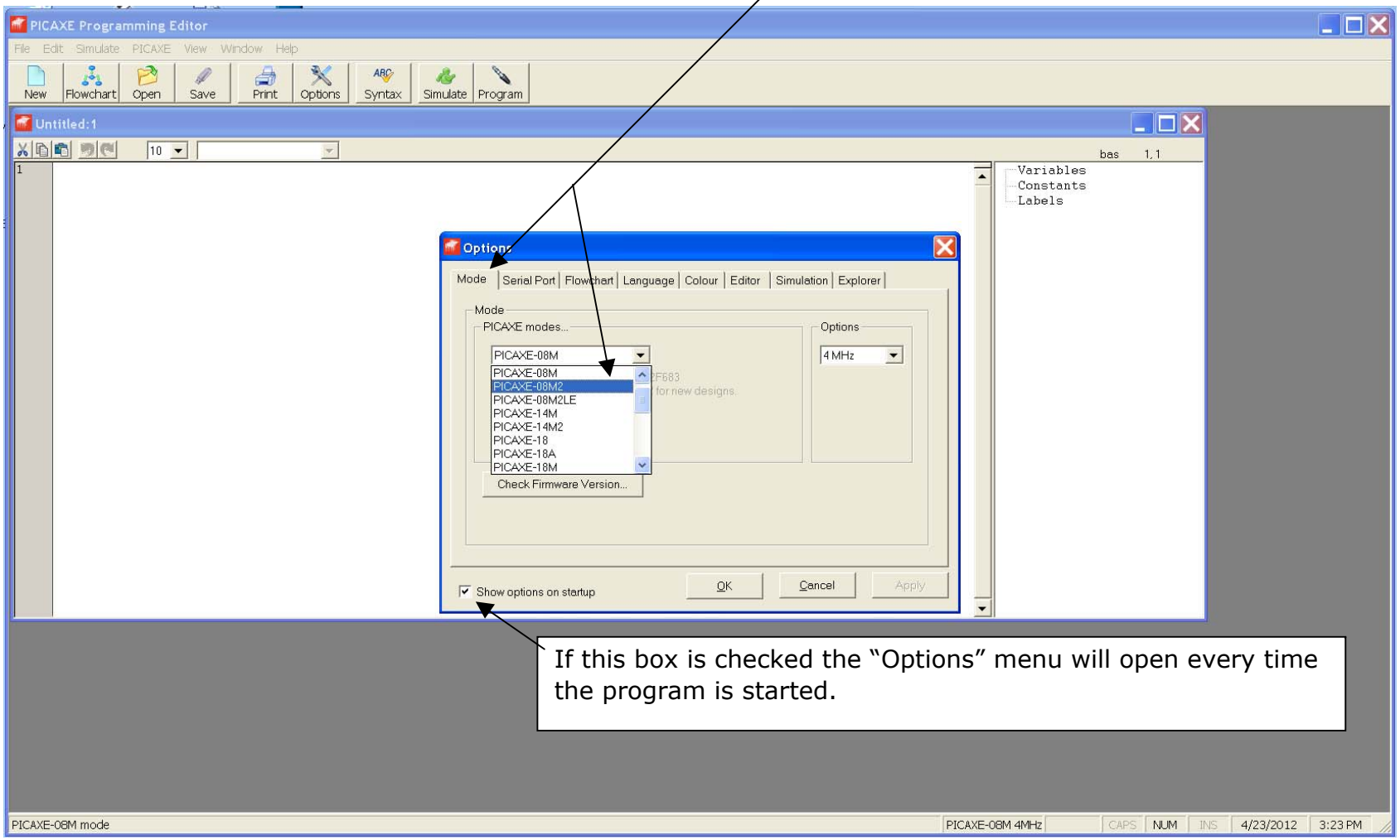
1. Follow instructions that appear to install editor.

2. Insert the USB programming cable into a USB port on your computer. The cable will configure itself automatically. If you have problems with configuration then contact Elenco®.
   www.help@elenco.com

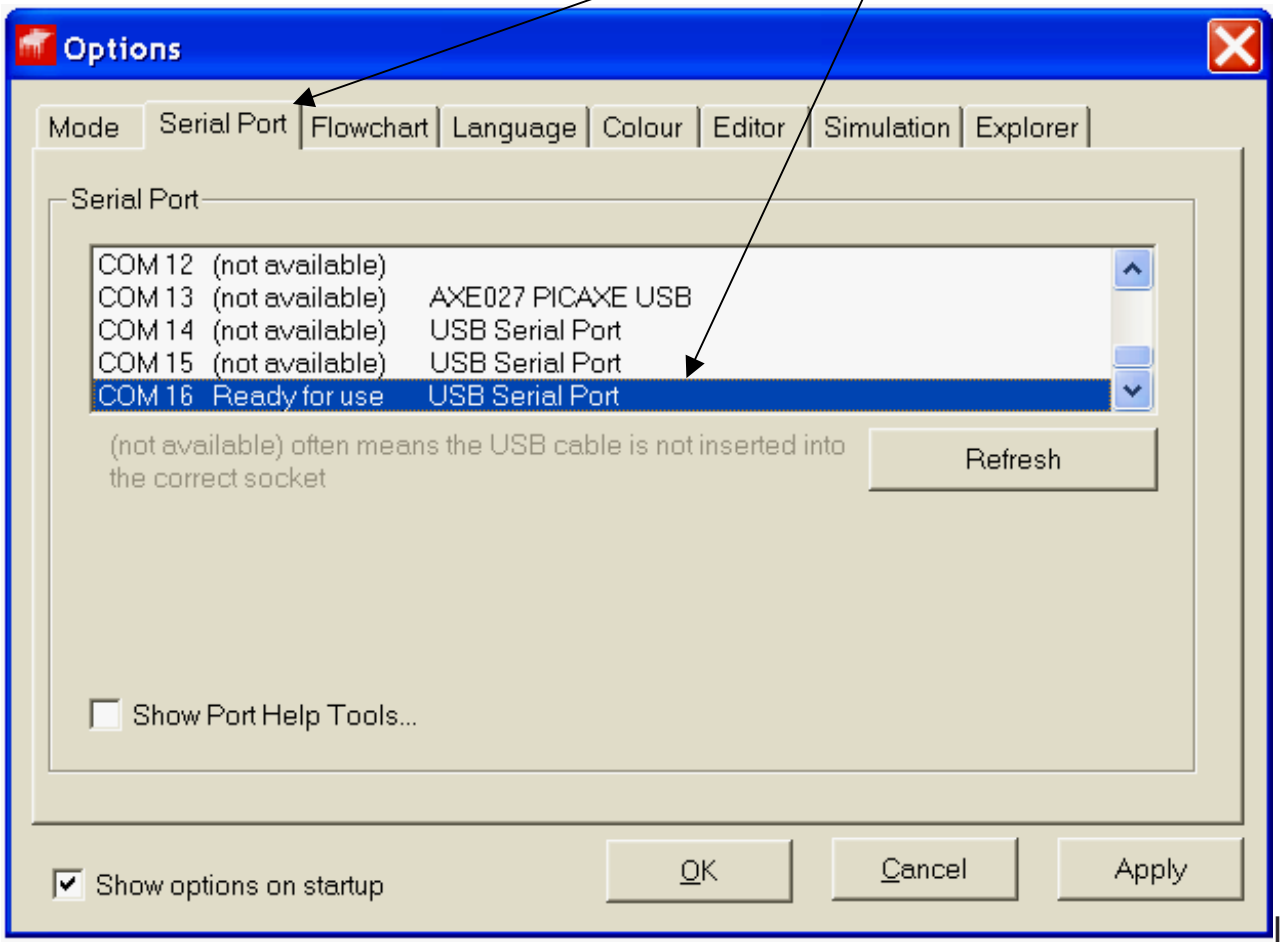
(note: If message appears " looking for TTL232R-3V3" the driver is on the Elenco CD)

3. Start the Programming Editor software by clicking on the Program Editor Icon. If you get the same version shown here the Icon will appear as shown here.

When the program opens click the “view” menu and then the Options menu to display the Options panel (this may also automatically appear on startup). On the ‘Mode’ tab select the 08M2 microcontroller.

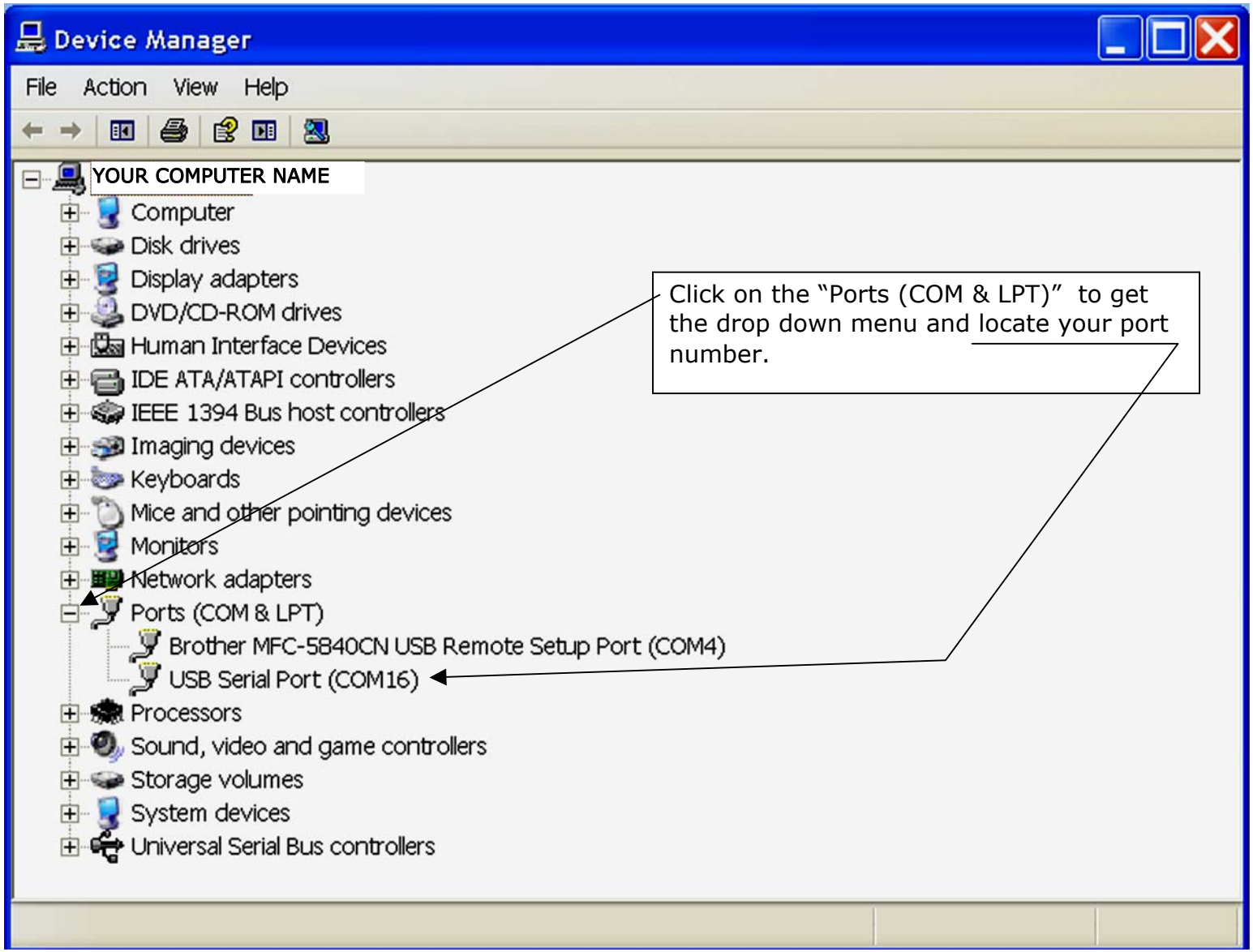


On the ‘Serial Port’ page also select the appropriate serial COM port (the port where you connected the USB programming cable).
COM port should say “Ready to use” If you have COM port trouble see next page.



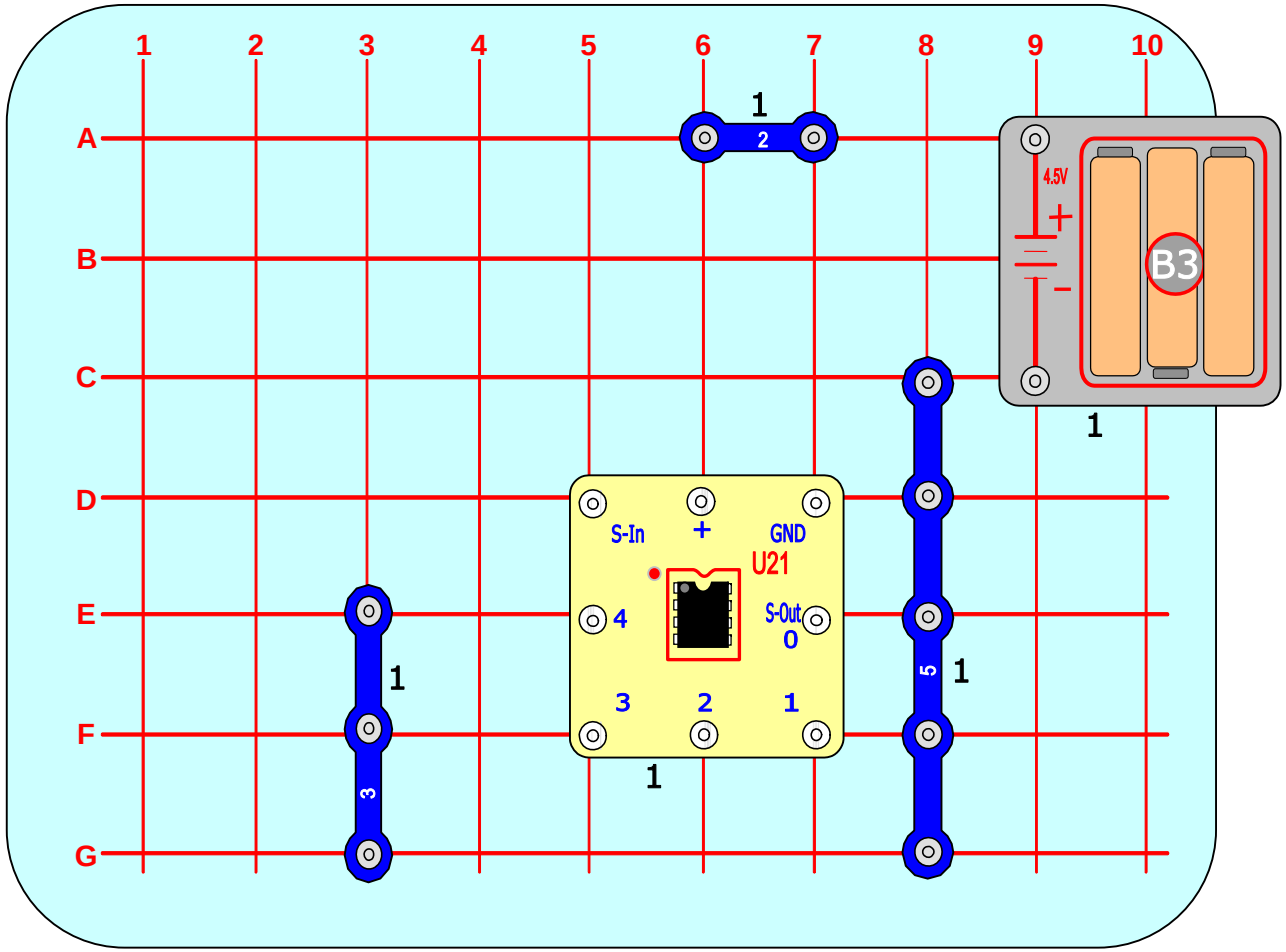
All windows may appear different depending on the contents of the file folders and the version of windows being used. The “Options” window should remain as shown but with different COM numbers depending on the port or plug used. If cables are move to different ports the option window may have to be changed to match the port being used.

If you do not know your COM port number, you can find it by clicking “START” on main screen, then click “Run” and type in devmgmt.msc and press enter key. The following screen should open.

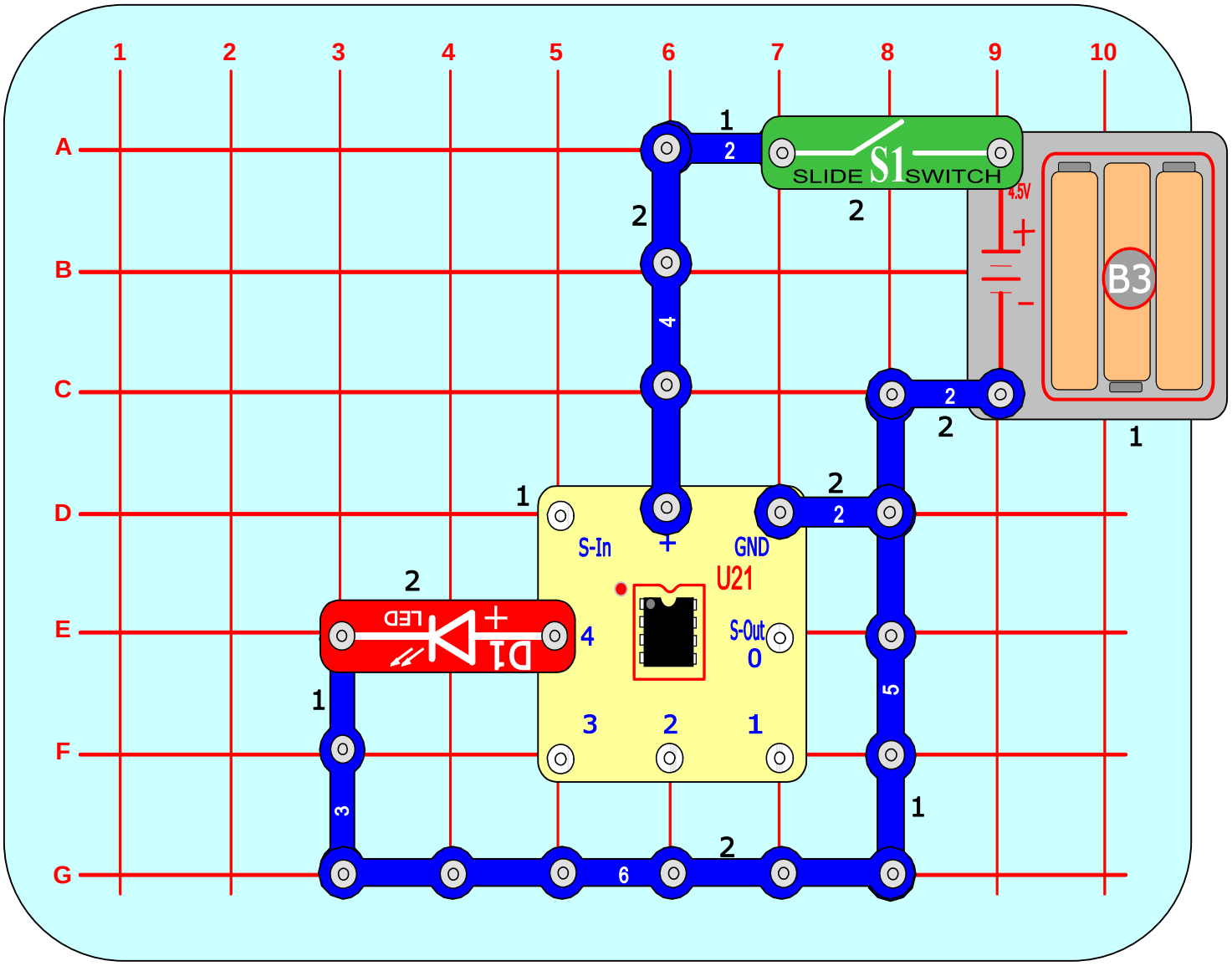


BUILDING THE MICRO CONTROLLER CIRCUIT

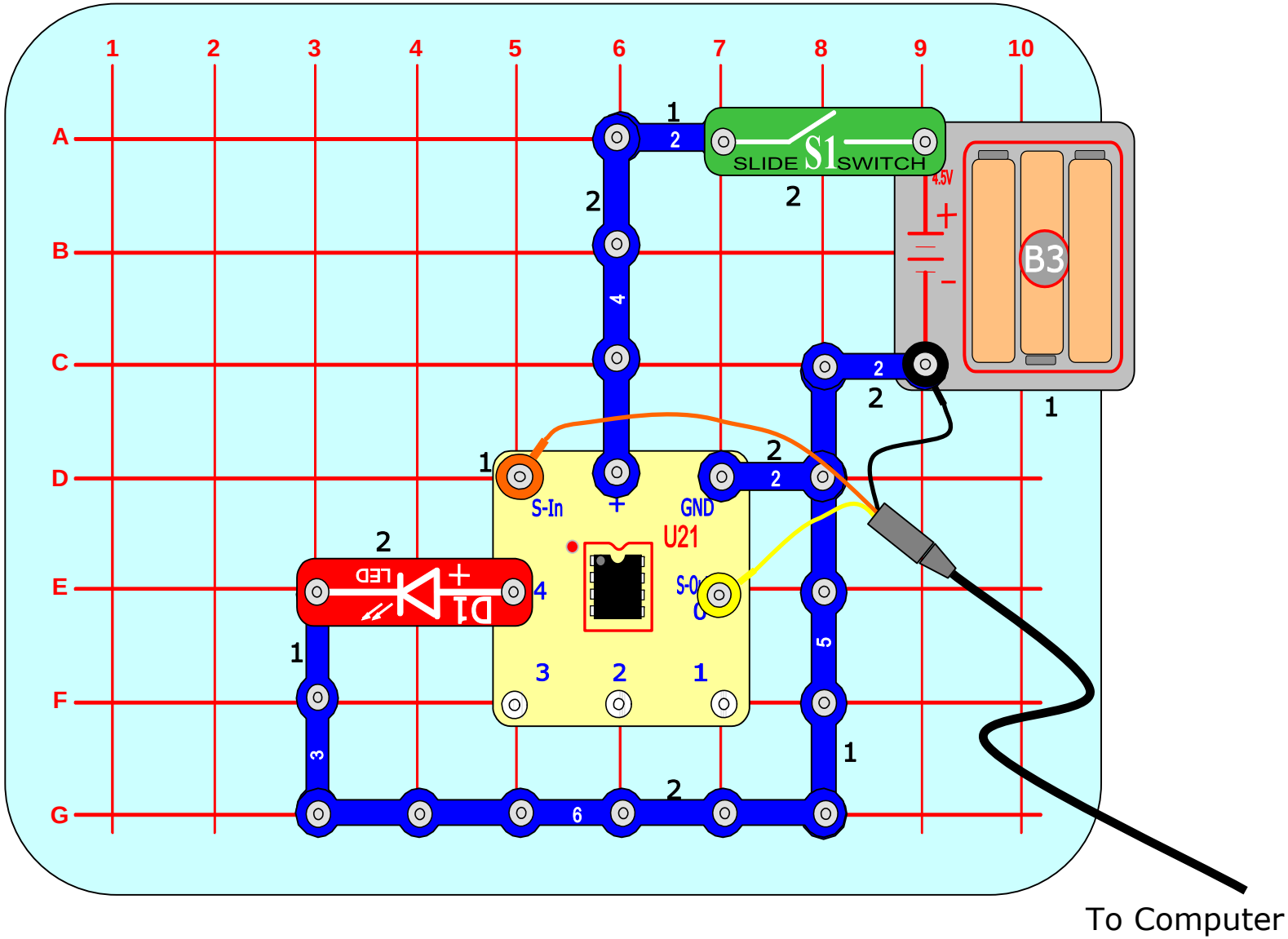
1. Snap Circuit® Boards are built one level at a time. The 70 post base grid is considered level 0.
2. Parts placed directly on the base grid are said to be on level 1 and will have a small black 1 next to the part.
3. Parts placed on level 1 parts are said to be on level 2 and will have a small black 2 next to the part.
4. The above process continues until all levels are completed.
5. Build level 1 for the Micro Controller Circuit shown here:



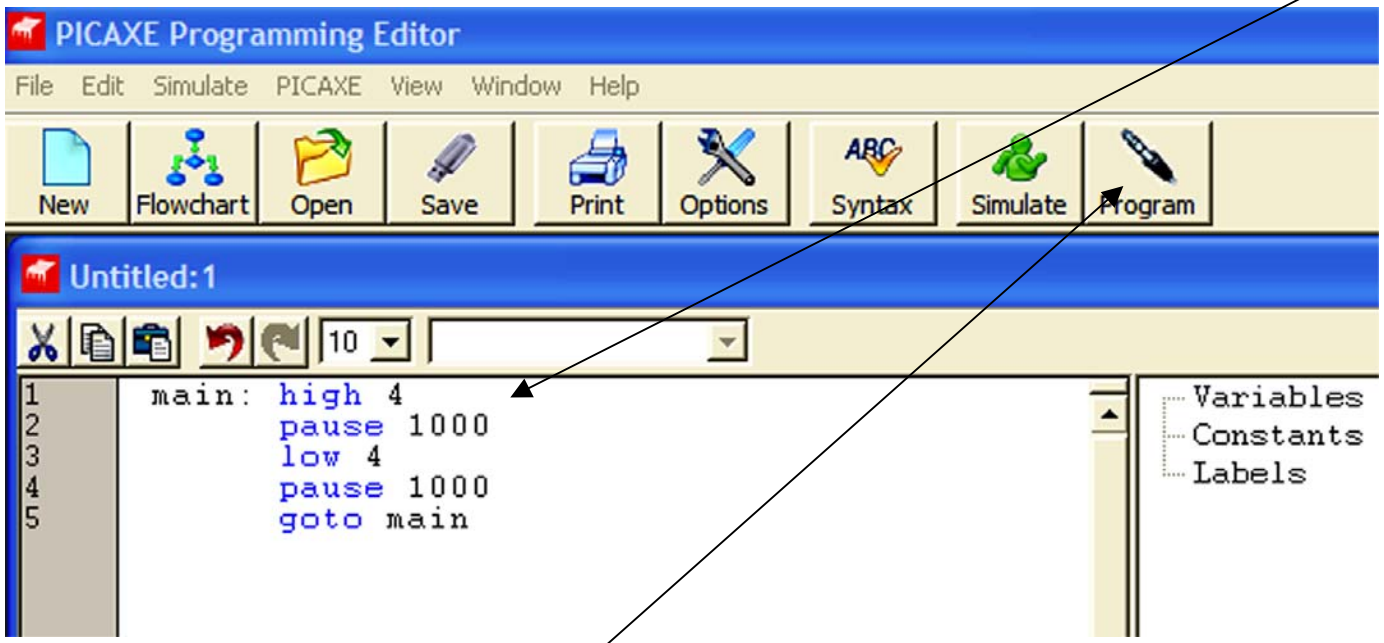
6. Add level two to the grid.



7. Connect the serial cable to the Snap Circuit® Micro as shown below. Make sure the yellow lead is connected to pin 0, the black ground lead is connected to battery minus, and the orange lead to S-In. Turn switch S1 to ON.



8. Using the program editor software, type in the following program:



You can check your program by clicking the Syntax Check button (ABC with a check mark on it). If Syntax is correct proceed to step 9.

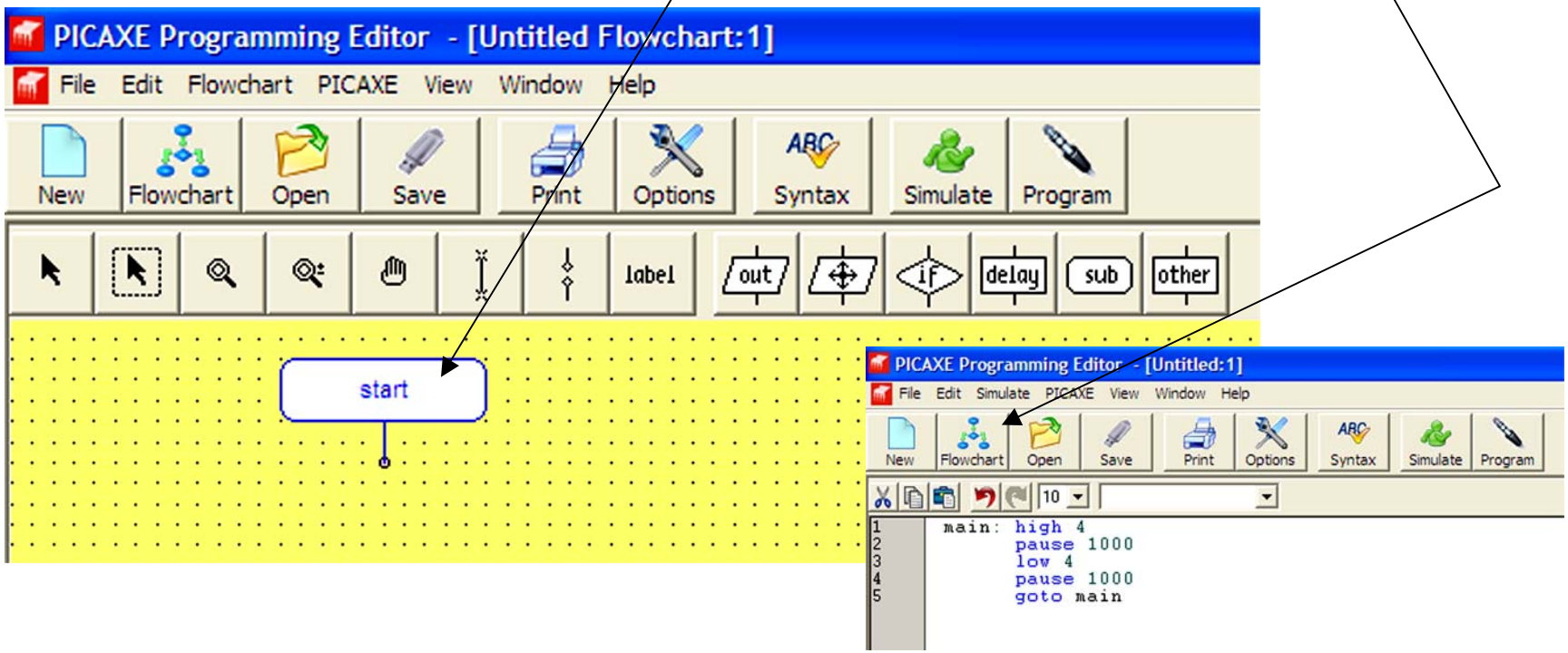
9. Click the Program button to download the program to the hardware. If problem in programming, turn power switch S1 off, click Program button again, and quickly turn S1 switch to on. After the download the output LED should flash on and off every second. Congratulations! You have just programmed a micro-controller integrated circuit to make an LED blink. Do not remove parts from the grid they will be used in the next section. You may remove the Programming Cable now. Take note that turning the S1 switch ON and OFF does not lose the programming. Turn the switch off to save battery power.

FLOW CHART PROGRAMMING AND SNAP CIRCUIT® BASICS.

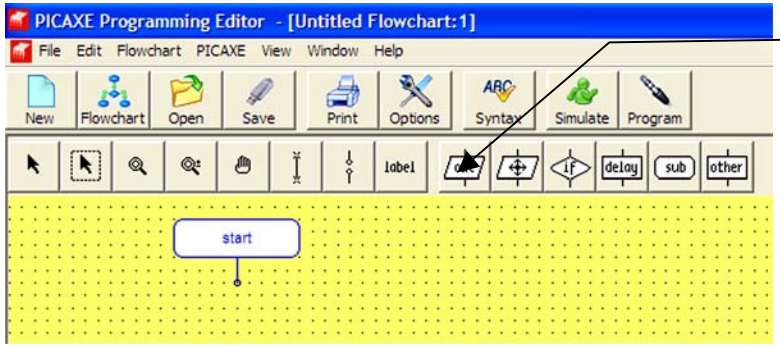
PROJECT 1:

In Section 1 you built the basic Snap Circuit® board that will be used for most of the projects in this section. It is important to have a clear road map when writing software. A Flow Chart will keep the problems at a minimum and allow the programmer to quickly and easily add and remove sections of programming without fatal errors. Open the programming editor and click on the New Flowchart button shown here. Ignore upgrade box if it appears and click hide for future flowcharts.

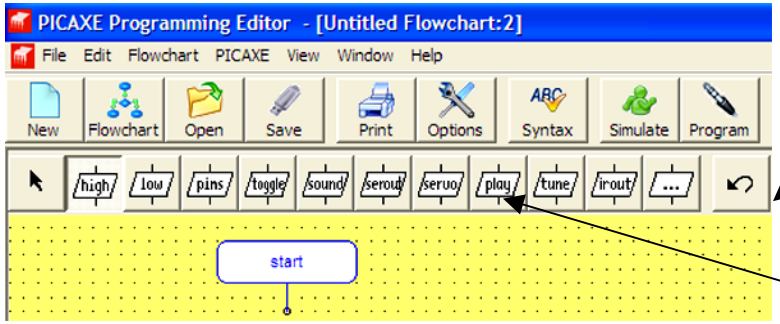
The screen shown here will open and the Start box will be added to the grid.

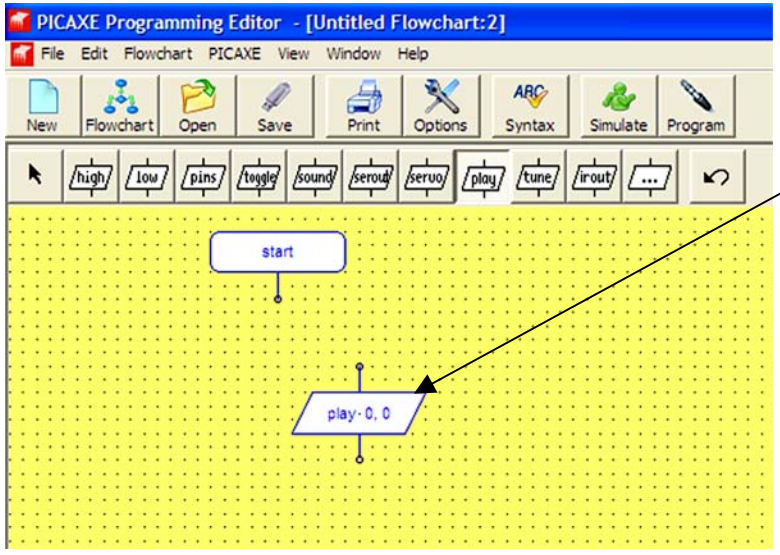


Our first flowchart project will be to play the melody to Happy Birthday with and without flashing lights.



First click the 'out' button. The menu bar will change to this menu bar.

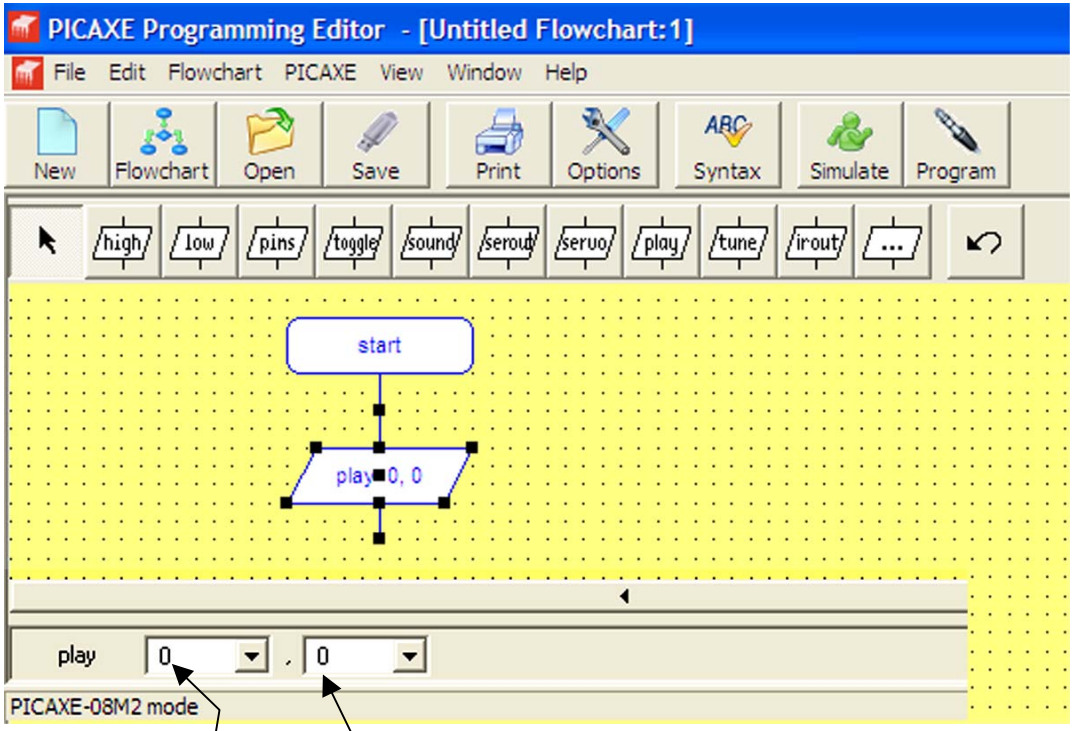




Click on the 'Play' Button and add this box to the chart by dragging down using left mouse button.

Next, click the right mouse button to activate the select arrow ⇖. Select the play box and drag it to the start as shown on next page.

The Play command has the following functions (Syntax)`



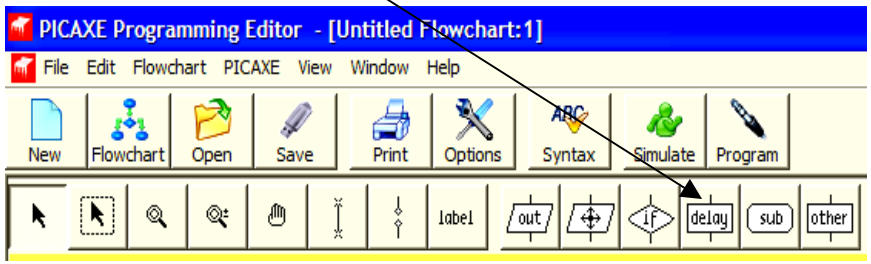
Function:
Play an internal tune on output pin 2.

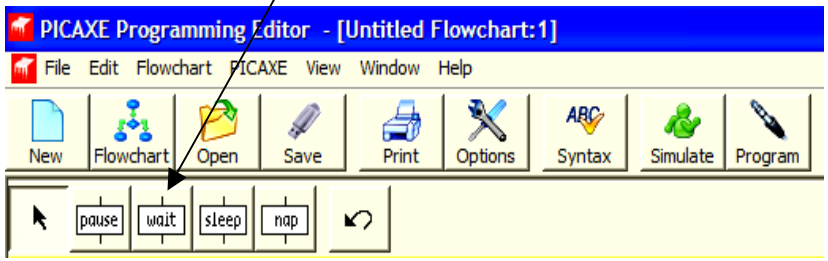
PLAY tune, LED

The **tune** constant (0 - 3) specifies which tune to play as follows;
0 - Happy Birthday
1 - Jingle Bells
2 - Silent Night
3 - Rudolph the Red Nosed Reindeer

LED is a constant (0 -3) that specifies outputs that flash as the tune is being played as follows;
0 - No outputs
1 - Output 0 flashes on and off
2 - Output 4 flashes on and off
3 - Output 0 and 4 flash alternately

The **tune** and **LED** constants can be changed in the edit box at the bottom of the program editor screen when the play box is active. After the tune plays, there should be a few seconds of delay and then the tune should play again. Return to the previous menu by clicking the curved arrow button ↶. Now click the delay button and use the previous technique to add the wait box.

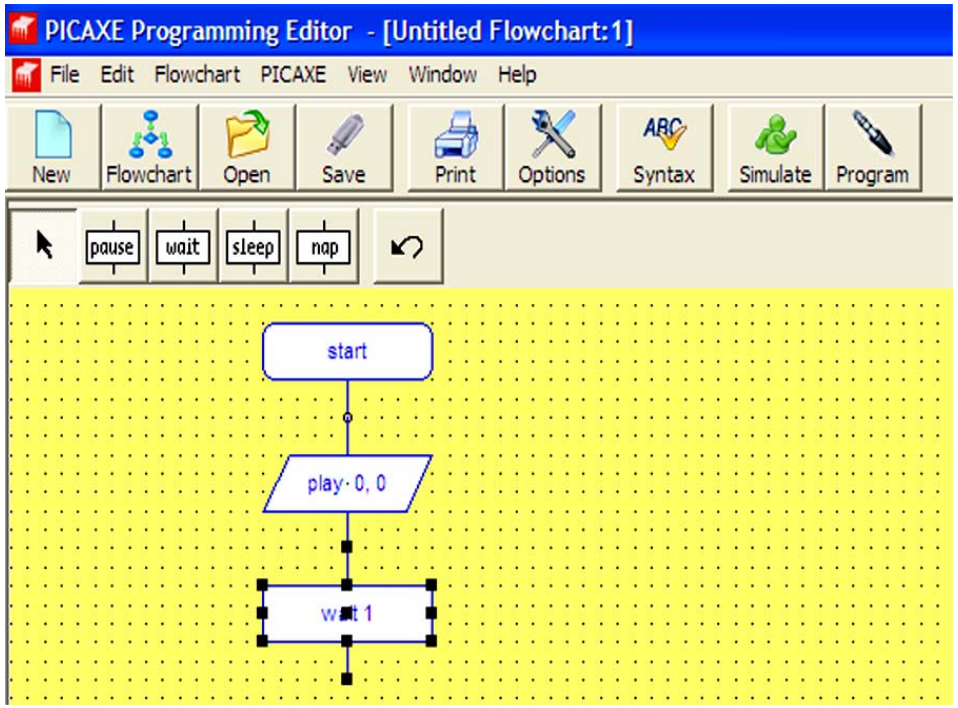




WAIT

Function:

Pause for some time in whole seconds.



Syntax:

WAIT seconds

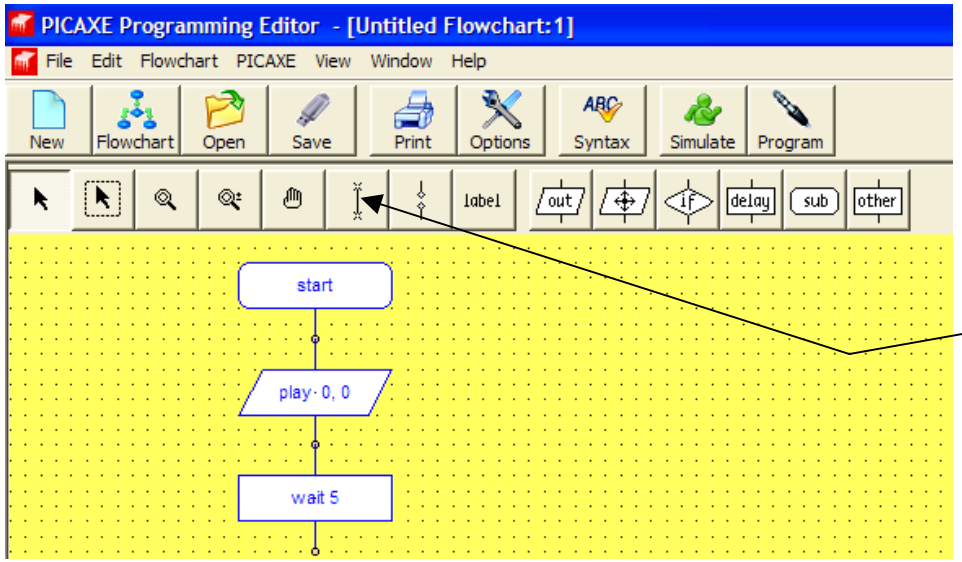
- Seconds is a constant (1-65), which specifies how many seconds to pause.

Information:

This is a 'pseudo' command that is equivalent to 'pause' times 1000, This command cannot be used with variables and is a fixed delay installed during programming. To change the delay due to an input or other information, use the Pause command.

Use the edit box at the bottom left corner of the program editor window to change the time to 5 seconds.

Note: The "Wait" box must be selected.

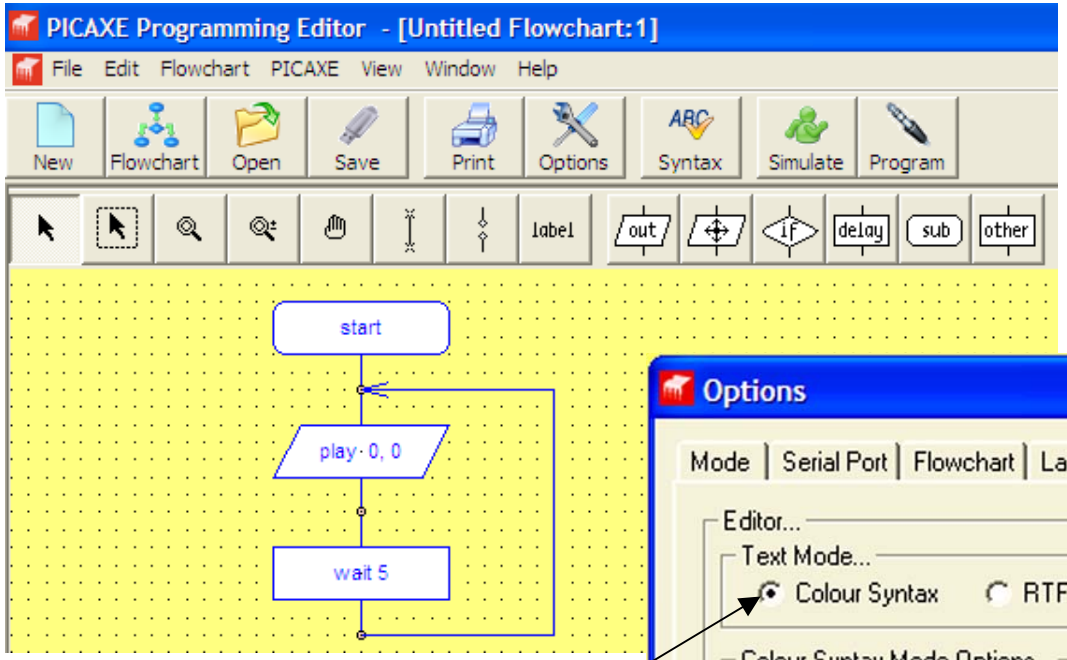


Next, the loop needs to be closed.

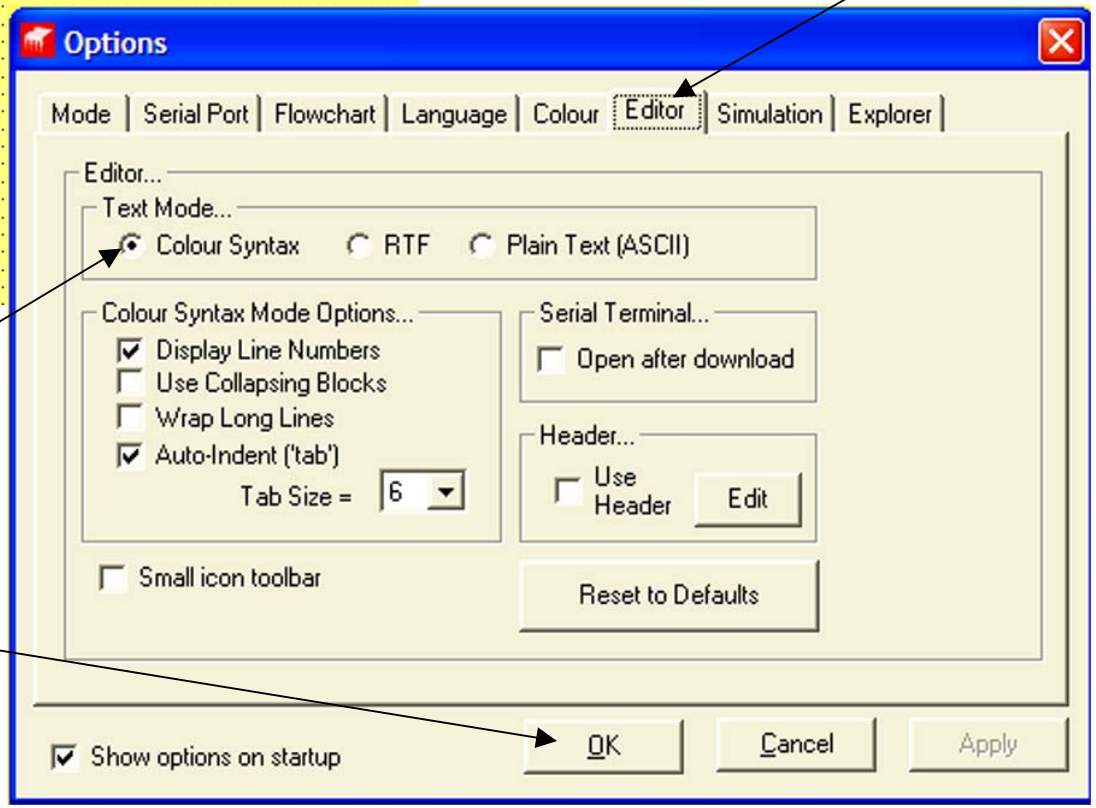
Click the return button to get back to the main menu and select the Draw Lines tool.

Use the 'Draw Lines' tool to make the flowchart on your program editor the same as the one shown on the next page.

TESTING A FLOWCHART PROGRAM



Before running a simulation for the first time the program editor should be adjusted as follows:
First click View in the headings and then select Options. When the window below opens, select the Editor tab.



Set the following:
Text Mode to 'Color Syntax'
Color Syntax Mode Options as shown here.

Exit the options screen by clicking the OK button on the bottom.

Every flowchart program should be tested and saved before it is converted to a down-loadable basic program. Make sure your audio is turn on and at a level you can hear, then click on the Simulate button.

Flowchart Simulation

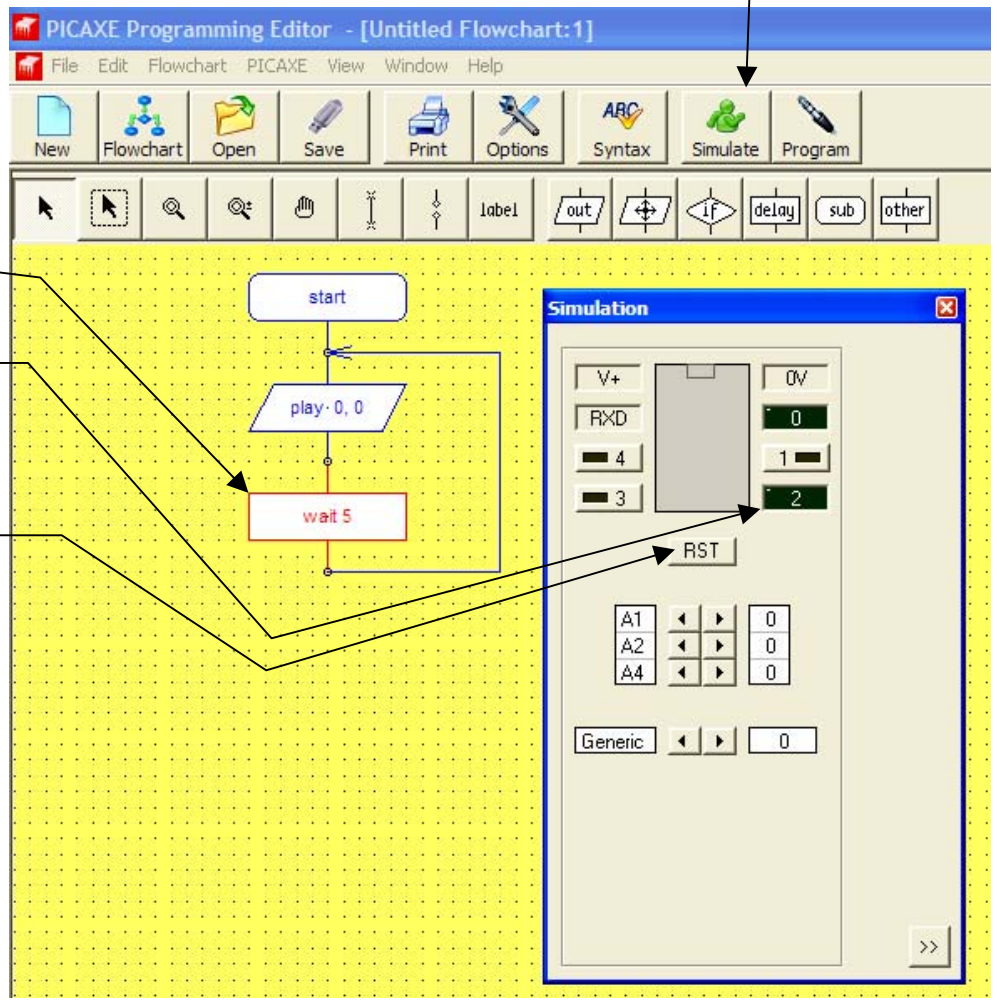
The program will start running and the current block running will be highlighted in red.

The output pin will be black and turn green when the song is playing.

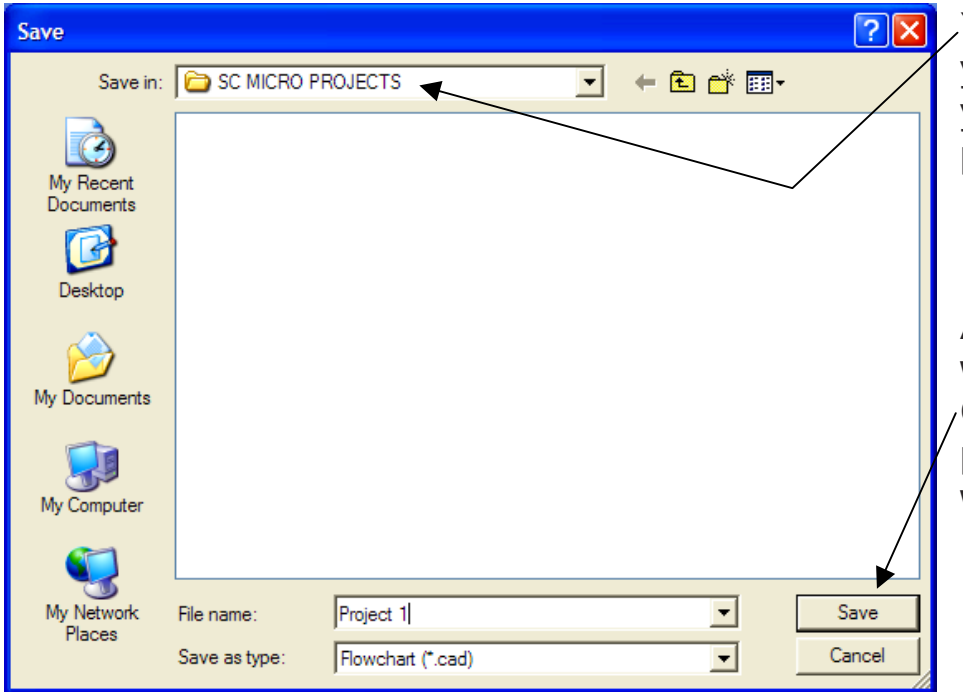
The RST button will reset the program to start when clicked.

The other features of the Flowchart simulator will be explained later as they are used. End the simulation by clicking anywhere on the grid.

After the Flowchart program has been tested it should be saved in a folder on your computer where all your Snap Circuit® programs should be stored. In these examples the folder C:\SC MICRO PROJECTS was created for this purpose.



Open the drop down menu under ‘File’ in the program editor and click on ‘save as’ to save the flowchart program in your Snap Circuit® folder as shown on next page.

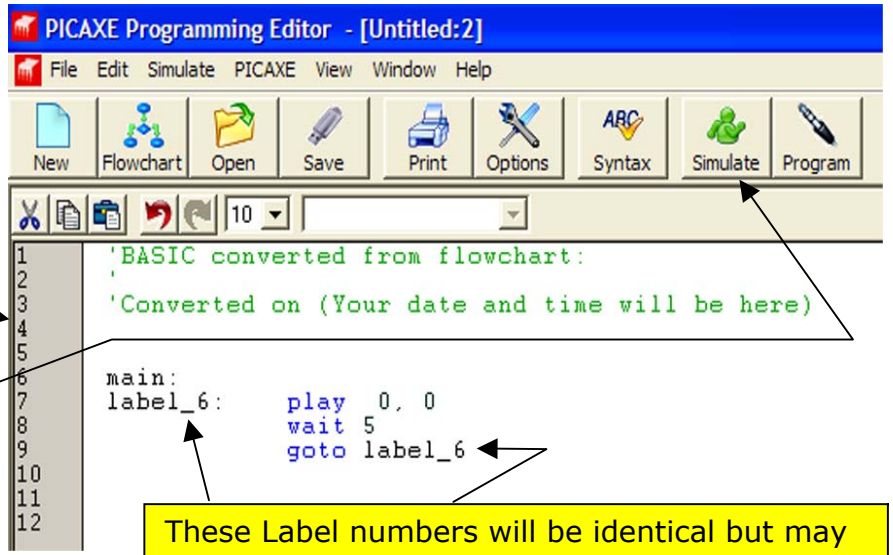


You may want to set up a special folder on your computer to store files as shown here, or you can use the one that comes up when first project is saved.

After filling in the proper information the window should appear as shown on the left. Click on the 'Save' button. The flowchart program will be stored and the save window will close.

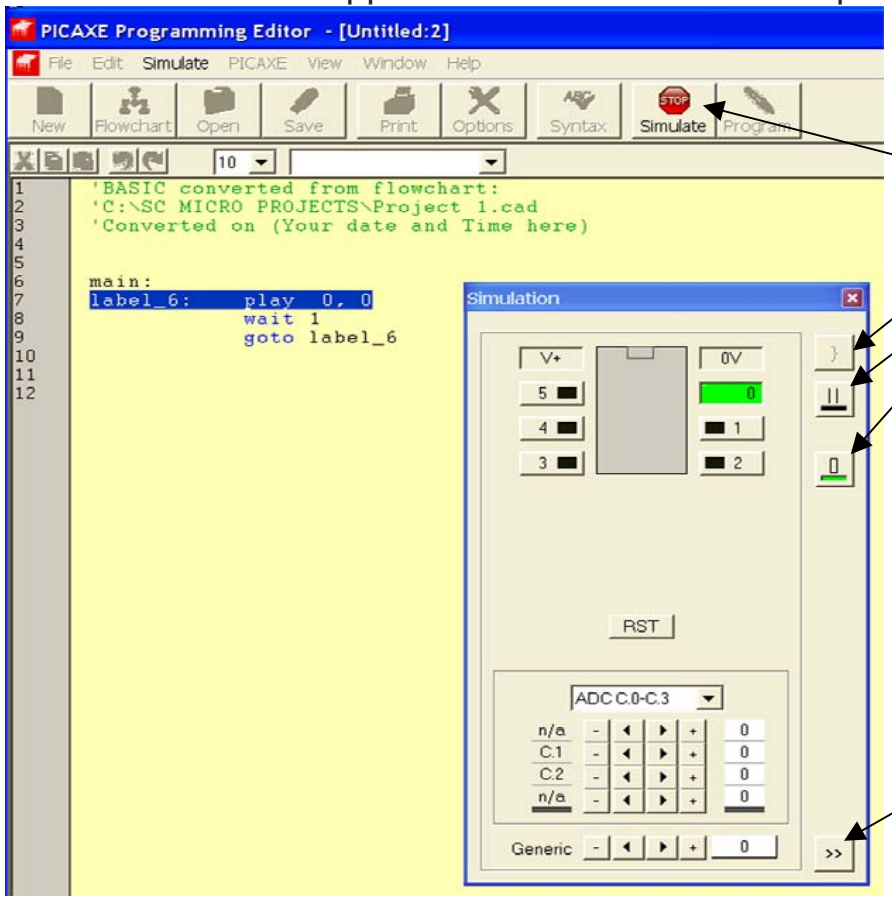
On the program editor screen, under the PICAXE® heading click on the 'Convert Flowchart to Basic …' command. The following screen will appear.

Open the simulator again by clicking on the Simulator button. The song should play.



Program Flow Control and Breakpoints

Three new buttons appear on the main simulation panel. They are shortcut buttons for the Simulation menu functions.



You can stop simulation by clicking Stop button.

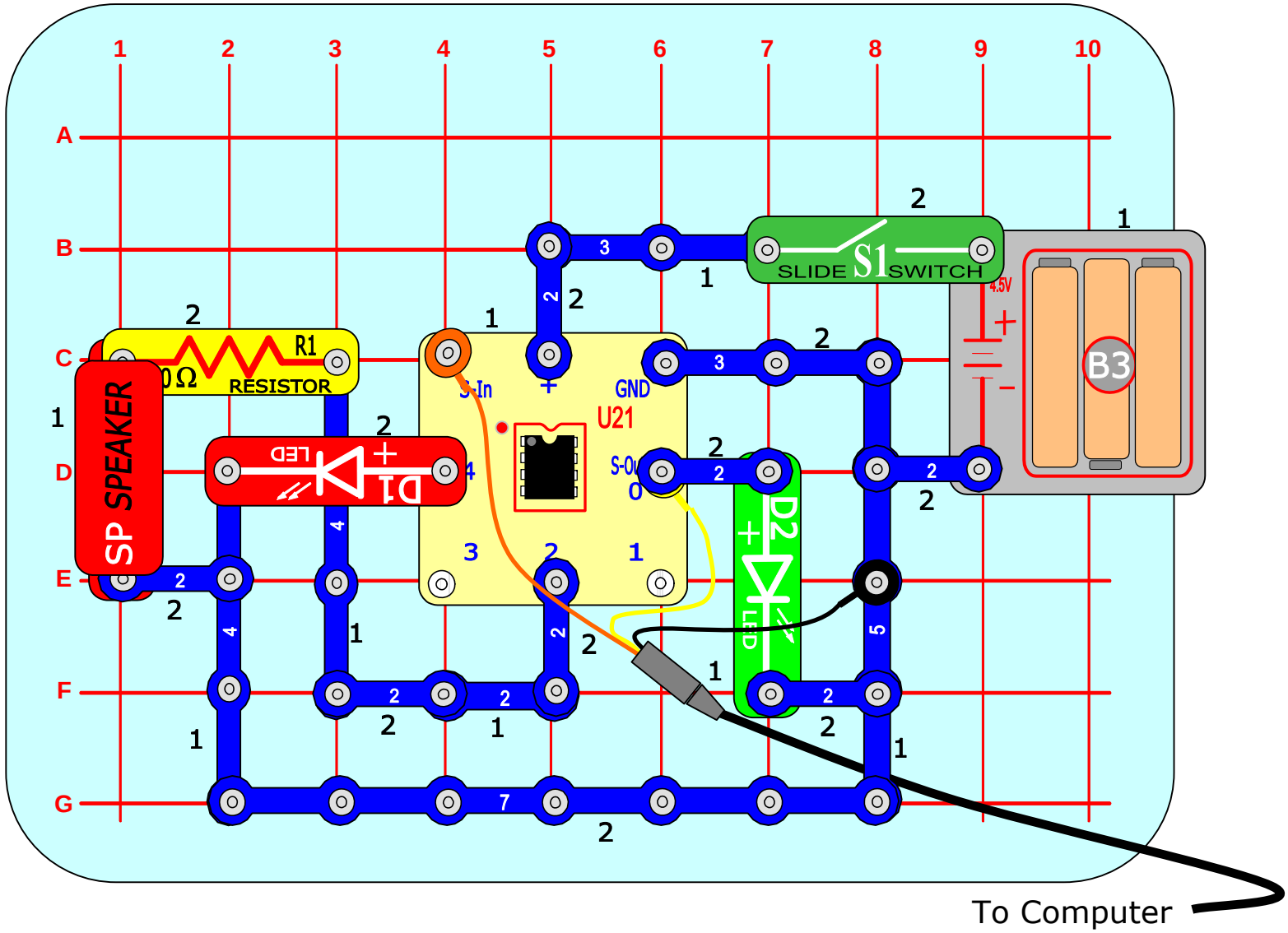
} single step through the simulation
| | pause the simulation at the current line
> start [ ] stop the simulation

Breakpoints can be placed or removed from the program by clicking on the line number in the margin. Alternately the Toggle breakpoint under the Simulate heading may be used to insert/remove a breakpoint at the current cursor position. Breakpoints are indicated by a red bar in the margin.

The >> button displays the variables panel.

Other **Simulation Options** will be discussed and demonstrated in future projects as they are used. Other windows that open on this screen will be discussed later.

Rework your Snap Circuit board to the circuit shown on the next page.

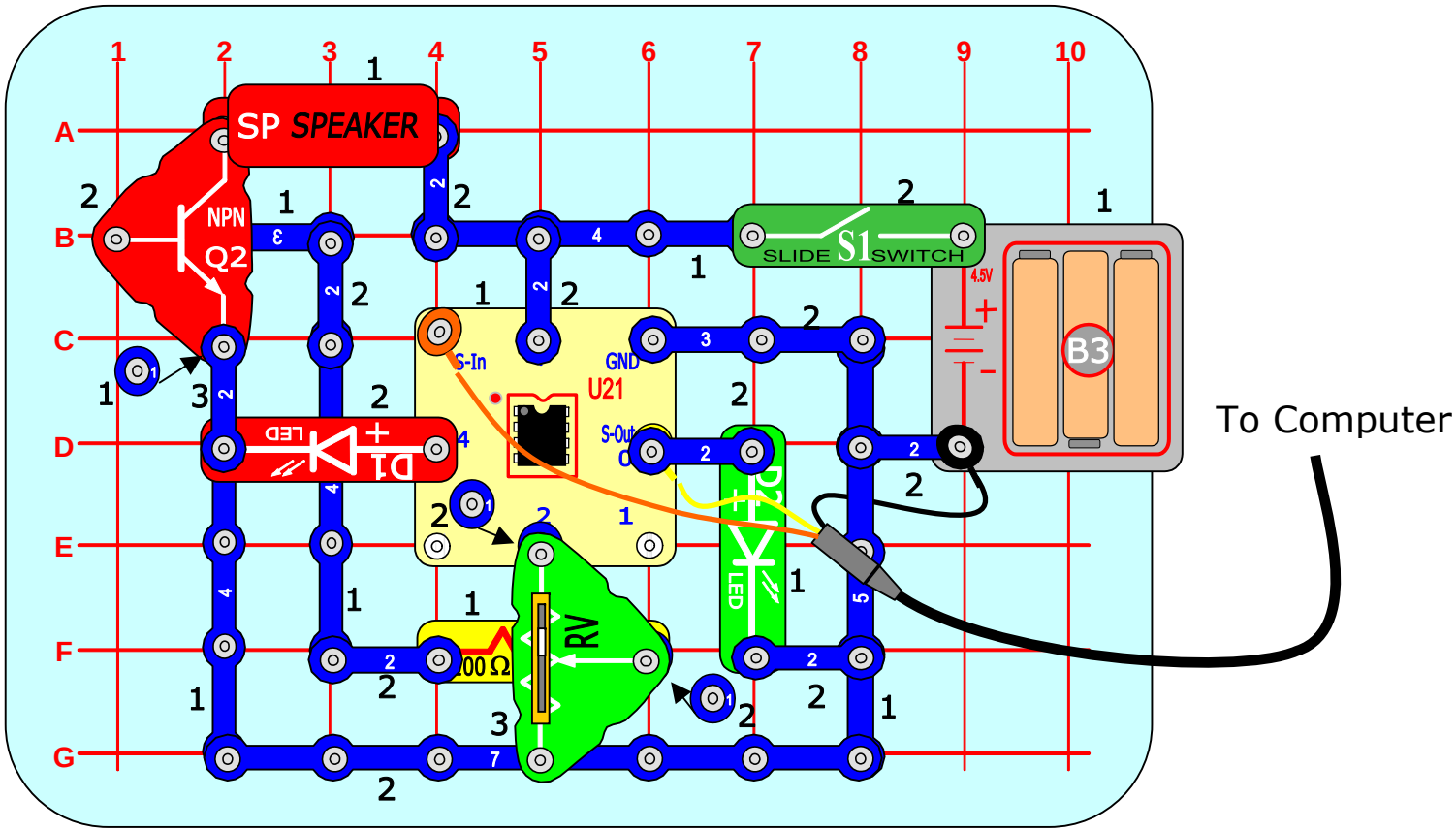


Turn switch S1 to ON. Click on the ‘Program’ button or press F5 to download the new program. The new program should play the Happy Birthday song and both LED’s should be off. In the program editor change the play command to ‘play 3, 3’ and download again. When the new song starts playing remove the yellow snap and rotate the ‘2 Snap’ to add the green LED. Rudolph the Red Nosed Reindeer should be playing and the LEDs should flash in tempo with the music. Take some time to experiment with different combinations of the play.

PROJECT 2: ADDING AMPLIFIER & LOUDNESS CONTROL

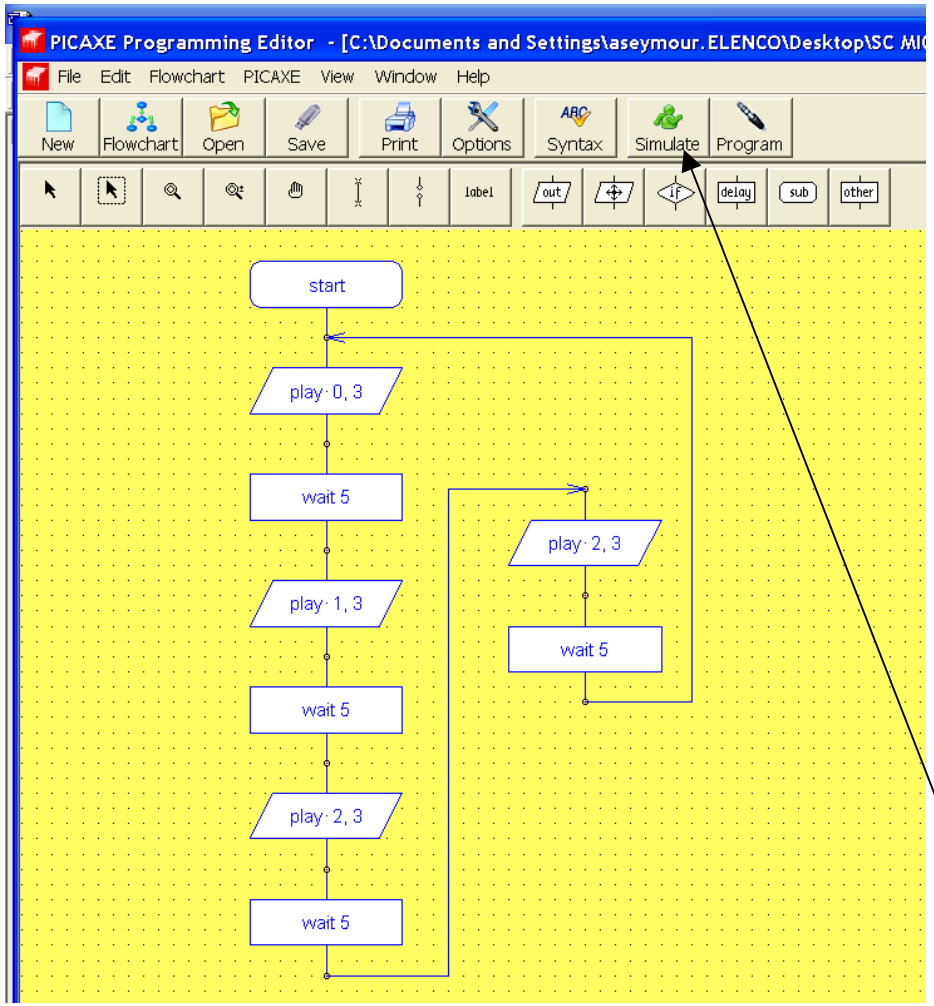
Modify the Snap Circuit® from the previous project to look like the circuit below. The variable resistor RV will be used to increase and decrease the audio level. The NPN transistor ‘Q2’ is used to amplify the power to the speaker ‘SP’.

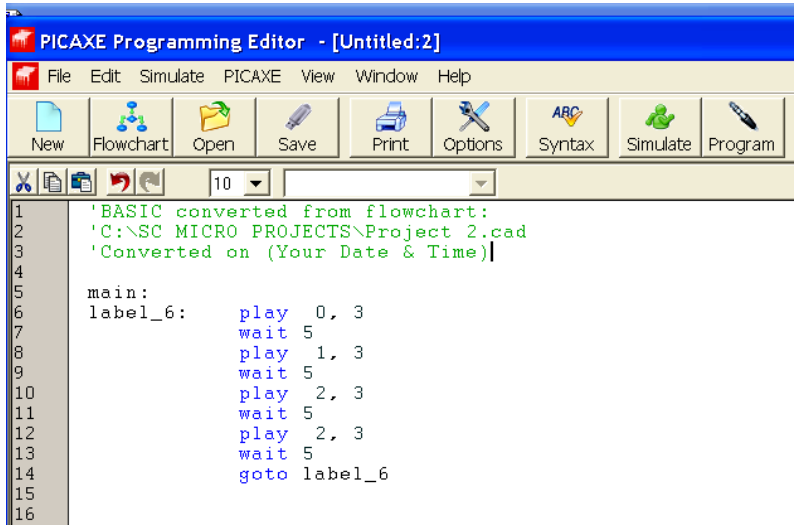
Assuming the micro-controller is still programmed to play and flash the lights, switch ‘S1’ to “ON” and test the circuit.



Use the loudness control to adjust audio level for desired loudness.

The advantage of a micro-controller is that the circuit need not change to produce different audio effects. By modification of the program, all four songs can play, one after the other. Consider the flowchart program shown below.





In the flow chart shown on the left, each song is played one following the other with a 5 second delay between each of the songs. Open the program editor and draw this flowchart. Use the simulate button to test that the program plays the songs and pauses for 5 seconds between each song. If the Flowchart passes the simulate test save it in your project folder.

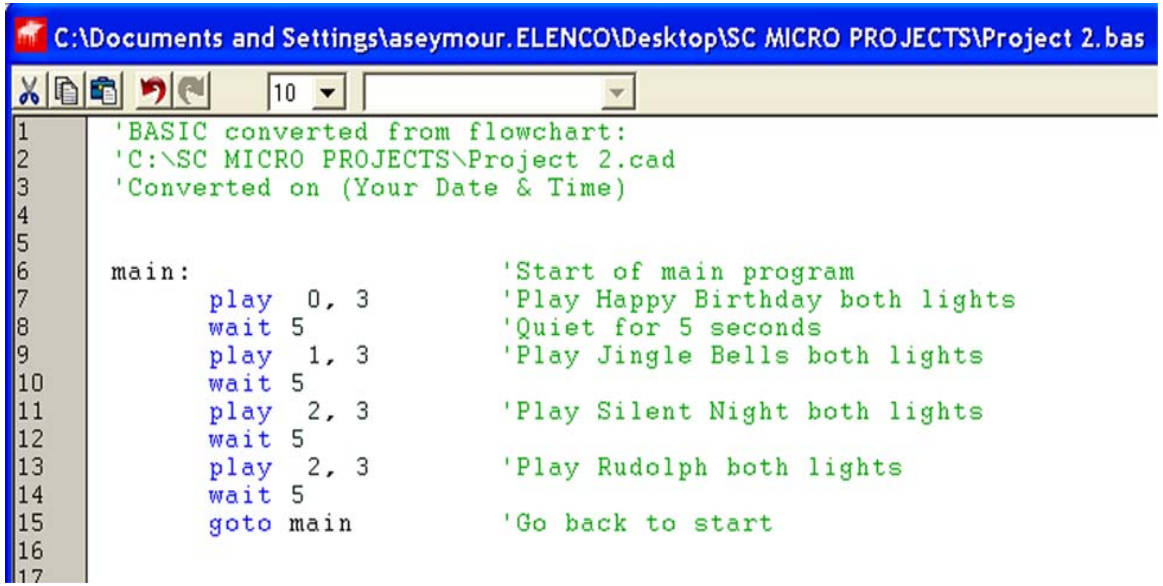
Under the PICAXE® heading click on the 'Convert Flowchart to Basic….' command. The screen shown above and on the right should appear.

Project 3: The Value of Comments and Checking Program Length

Sometimes it can be hard to remember the purpose for each step of the program. **Comments** (an explanation after the apostrophe (') symbol) can make each line of a program much easier to understand and remember. These comments are ignored by the computer when it downloads a program to the micro-controller.

A label (e.g. **main:** in the program above) can be any word (apart from keywords such as 'switch'), but the label must begin with a letter. A label must also end with a colon (:). The colon 'tells' the program editor that the word is a label.

The previous program uses the **wait** command. The commands **wait** and **pause** both create time delays. However **wait** is used with whole seconds, **pause** can be used for shorter time delays (measured in milliseconds or 1000th of a second).
Wait must be followed by a number between 1 and 65.
Pause must be followed by a number between 1 and 65535.

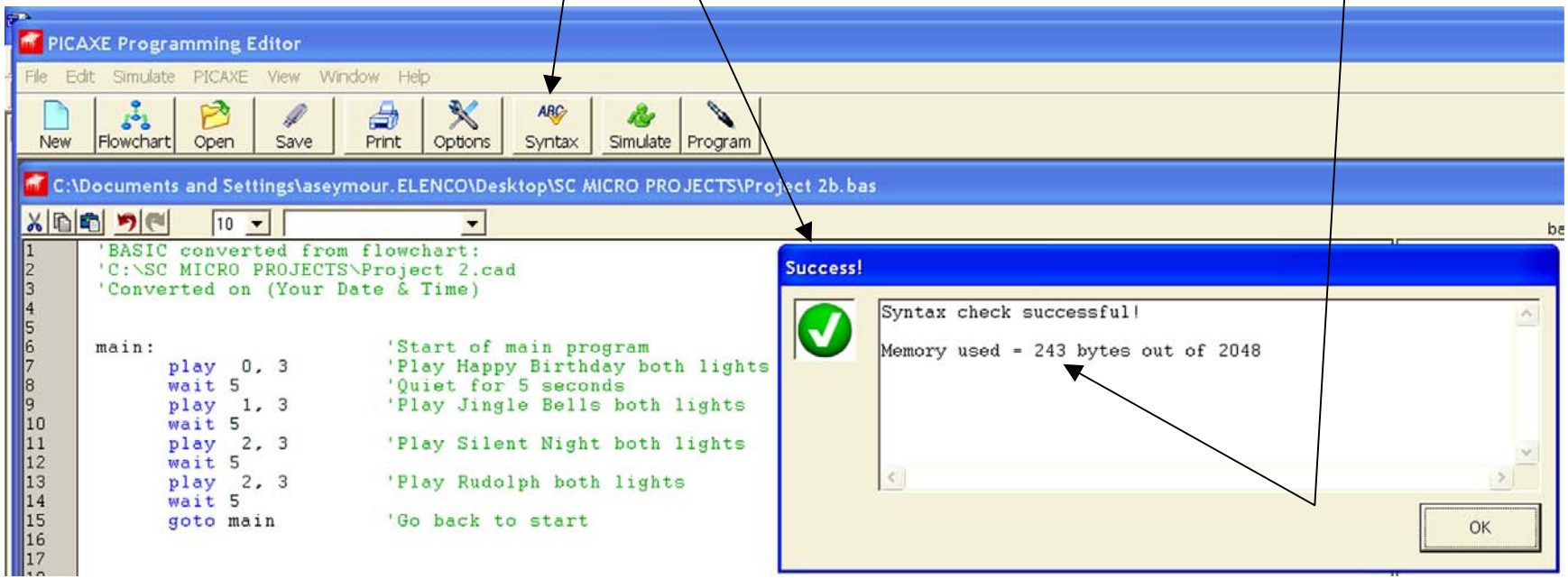


It is also a good programming technique to use tabs (or spaces) at the start of lines without labels so that all the commands are neatly aligned. The term **'white-space'** is used by programmers to define tabs, spaces and blank lines, and the correct use of white-space can make the program listing much easier to read and understand. Note these changes were made to the previous program and are shown here. Save program after these changes and exit the flowchart conversion screen. On main editor screen reopen program.

Checking Program Length.

As programs become complex it is possible the micro-controller will run out of memory to store the program. Check the previous program length by clicking the 'Syntax Check' button.

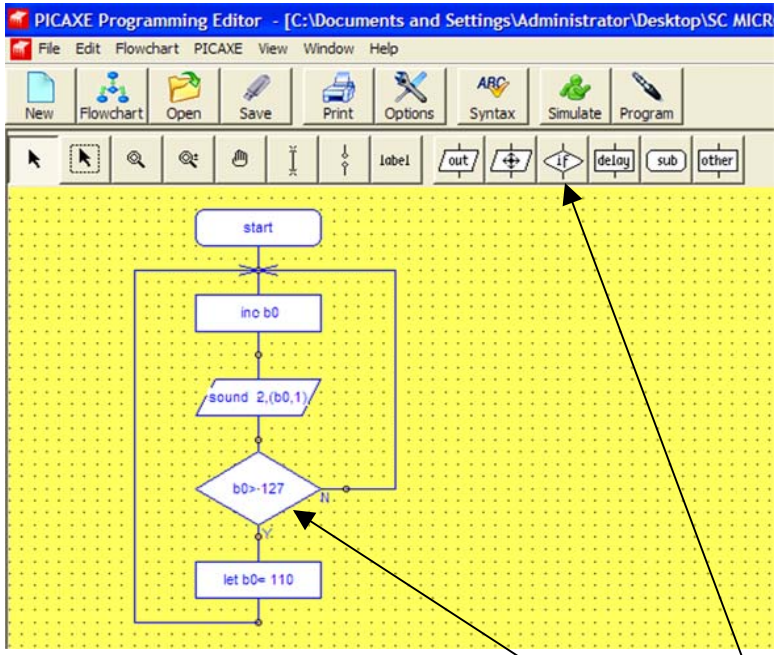
If there are no programming errors the 'Success!' window will open and show the amount of memory used by the program.



To play all 4 songs and flash both lights, this program used 243 bytes out of 2048 or 11.9% of the total memory. Remove the '2 Snap' on U8 (S-Out or Pin 0) and connect the red programming wire. Download this program. Then reconnect the '2 Snap' and run the program.

Project 4: Other Sounds

The ‘sound’ command



Syntax:

SOUND pin,(note,duration,note,duration...)

- Pin is a variable/constant (0-4), which specifies the i/o pin to use.
- Note(s) are variables/constants (0-255) which specify type and frequency.

Note 0 is silent for the duration.
Notes 1-127 are ascending tones.
Notes 128-255 are ascending white noises.

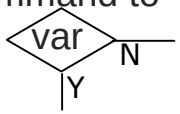
- Duration(s) are variables/constants (0-255) which specify duration of the note (multiples of approx 10ms).

Function:

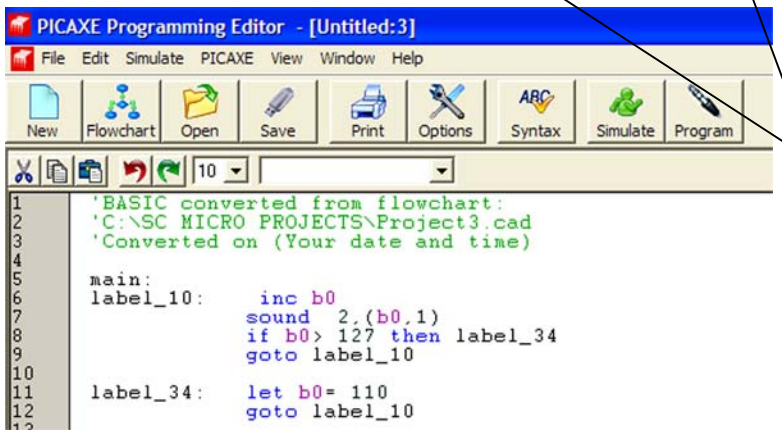
Play sound ‘beep’ (1-127) or noises (128-255).

Information:

Frequency and duration must be used in ‘pairs’ within the command. Draw the flow chart shown here.

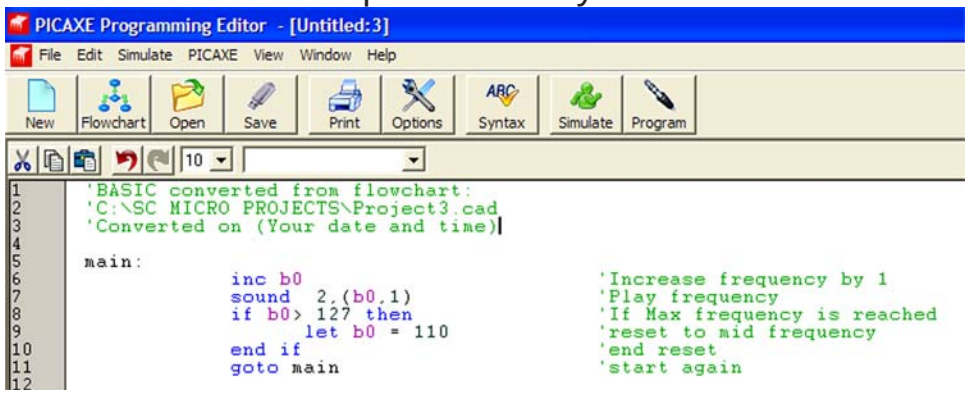
Use the ‘other’ button and the ‘…’ box to create the ‘inc b0’ box and the “let b0= 110” box. The ‘Sound’ command is under the ‘out’ menu. Be sure to edit the sound command to “sound 2,(b0,1)”. Use the IF button and the 

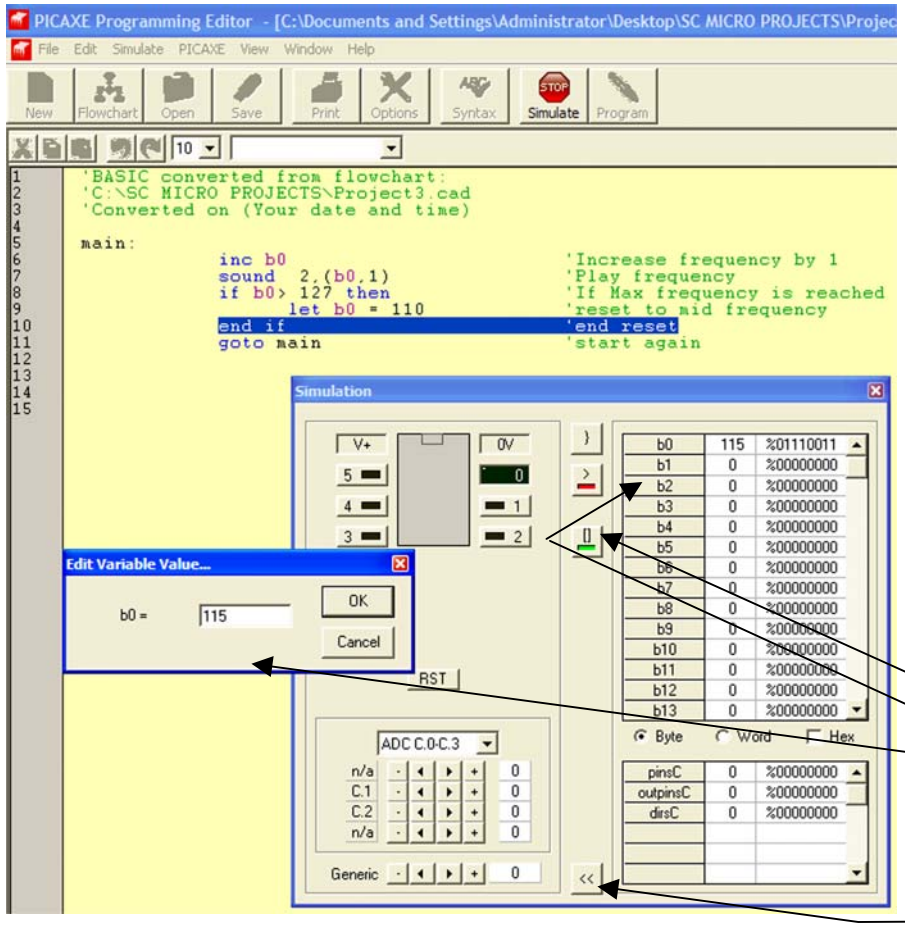
button to create the “b0>127” label.



Save the flowchart for your reference and convert it to a basic program. The converted program should be similar to the one shown on the left. This program is 21 bytes long and looks a little confusing. Try editing the program to appear as shown on the next page.

This program is 2 bytes less and much easier to read with less jumping and only one label. Of course the comments will help later when you edit this for different applications.





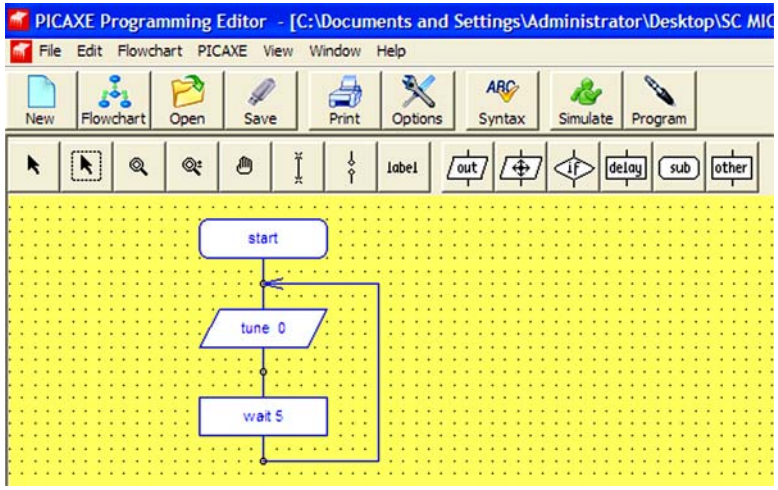
It is a good practice to build up a library of small sub routines and then use them in different applications when they are needed. The more information stored in the library the easier it will be to import and use these routines later.

If you run a simulation of this program it will take a long time for the b0 variable to reach 127. Open the variable panel;
Pause the program, and double click on the b0 to change it to 115.

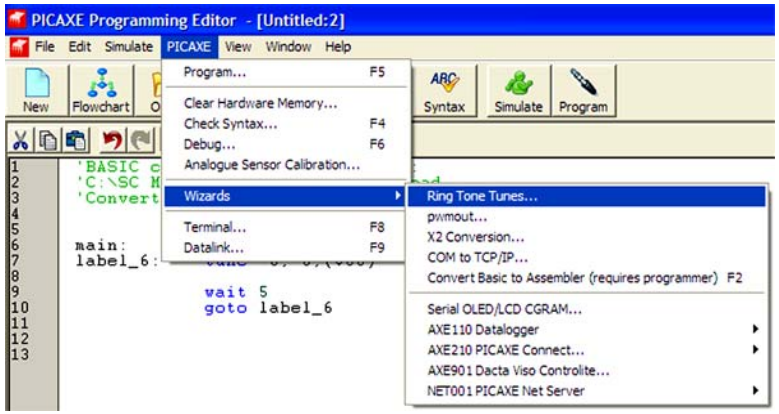
Restart the program and watch the b0 variable loop between 110 and 127. In simulation the sound command does not play the frequency. Instead a 'Beep' is produced to let you know a sound command was executed. After down loading the real sounds will be heard.

Save program to your library then download it to play special effect sound. Close file.

1.Pause & Start Button
2.Double Click b0
3.Edit to 115

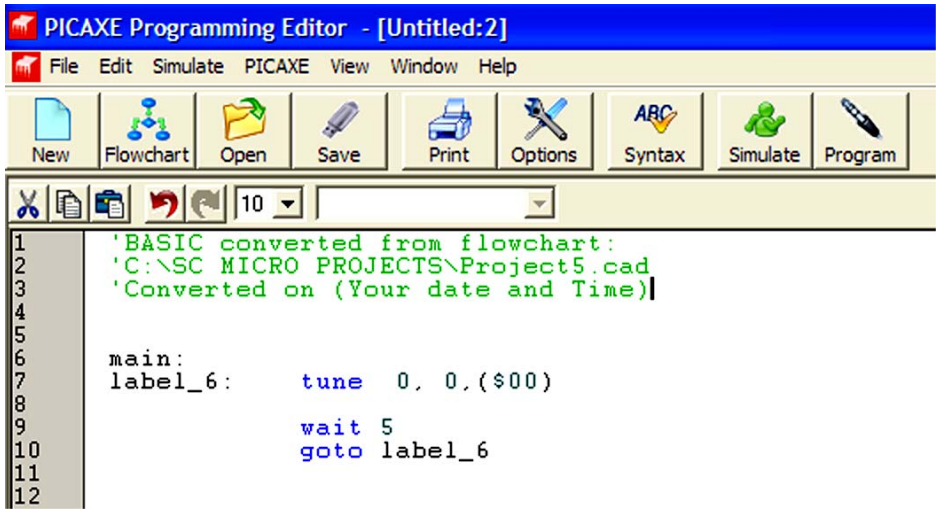


Click on the PICAXE menu again and pick Wizards and Ring Tone Tunes as shown below.

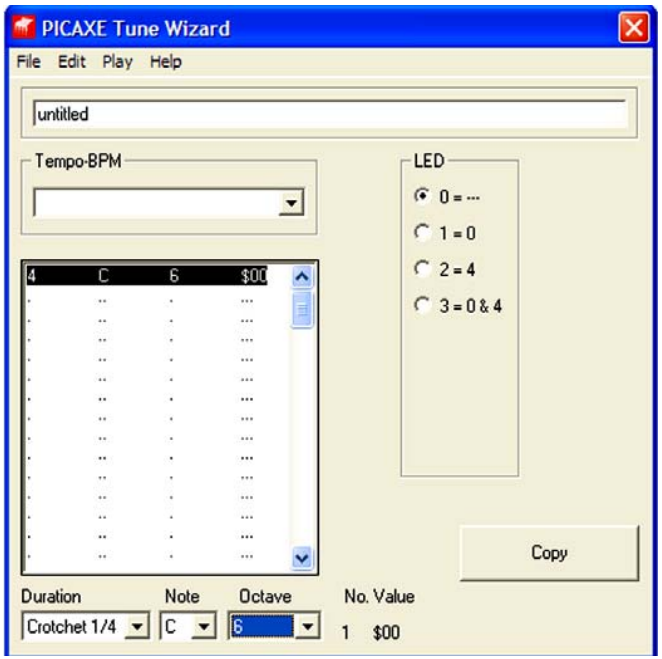


Project 5: The Tune Wizard

First create the simple flowchart shown here. In many cases the flowchart will serve to indicate a process but the basic program will be greatly modified and not a direct conversion of the flowchart.
Next click on the PICAXE menu and select “Convert Flowchart to Basic…”. This will open the window shown below. Place the curser at the beginning of line 8.



This will open the wizard as shown on the next page.

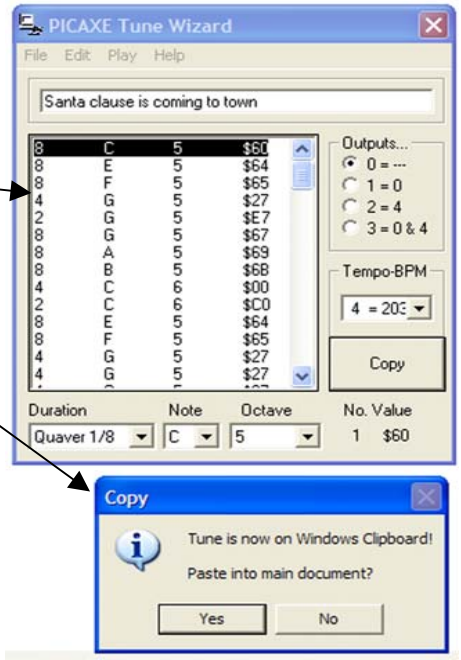


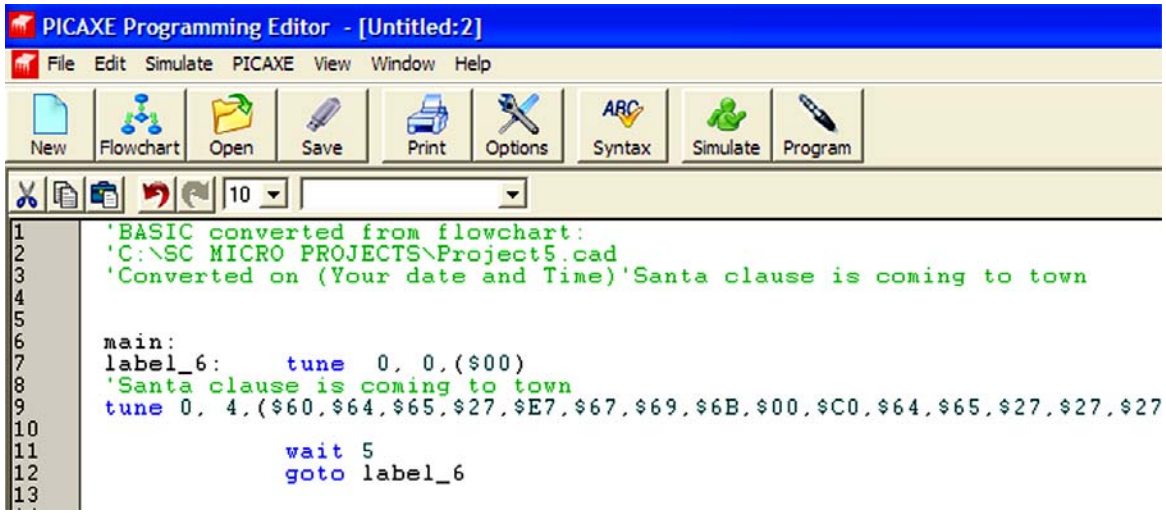
The Tune Wizard allows musical tunes to be created for the micro-controller. Tunes can be entered manually or imported from another source. These tunes or ring tones are also available on the Internet, and used on some cell phones. The tunes created by this micro-controller can only play one single note at a time (monophonic). The micro-controller cannot play multiple notes (polyphonic) ring tones found also on the Internet.

Start by opening a tune that already is on your computer. Click on the File menu and select 'Open'.
Find the Program Editor directory and open the Christmas_1 folder as shown here.

Open this folder (you may have to unzip – see page 34 for instructions on extracting) and find the tune "Santa clause is coming to town" and select it. The wizard should load all the notes as shown here.
Click the Copy button and the Copy window should open.

Click the "Yes" button in the Copy Window to put tune into the program as shown on next page.

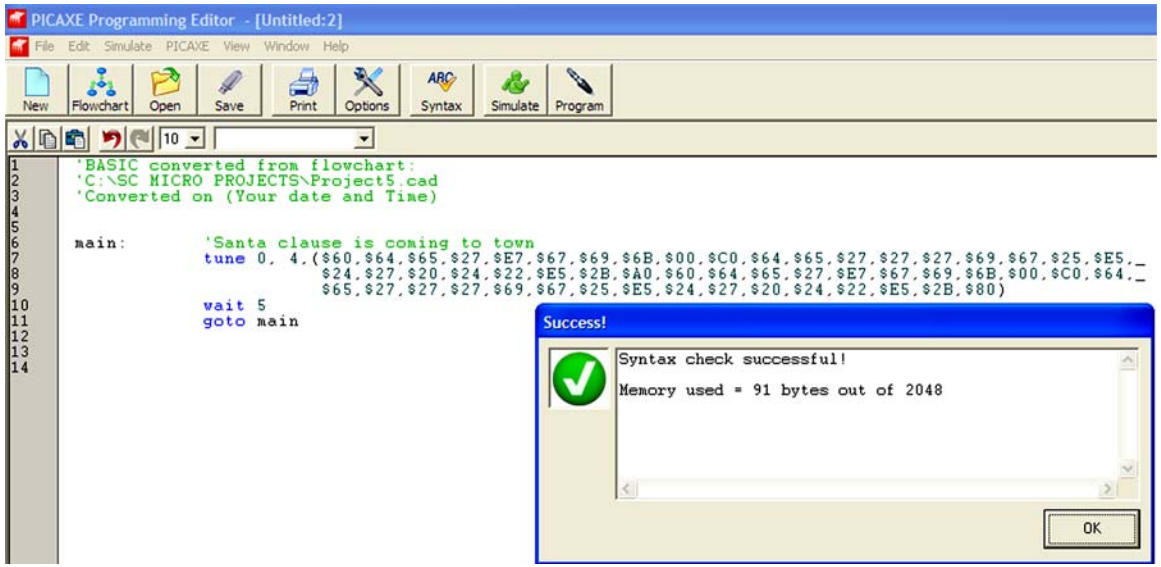




Santa clause is coming to town was placed at the curser point before opening the tune wizard.

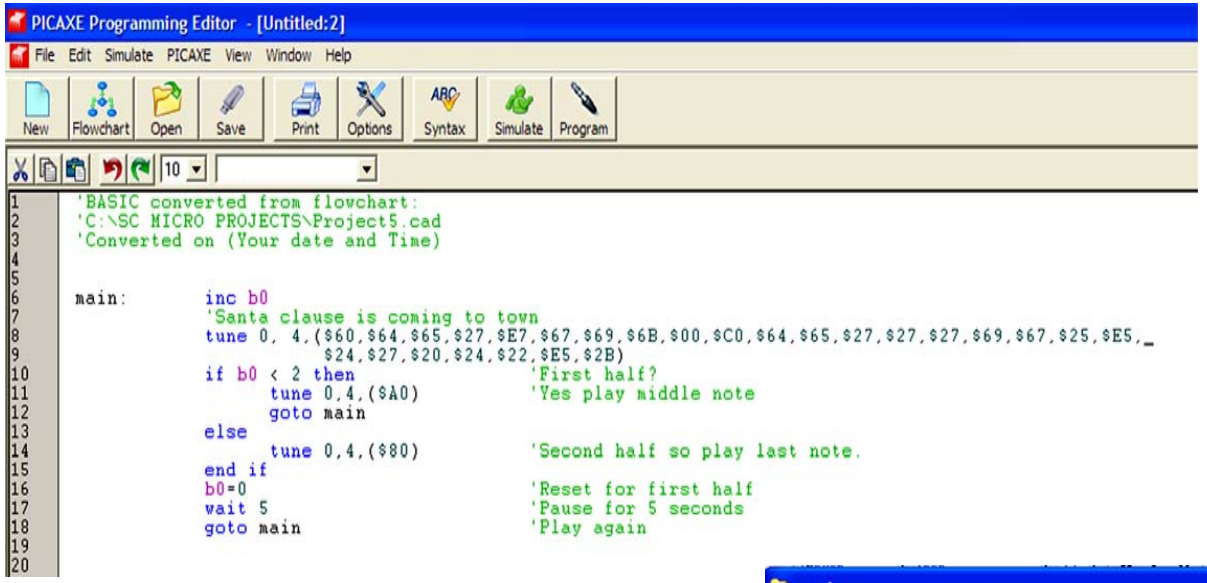
Notice the tune data goes off the visable screen and the program has an extra tune command that is no longer needed.

Clean up the program to look like the one shown below using the hints shown.

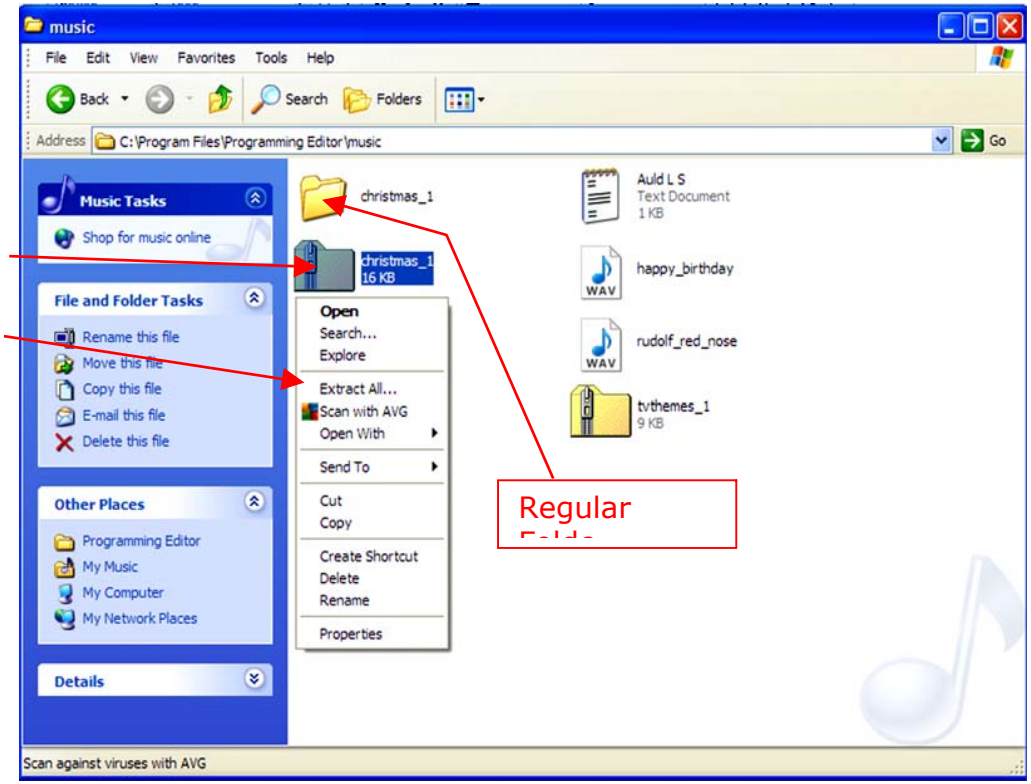


The above tune that was on one line has been converted to three lines by adding the underscore character at the end of each line. The extra tune command was eliminated with the labels that were not needed. Runnig syntax check shows the tune uses 91 bytes to program! OUCH! That is a great deal of memory for just one tune. Play the tune by clicking the simulate button.

Notice how the first 26 notes are identical to notes 28 to 53. Except for the last note and note 27 the tune is the same. Redundancy in programming is a waste of memory. Change the tune section of your program by typing in or deleting the information on your screen to match the following;



A syntax check will show this section is 71 bytes, or a savings of 20 bytes of memory. The tune will play correctly after down load but may have a few pauses when played by the simulator.
The easiest way to place a tune into your program is to import them from the music folder supplied in the Program Editor folder.

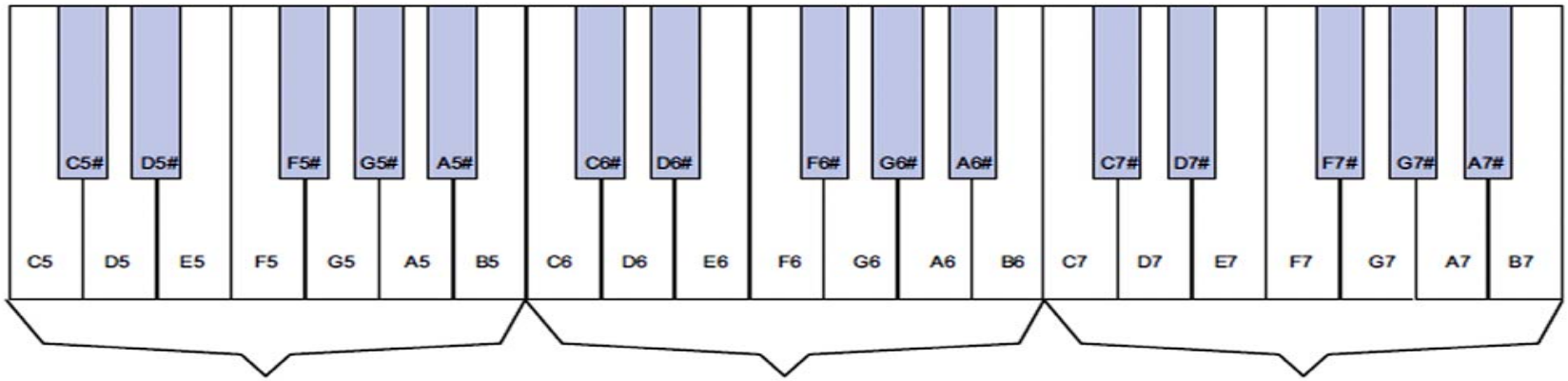


Some folders may have to be unzipped before they can be used. Use the music folder in the Program Editor folder as shown on the right. The tune can then be selected from the regular Christmas_1 or tvthemes_1 folders. First select the zip folder and use the right mouse button to get the drop down menu shown here. Next click on the extract all with left mouse button to make the regular folder that can be used by the wizard. Tunes can now be loaded and tested on the computer by clicking the ‘Play’ menu. The tune played may differ slightly due to the different ways that the simulator generates the notes. Once your tune is complete, click the ’Copy’ button to copy the tune command to the Windows clipboard. The tune can then be pasted into your main program at the curser position. Of course tunes can also be written into the wizard one note at a time.

Tune Wizard menu items and shortcut keys:

File		
New	Ctrl + N	Start a new tune
Open	Ctrl + O	Open a previously saved tune
Save As	Ctrl + S	Save the current tune
Import Ring tone		Open a ring tone from a text file
Export Ring tone		Save tune as a ring tone text file
Export Wave		Save tune as a Windows .wav sound file
Close	Ctrl + X	Close the Wizard
Edit		
Insert Line	Shift + Ins	Insert a line in the tune
Delete Line	Shift + Del	Delete the current line
Copy BASIC	Ctrl + C	Copy the tune command to Windows clipboard
Copy Ring tone		Copy tune as a ring tone to Windows clipboard
Paste BASIC		Paste tune command into Wizard
Paste Ring Tone	Ctrl + V	Paste ring tone into Wizard
Play		Play the current tune on the computer's speaker
Help		Open a help file.

To create a song note by note use the chart and guide shown below.



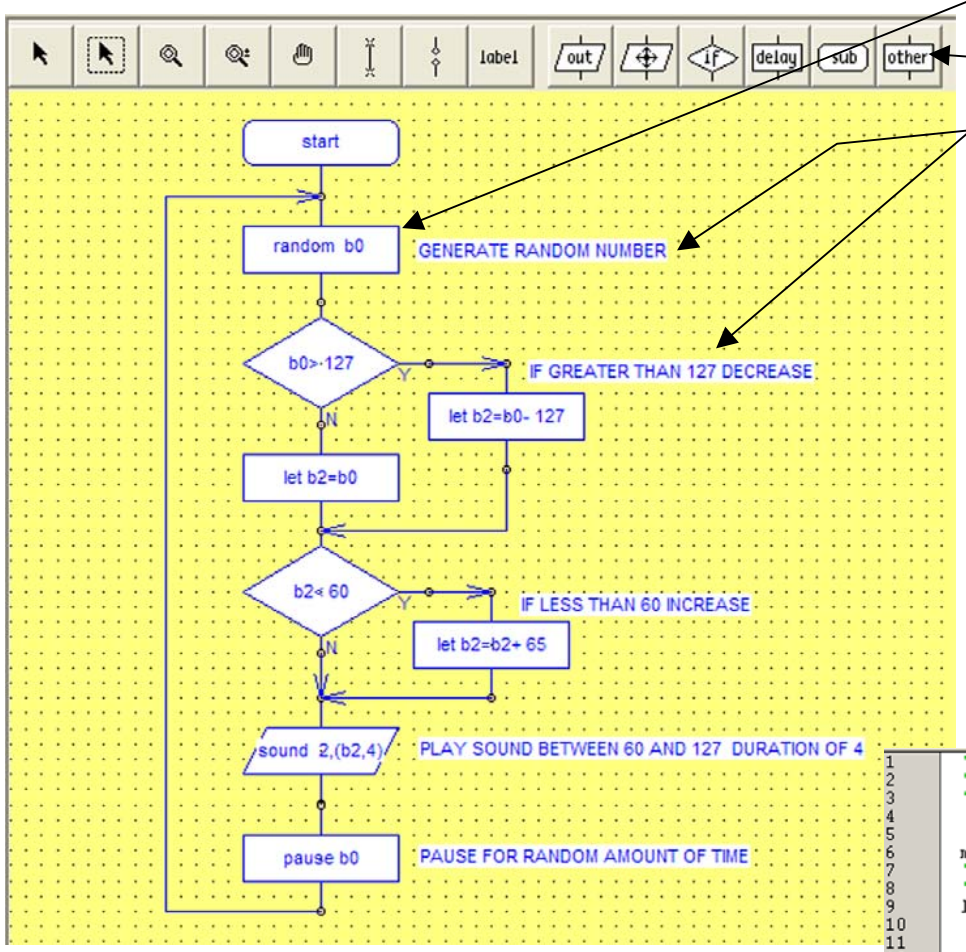
C5 = 262 Hz
C5# = 277 Hz
D5 = 294 Hz
D5# = 311 Hz
E5 = 330 Hz
F5 = 349 Hz
F5# = 370 Hz
G5 = 392 Hz
G5# = 415 Hz
A5 = 440 Hz
A5# = 466 Hz
B5 = 494 Hz

C6 = 523 Hz ("Middle C")
C6# = 554 Hz
D6 = 587 Hz
D6# = 622 Hz
E6 = 659 Hz
F6 = 698 Hz
F6# = 740 Hz
G6 = 784 Hz
G6# = 831 Hz
A6 = 880 Hz
A6# = 932 Hz
B6 = 988 Hz

C7 = 1047 Hz
C7# = 1109 Hz
D7 = 1175 Hz
D7# = 1245 Hz
E7 = 1318 Hz
F7 = 1396 Hz
F7# = 1480 Hz
G7 = 1568 Hz
G7# = 1661 Hz
A7 = 1760 Hz
A7# = 1865 Hz
B7 = 1975 Hz

Project 6: Robotic Sounds

This project explains the use of the random command and labels. Build the following flowchart.

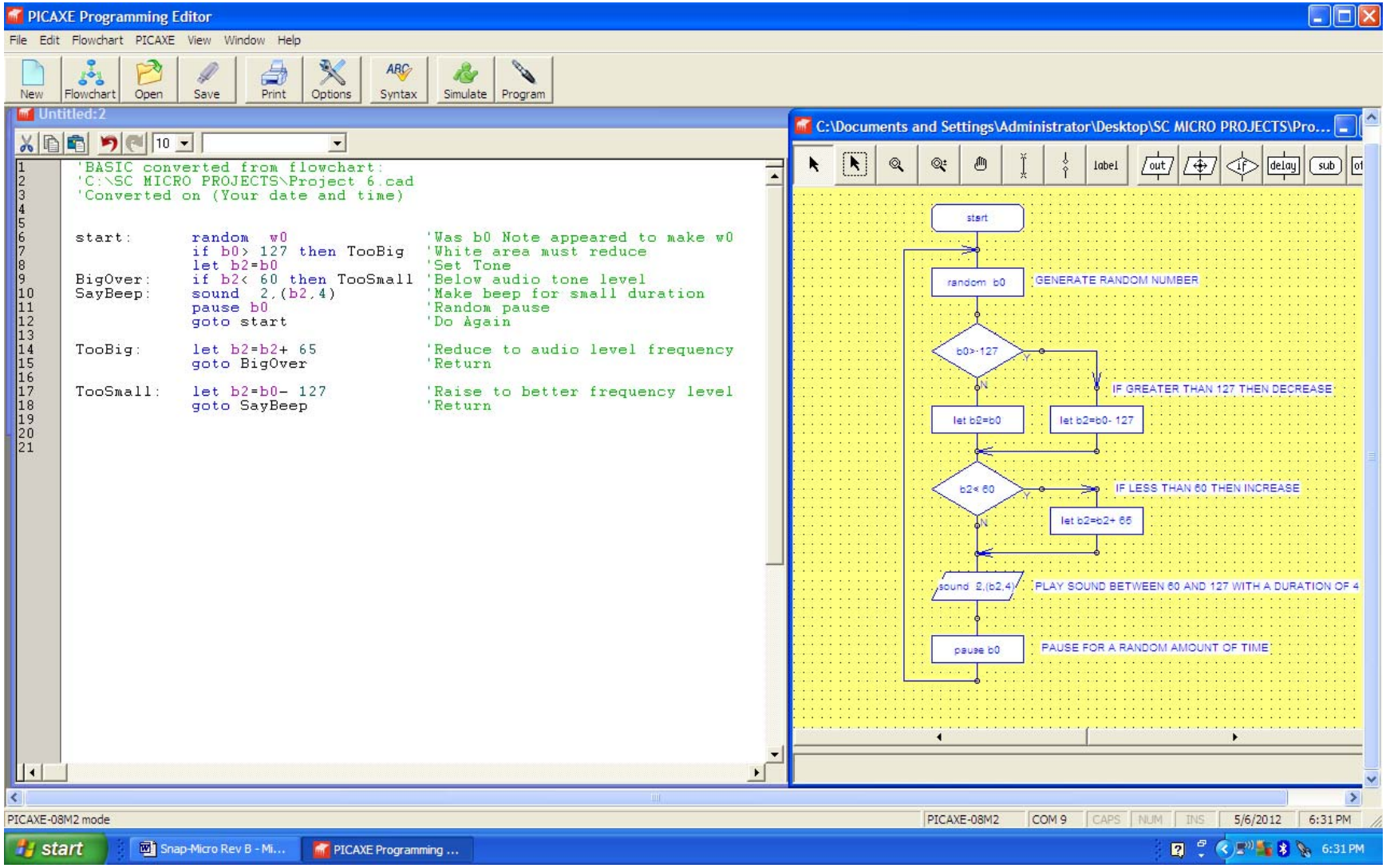


The random box can be found under the 'other' menu.

Use the 'label' button to add notes to help explain each section of the flowchart. A good practice is to create a word document with both flowchart and basic program side by side. These labels help tie the two charts together.

When you test your flowchart with simulate, blocks should turn red as the program loops. A beep should be heard when the sound block turns red. If no errors occur, save the flowchart to your library. Use the PICAXE® drop down menu to convert this chart to a basic program. The result should be similar to the one shown here.

```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\PROJECT6.CAD
'Converted on Date at Time


main:
'Note: it is recommended to change byte variable here to word variable!
'(see manual for details about 'random' command operation)
label_6:    random  b0
            if b0> 127 then label_18
            let b2=b0
label_27:   if b2< 60 then label_33
label_3A:   sound  2,(b2,4)
            pause b0
            goto label_6

label_18:   let b2=b0- 127
            goto label_27

label_33:   let b2=b2+ 65
            goto label_3A
```



The picture above shows the same basic program after changes and editing to make it more readable and the flowchart. This picture was created by overlapping the windows in the program editor and using the print screen <prt sc> key to place entire screen on clip board. It was then pasted into this word document. A document similar to the picture above can be created and stored in your library for future use.

Project 7: SWITCHES AND DIGITAL INPUTS

Digital Inputs

A digital input can only be ‘on’ or ‘off’. Some examples of a digital inputs found in Snap Circuits® are:

• Push Button Switch 

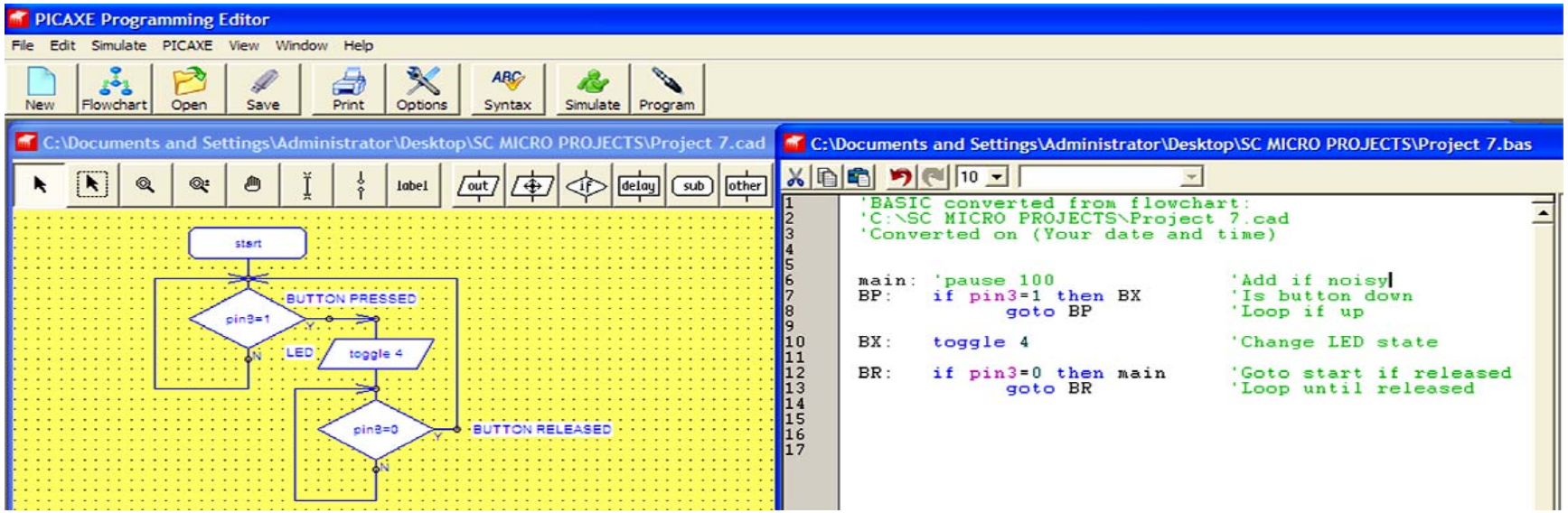
Always ‘OFF’ when released

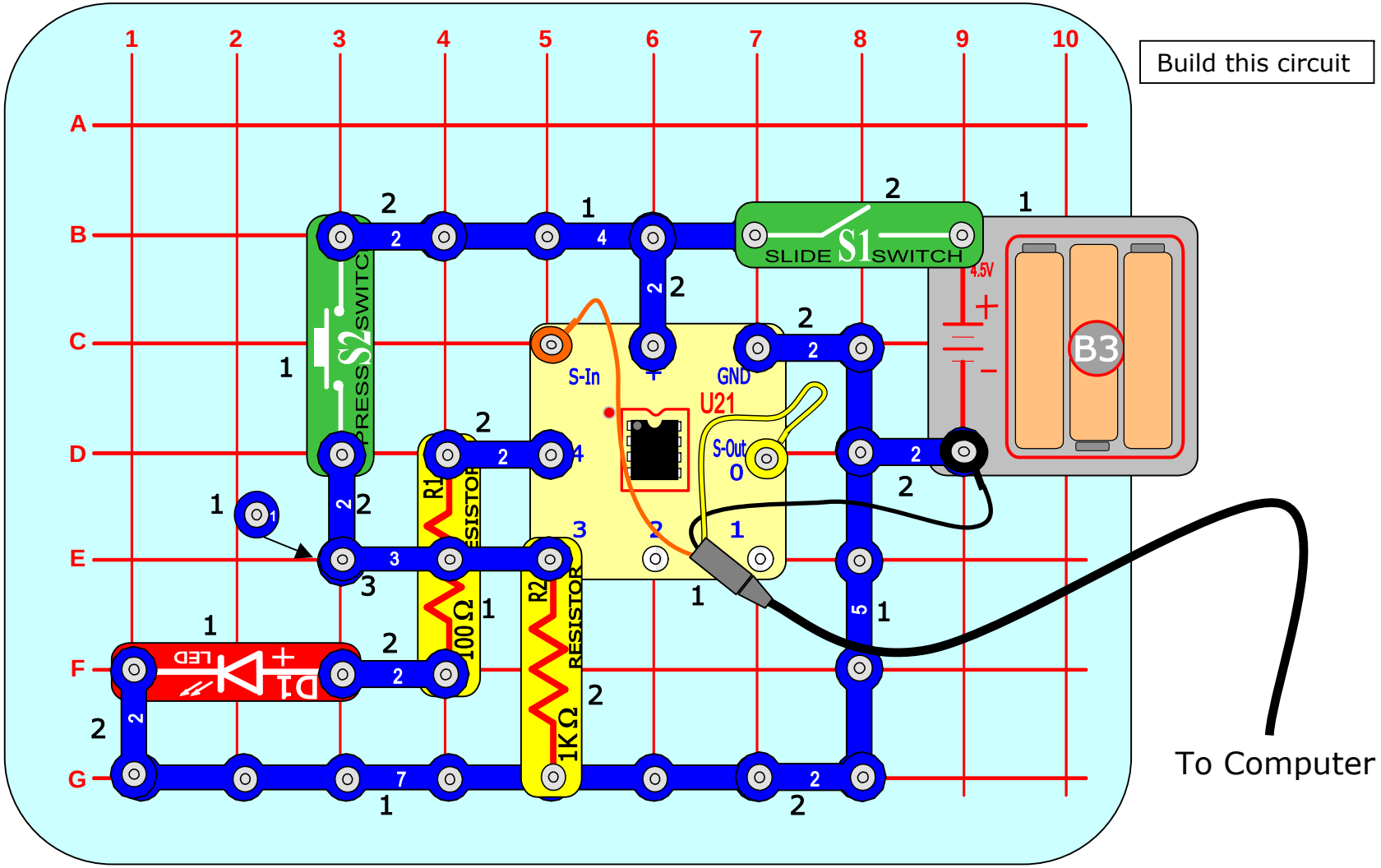
• Slide Switch 

Stays ‘ON or ‘OFF’ after switching.

Most switches use a metal contact that snaps into place. This action may cause the switch to bounce and produce “switch noise” when it is closed. The program below should change the state of the LED each time the Push Button switch is pressed. In this program output pin 4 toggles (changes state) every time the push switch on input pin 3 is pressed.

Using the Program Editor construct the flowchart shown here. Then use the PICAXE® menu to convert it to a program similar to the program on the right. When the program on the right was edited for clarity, the ‘pause 100’ was added as a note. This should be made a command by removing the first apostrophe in front of the “pause” if the switch is very noisy when pressed or released. Save chart and program.

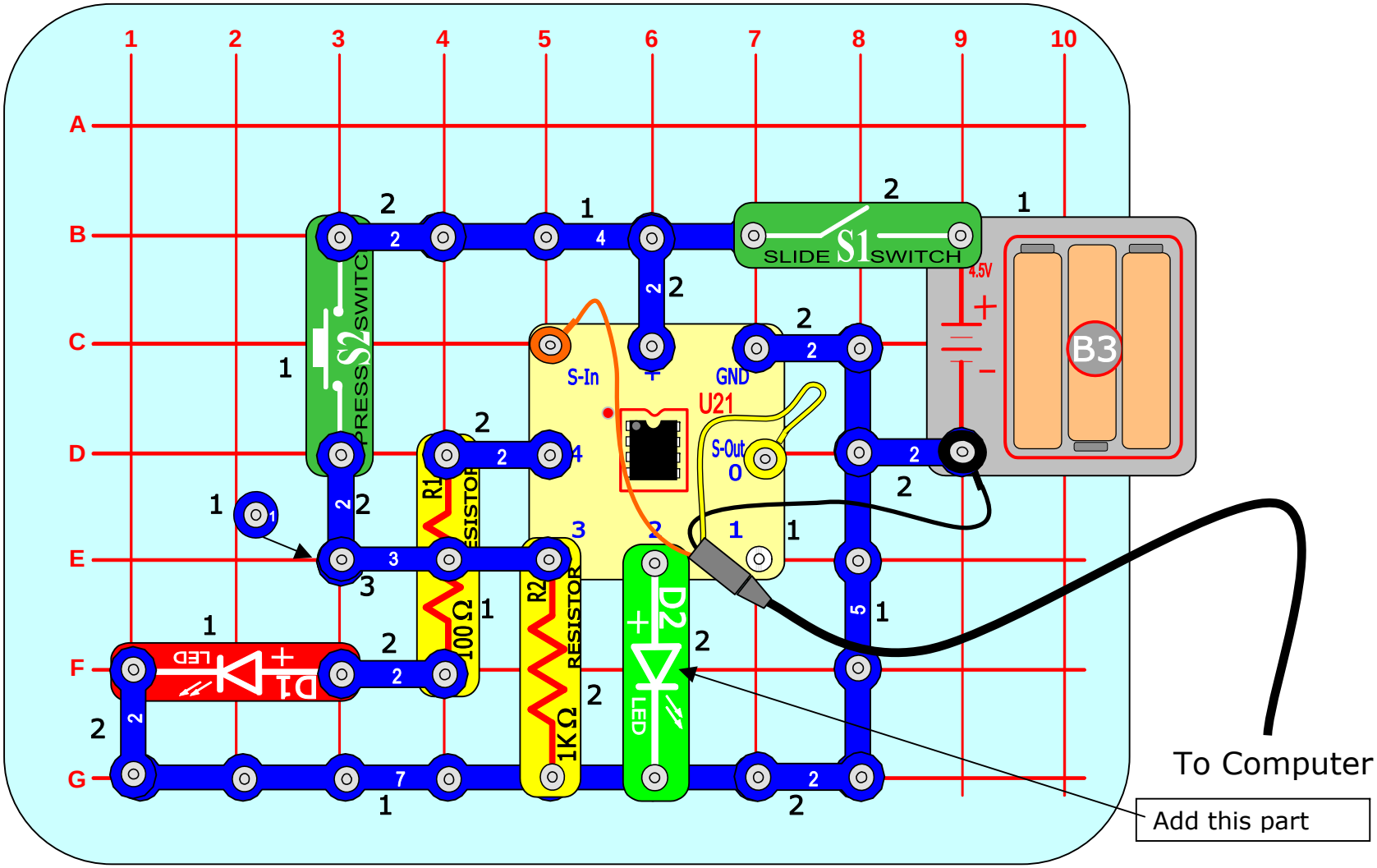




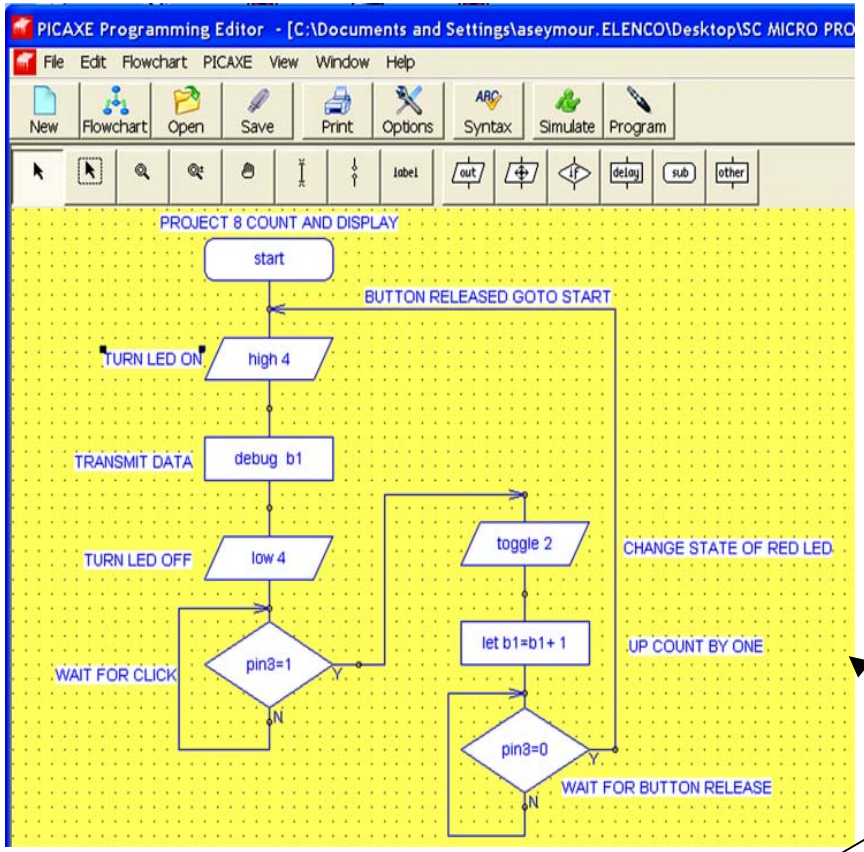
Load the program into the micro-controller. As shown in the flowchart, the first two lines make up a continuous loop. If the input is off (=0) the program just loops around between program lines numbered 7 and 8 on the left side of the program. If the switch is on (=1) the program jumps to the label called ‘BX’. The state of output pin 4 is toggled and then the program drops into a second continuous loop that waits for the button to be released. After the button is released the program jumps back to the beginning and waits for the next push. Take note that only a label command follows the “then” in the “if” statement. No other words apart from a label are allowed in this position. Keep circuit for next project.

PROJECT 8: COUNTING AND DISPLAYING EVENTS

Modify the previous circuit to look like the one shown below.



Create the flowchart and program shown on the next page.



debug

Syntax:

DEBUG {var}

- Var is an optional variable value (e.g. b1).

Function:

Display variable information on computer in Debug Window

The debug command uploads the current variable values for all the variables via the download cable to the computer screen. This enables the computer screen to display all the variable values in the micro-controller for debugging purposes.

Note that debug acts on 'B.0 / output 0'. Therefore programs that use output 0 may corrupt the data condition. In this case it is recommended to use the following structure before a debug command.

```
low B.0        ' reset B.0 to correct condition
pause 500      ' wait a while
debug          ' display values on computer screen
```

Note that the debug command uploads a large amount of data and so significantly slows down any program loop.

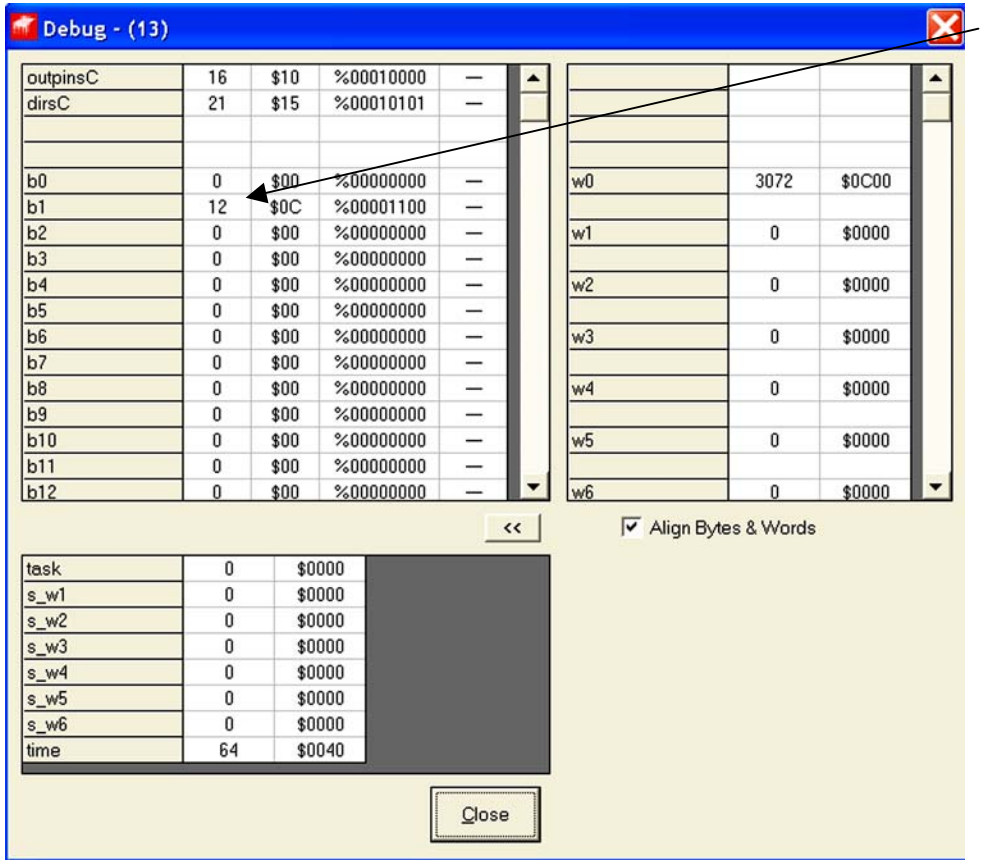
1. Make the flowchart shown on left.
2. Use PICAXE drop down menu to convert to basic
3. Edit program for clarity and add safety reset.
4. Go to next page for notes on running the program.

C:\Documents and Settings\aseymour.ELENCO\Desktop\SC MICRO PROJECTS\Project 8.bas

```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project 8.cad
'Converted on (Your date and time)
'Edited

main:       high 4                  'Turn on green LED
            low 0                   'Reset pin 0
            pause 500               'Let things settle
            debug  b1               'Display data on screen
            low 4                   'Turn off green LED
Hloop:      if pin3=1 then Hpress   'Check for button press
            goto Hloop              'Loop if no press

Hpress:     toggle 2                'Change red LED state
            let b1=b1+ 1            'Increment b1 variable
Lrelease:   if pin3=0 then main     'If button released goto start
            goto Lrelease           'Otherwise loop until released
```

Untitled:2

```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project 8.cad
'Converted on (Your date and time)

main:
label_6:    high 4
            debug  b1
            low 4
label_1B:   if pin3=1 then label_26
            goto label_1B

label_26:   toggle 2
            let b1=b1+ 1
label_34:   if pin3=0 then label_6
            goto label_34
```



Switch pressed 12 times

If the switch is pressed and released quickly while the red LED is still on, the event will be missed. If it is only pressed and not released, then the event will be captured.

Notice that the b1 variable value is shown in decimal (12), hexadecimal ($0C), and binary (%00001100). For now the decimal output will be all we need. Also the b1 variable is the high byte of W0, therefore W0 will increase by 256 every time b1 increases. For more information on this use the help file provided in the program editor.

Since the green LED is only ON during odd values of b1, the light should have been lit during values 1,3,5,7,9, and 11. The light has been turned on 6 times, off 6 times, and is presently off. Although we are using a light, the event could have been a switch to open a door, turn on a motor, start an oven, or any other event that would need monitoring. The Debug window is not very user friendly but there is another command ‘sertxd’ that can improve monitoring.

The same circuit will be used in the next project.

PROJECT 9 - Using Serial Terminal with Sertxd

The sertxd command sends a user defined serial string to the computer (at baud rate 4800). This can be displayed by the included Serial Terminal function under the PICAXE®>Terminal drop down menu. The Serial Terminal can also be automatically opened every time a download takes place by checking the "Open After Download" → 

box in the View>Options>Editor drop down menu.

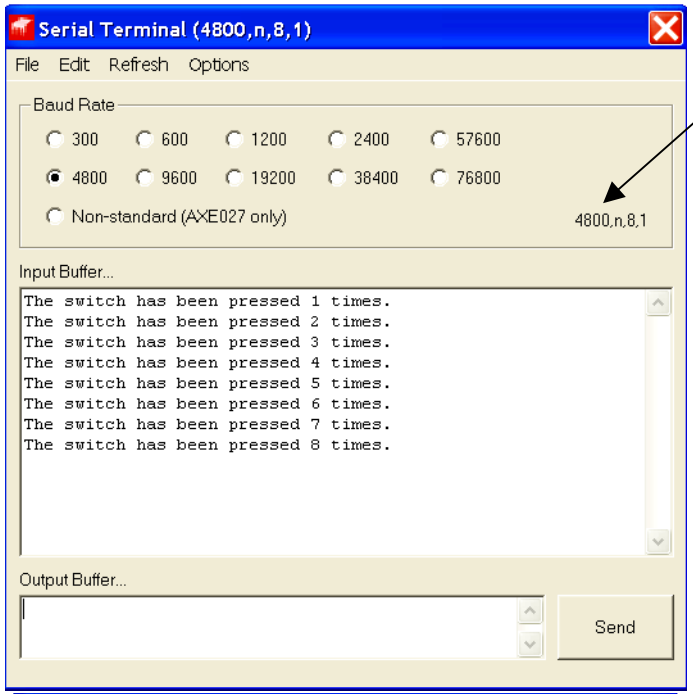
No flow chart is needed in this project since we are only changing the 'debug' command to the 'sertxd' command. Open project 8 basic program in the editor and change it to the following.

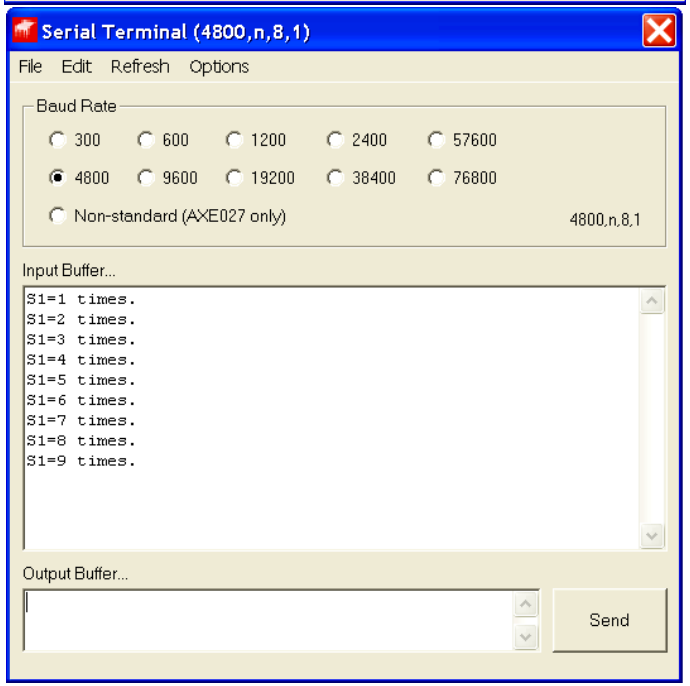
```
'BASIC converted from project 8 by hand:
'C:\SC MICRO PROJECTS\PROJECT9.BAS
'Converted on DATE at TIME

main:       high 4                  'dislay message on terminal screen
            sertxd("The switch has been pressed ",#b1," times.",13,10)
            low 4

Hloop:      if pin3=1 then Hpress   'BUTTON PRESSED
            goto Hloop              'Loop if not pressed

Hpress:     toggle 2
            let b1 = b1 + 1         'increment b1


Lrelease:   if pin3=0 then main     'if BUTTON RELEASED goto start
            goto Lrelease           'otherwise loop until released
```

All text must be placed in quotation marks and should appear in red (default color, but may be changed in program editor options.)
10 = line feed (LF)
13 = carriage return (CR)

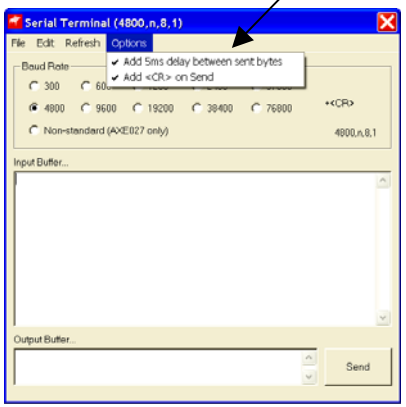
Variables need the # sign in front of them and will be displayed in purple.

To open the Terminal window use the PICAXE® drop down terminal or press F8 function key. The terminal window shown on the next page will open.





Each time used, go to Options and set up as shown here;



Make sure the Baud Rate is set to 4800.

Each time the S2 pushbutton is pressed the micro-controller records the event and then transmits the message shown on the Serial Terminal.

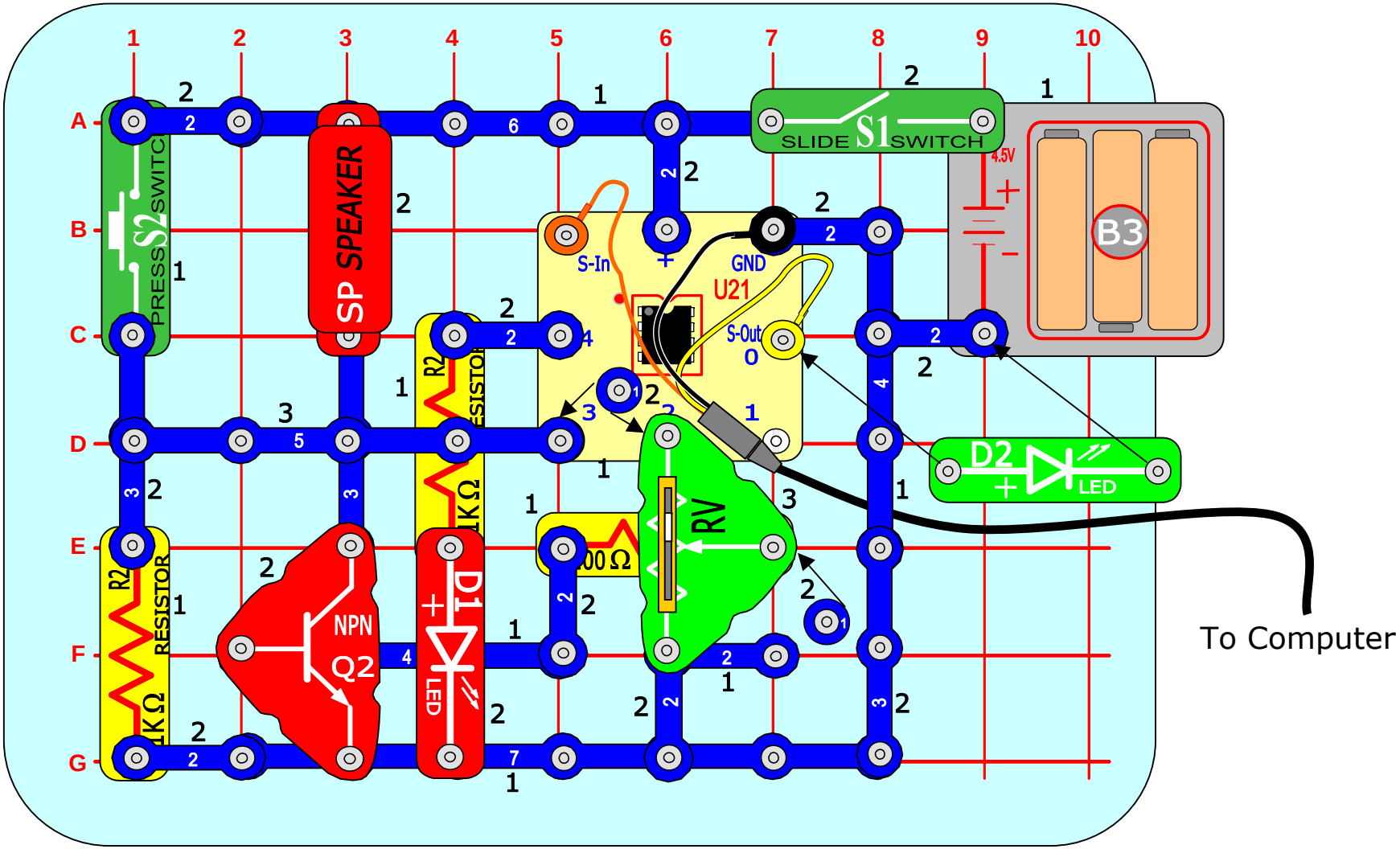
Notice how much faster this message transmits compared to the debug information. Try and press the pushbutton fast enough to miss an event. To make the transmission even faster, shorten the data to something like “S1= ”, #b1, “ times”, 13, 10. The first line will say “S1=0 times.” if the micro is turned off and back on again. The second line in the program will then be transmitted after the switch is pressed and will say “S1=1 times.”.

Data can also be stored and not transmitted until required or asked for by main computer.

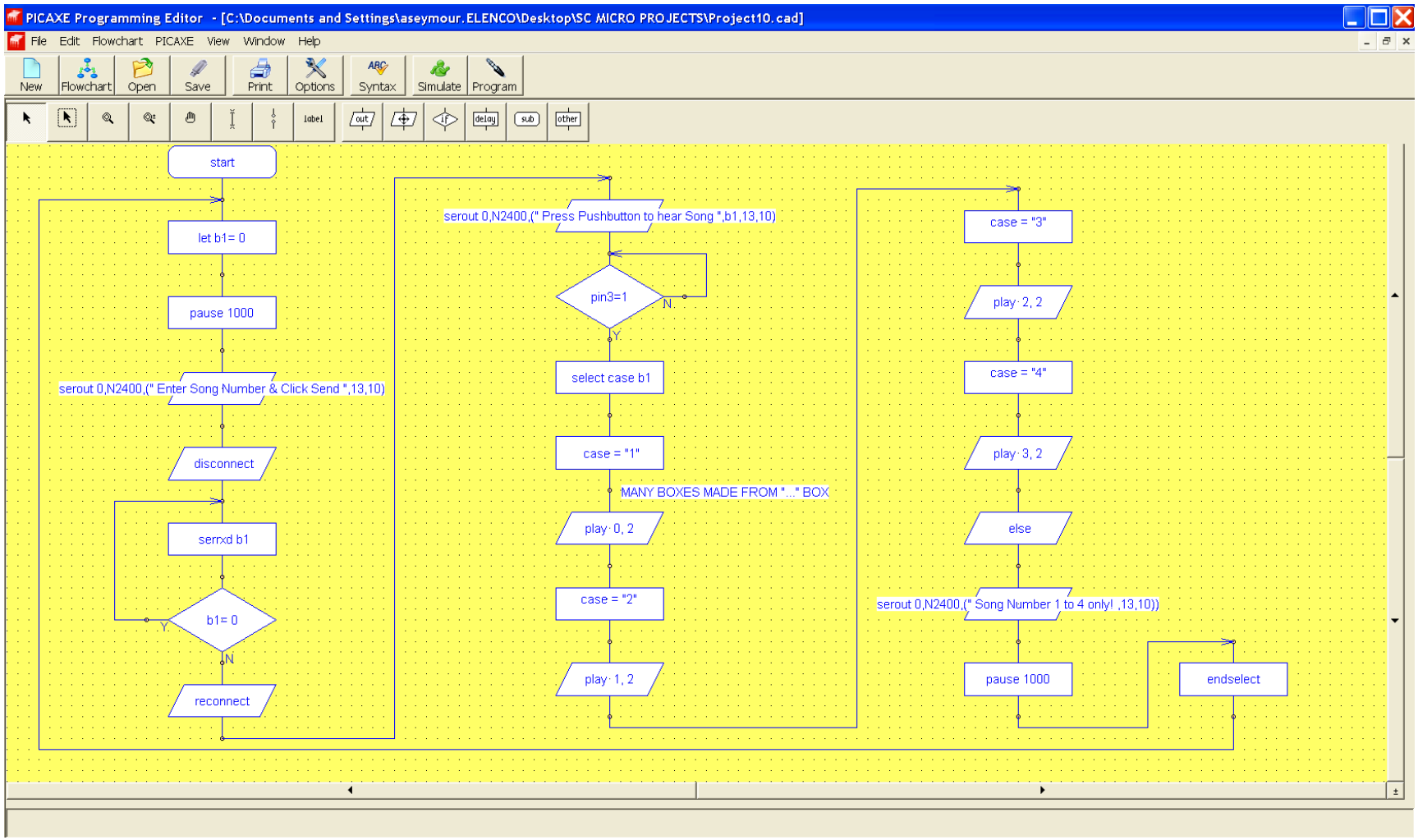
The next project will use the terminal to send information to the micro-controller, process the information, and produce an output or result of the input.

PROJECT 10 - Using Serout, Serrxd, & Terminal Window

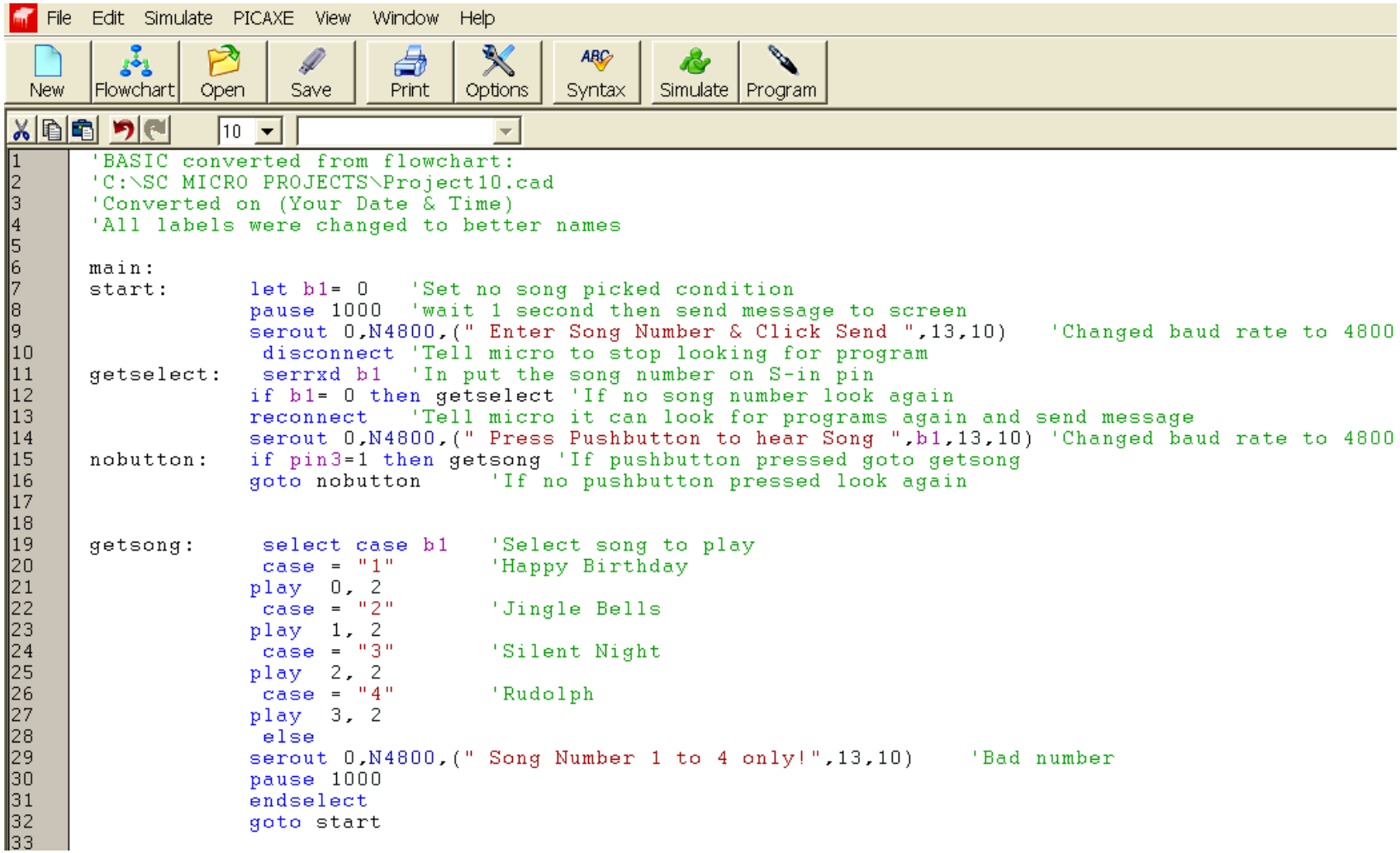
Build the circuit shown here. Place D2 over yellow lead and 2 snap as shown.



Open the program editor and make the following flowchart.



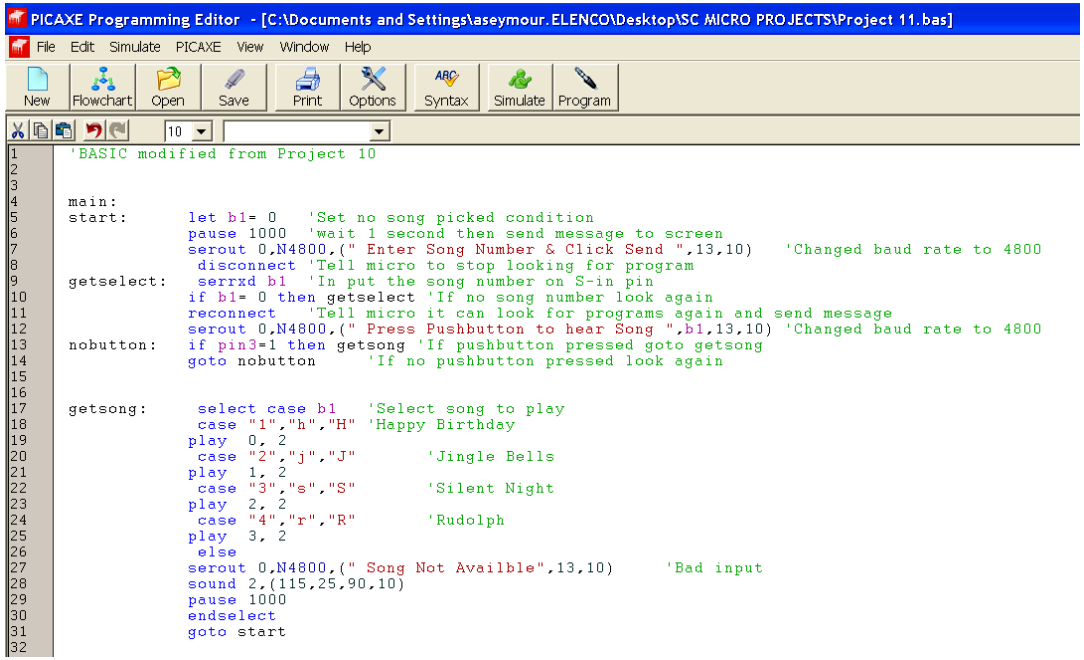
Save flowchart and then use picaxe menu to convert flowchart to Basic and edit as shown on next page.




```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project10.cad
'Converted on (Your Date & Time)
'All labels were changed to better names

main:
start:      let b1= 0    'Set no song picked condition
            pause 1000   'wait 1 second then send message to screen
            serout 0,N4800,(" Enter Song Number & Click Send ",13,10)    'Changed baud rate to 4800
             disconnect 'Tell micro to stop looking for program
getselect:   serrxd b1   'In put the song number on S-in pin
            if b1= 0 then getselect 'If no song number look again
            reconnect    'Tell micro it can look for programs again and send message
            serout 0,N4800,(" Press Pushbutton to hear Song ",b1,13,10) 'Changed baud rate to 4800
nobutton:   if pin3=1 then getsong 'If pushbutton pressed goto getsong
            goto nobutton      'If no pushbutton pressed look again


getsong:     select case b1    'Select song to play
             case = "1"        'Happy Birthday
            play  0, 2
             case = "2"        'Jingle Bells
            play  1, 2
             case = "3"        'Silent Night
            play  2, 2
             case = "4"        'Rudolph
            play  3, 2
             else
            serout 0,N4800,(" Song Number 1 to 4 only!",13,10)     'Bad number
            pause 1000
            endselect
            goto start
```

Save program then load the program into the micro-controller. Depending on your option settings the terminal window may open after download. If not open the terminal window (F8). Turn the micro-controller on and the message “ Enter Song Number and Click Send “ should appear in the terminal window. Enter the song numbers 1-4 and click the Send button. The message “Press Pushbutton to hear Song” will appear in the Terminal Window. Press the pushbutton on the snap circuit board and the song you picked should play.

PROJECT 11 – Adding Modifications




```
'BASIC modified from Project 10


main:
start:      let b1= 0   'Set no song picked condition
            pause 1000  'wait 1 second then send message to screen
            serout 0,N4800,(" Enter Song Number & Click Send ",13,10)   'Changed baud rate to 4800
             disconnect 'Tell micro to stop looking for program
getselect:   serrxd b1  'In put the song number on S-in pin
            if b1= 0 then getselect 'If no song number look again
            reconnect   'Tell micro it can look for programs again and send message
            serout 0,N4800,(" Press Pushbutton to hear Song ",b1,13,10) 'Changed baud rate to 4800
nobutton:   if pin3=1 then getsong 'If pushbutton pressed goto getsong
            goto nobutton     'If no pushbutton pressed look again


getsong:     select case b1   'Select song to play
             case "1","h","H" 'Happy Birthday
            play  0, 2
             case "2","j","J"       'Jingle Bells
            play  1, 2
             case "3","s","S"       'Silent Night
            play  2, 2
             case "4","r","R"       'Rudolph
            play  3, 2
             else
            serout 0,N4800,(" Song Not Availble",13,10)     'Bad input
            sound 2,(115,25,90,10)
            pause 1000
            endselect
            goto start
```

In the previous project a number greater than 4 would produce an error message. The songs were programmed as follows;

1. **H**appy Birthday
2. **J**ingle Bells
3. **S**ilent Night
4. **R**udolph the red nose reindeer

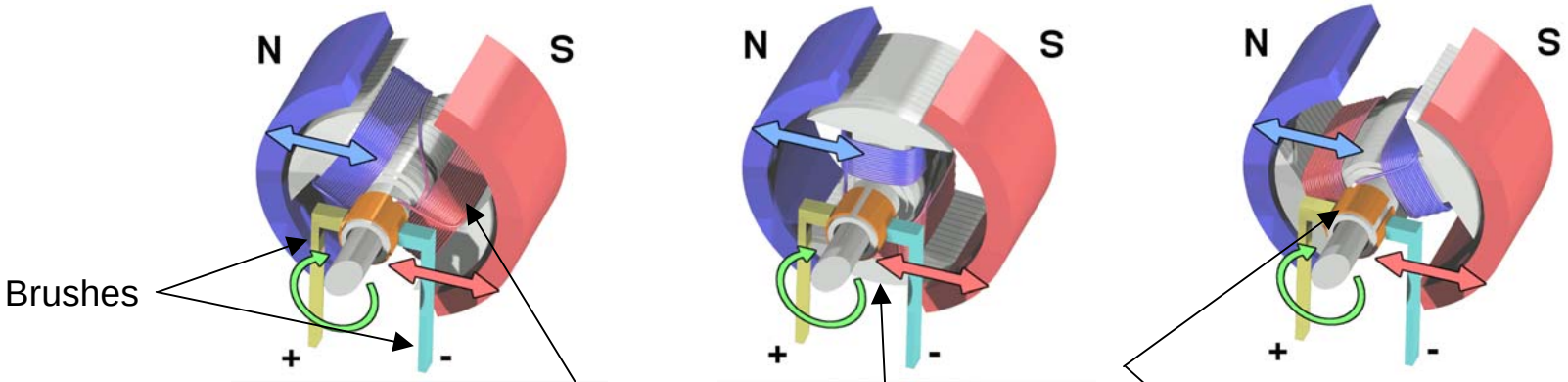
Let's modify the program so a small or capitol first letter of the song will also play that song.

On the left you can see the case equal sign has been removed and a list of inputs has been added.

What happens when a wrong input is entered? Try making your own sound for an error.

PROJECT 12 – THE DC MOTOR/GENERATOR

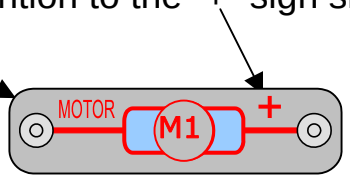
The information shown here was reproduced from the web site;
http://en.wikipedia.org/wiki/Brushed_DC_Electric_Motor#Simple_Two_Pole_DC_Motor



A simple DC electric motor. When the coil is powered, a magnetic field is generated around the armature. The left side of the armature is pushed away from the left magnet and drawn toward the right, causing rotation.	The armature continues to rotate.	When the armature becomes horizontally aligned, the commutator reverses the direction of current through the coil, reversing the magnetic field. The process then repeats.

Because DC motors use brushes and act as generators they will produce voltages that interfere with the micro-controller program. The DC motor provided with your XP Snap Circuits® parts was modified to reduce this problem. The motor number should be MX and the motor should have 3 capacitors inside the case. The following circuit will test the program while motor is running with and without a load.

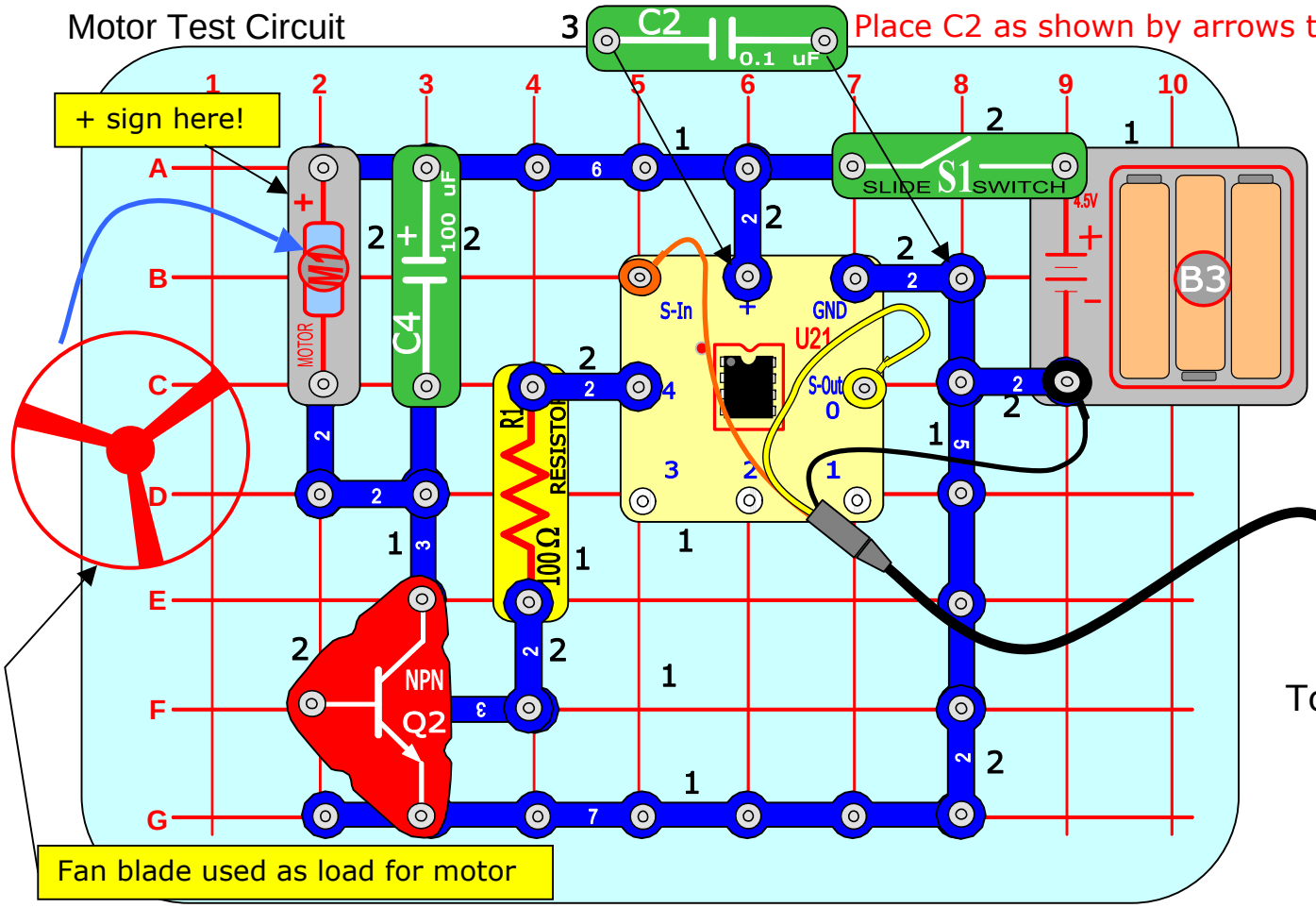
The following symbol is used to represent the DC Motor. Pay attention to the "+" sign since it will determine the direction of rotation when power is applied.

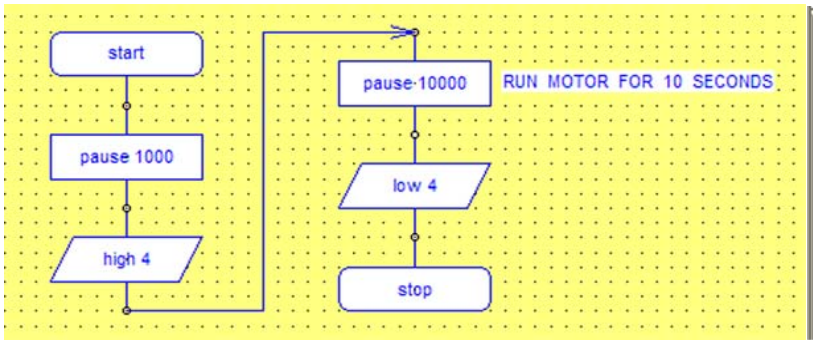


Build the circuit shown on the next page and open the program editor to make the flow chart that follows.

Convert flow chart to the basic program shown. Use only fresh alkaline batteries in this project.

Motor Test Circuit





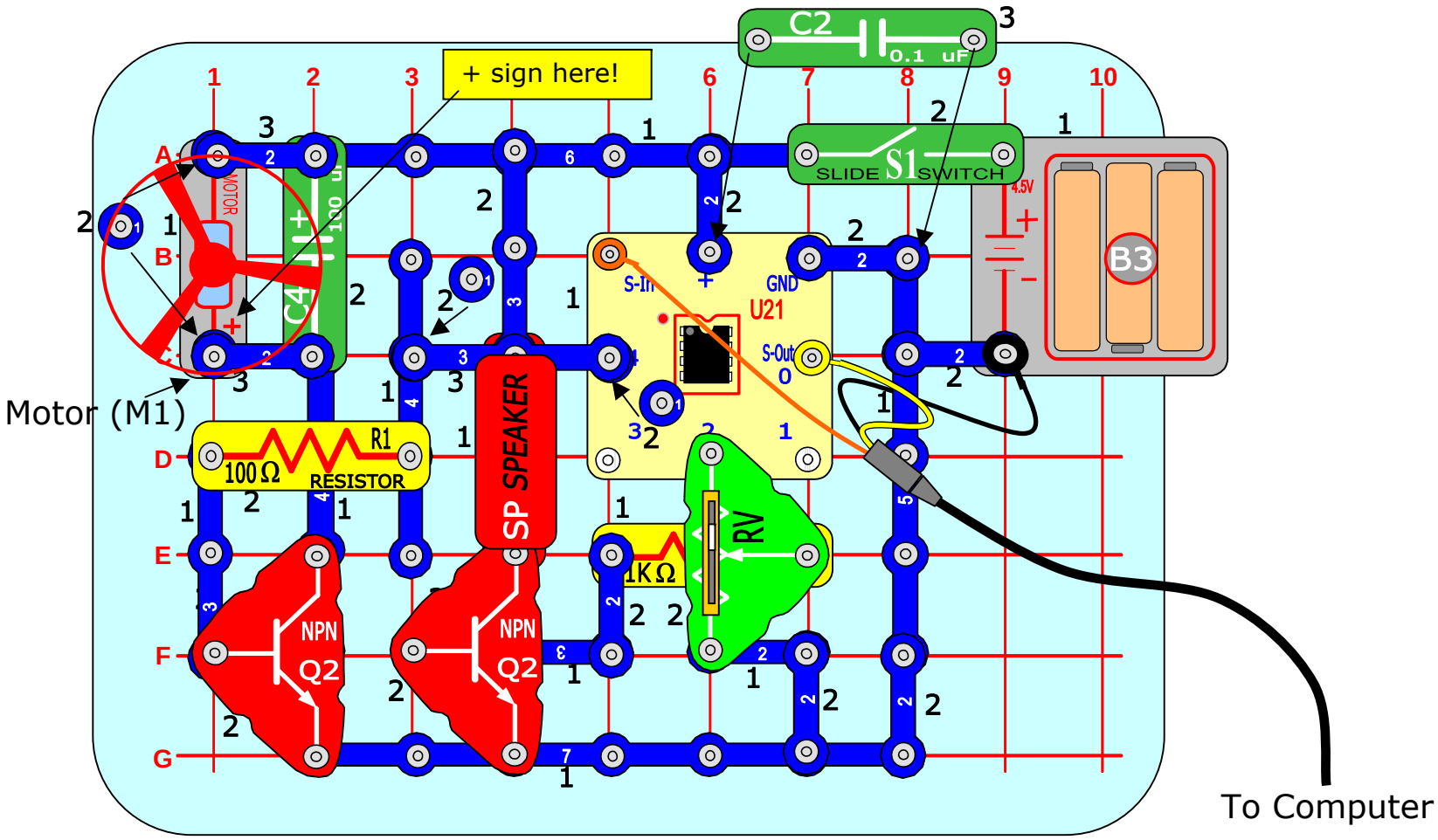

```
'BASIC converted from flowchart project12:
'C:\SC MICRO PROJECTS\PROJECT12A.CAD
'Converted on DATE at TIME


main:
        pause 1000
        high 4
        pause 10000
        low 4
        stop
```

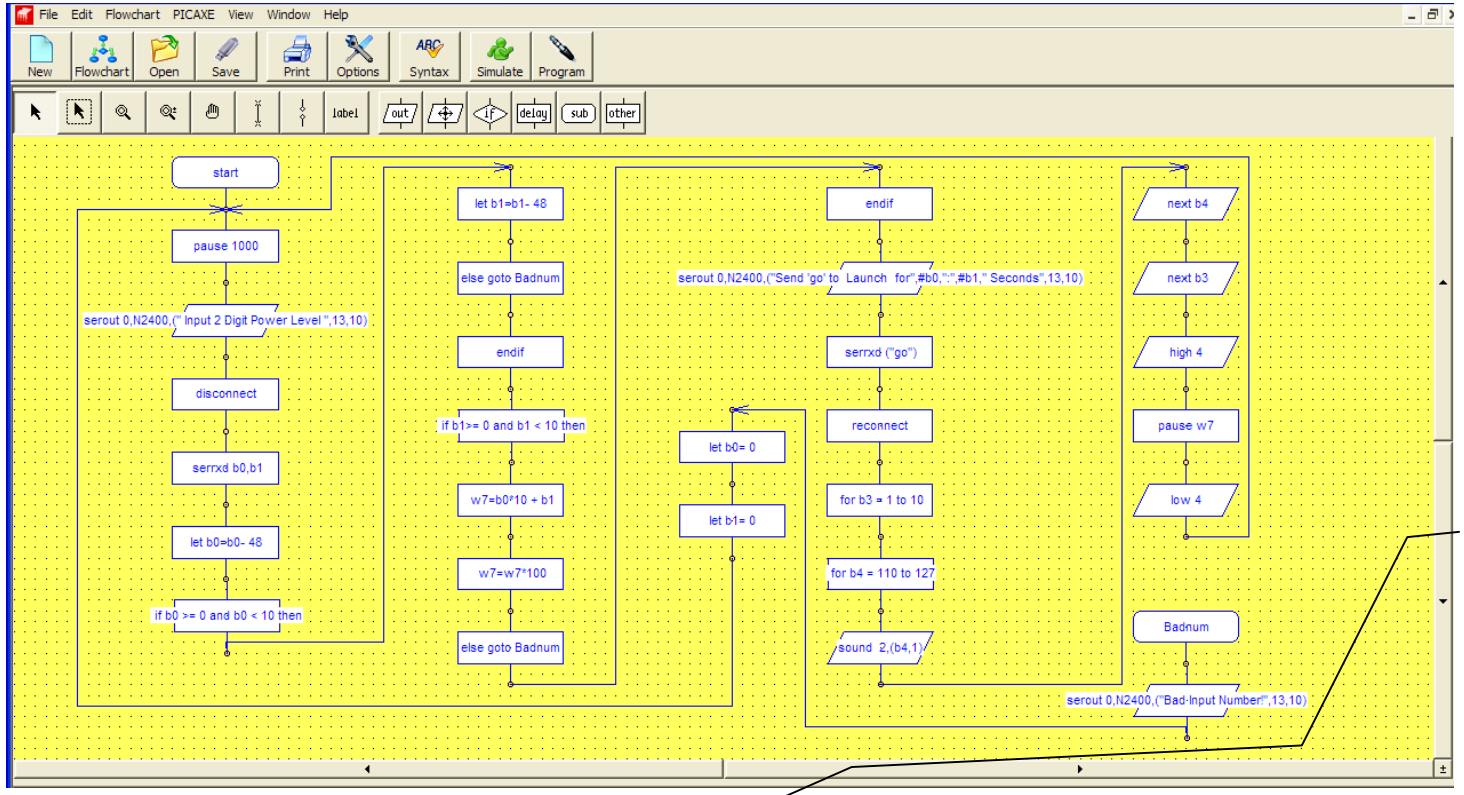
Download and run program with load (Fan) and without load. Time should be 10 seconds on each run. If program stops before 10 seconds check batteries. Always use fresh alkaline batteries.

SECTION 3: PROGRAMMING FOR SNAP CIRCUITS®

PROJECT 13 – THE FLYING SAUCER

Build the following Snap Circuit® … note the "+" on the motor.





Build or download the flowchart on the left. Use the PICAXE menu to convert to the program below and modify as shown.

After conversion the baud rate was changed from N2400 to N4800. All other baud rates were also changed to N4800.
Labels were also changed after conversion.
Program the micro-controller and use F8 key to open terminal and use launch pad to send flying propeller to different heights. Enter 05 for .5 seconds of motor spin. And 45 for 4.5 seconds of motor spin. The longer the motor spins the higher the fan will fly. Go must be entered after the time to launch the fan. A warning sound will play before each launch.

File Edit Simulate PICAXE View Window Help

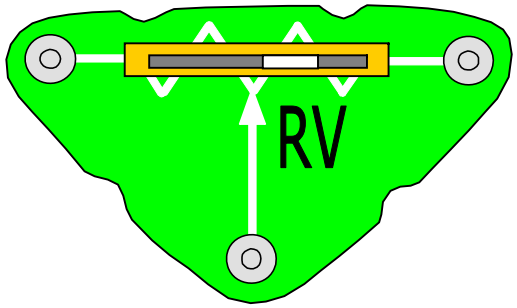
New Flowchart Open Save Print Options Syntax Simulate Program

```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project13.cad
'Converted on (Your Date & Time)
'Labels renamed & baud changed to 4800


main:
        pause 1000
        serout 0,N4800,(" Input 2 Digit Power Level ",13,10)
        disconnect                      'Stop looking for downloads and use S-In as input
         serrxd b0,b1                   'Input 2 ascii characters
        let b0=b0- 48                   'Convert first ascii character to decimal number
         if b0 >= 0 and b0 < 10 then    'Is first character a digit from 0 to 9?
        let b1=b1- 48                   'If yes convert second ascii to a number
        else goto Badnum                'If no goto bad number routine
        endif                           'Finished with first character
         if b1>= 0 and b1 < 10 then     'Is second ascii a digit
         w7=b0*10 + b1                  'If yes calculate delay in milliseconds
         w7=w7*100                      'Convert to seconds and one decimal place
        else goto Badnum                'If no goto bad number routine
        endif                           'Done checking input - Range allowed = 0 to 9.9 sec.
        serout 0,N4800,("Send 'go' to  Launch  for ",#b0,":",#b1," Seconds.",13,10)
         serrxd ("go")                  'Look for "go" command
        reconnect                       'Done using S-In so start looking for downloads again
         for b3 = 1 to 10               'Do 10 siren sounds
         for b4 = 110 to 127            'Make siren sound
        sound  2,(b4,1)                 'Do Alarm for prelaunch
         next b4
         next b3
        high 4                          'Start motor
        pause w7                        'Run for launch time set
        low 4                           'Stop motor
        goto main

Badnum:
        serout 0,N4800,("Bad Input Number!",13,10)
        let b0= 0
        let b1= 0
        goto main
```

PROJECT 14 – Analogue Sensors & Analogue to Digital Conversion (adc)

Analogue Sensors:
An analogue sensor measures continuous signals such as light, sound level, position, or voltage.
Common examples of analogue sensors are:

• Variable Resistor (RV)

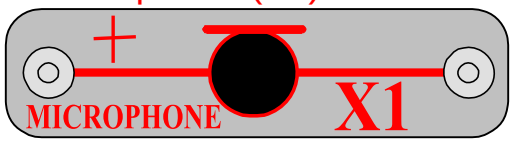


The variable resistor provides a varying voltage dependent on the center arm position. A voltage signal from 0 to 4.5 volts can be placed on pin 1 by adjusting the slider. The micro-controller converts this analog input into a digital number that can be represented by a decimal number in the range 0 to 255 (8 bits) or 0 to 1023 (10 bits).

• The “Photo Resistor” (RP) or “Light Dependant Resistor” (LDR)

The photo resistor or light dependent resistor provides a varying voltage dependent on the amount of light. A voltage signal that changes with the intensity of light can be placed on a microchip input by using the photo resistor. The micro-controller converts this analog input into a number that represents the amount of light on the resistor.

• Microphone (X1)



The microphone provides a varying voltage dependent on the amount of sound present. A voltage signal that changes with the intensity of sound can be placed on a microchip input by using the microphone. The micro-controller converts this analog input into a number that represents the amount of sound present.

readadc

Syntax:

READADC channel,variable

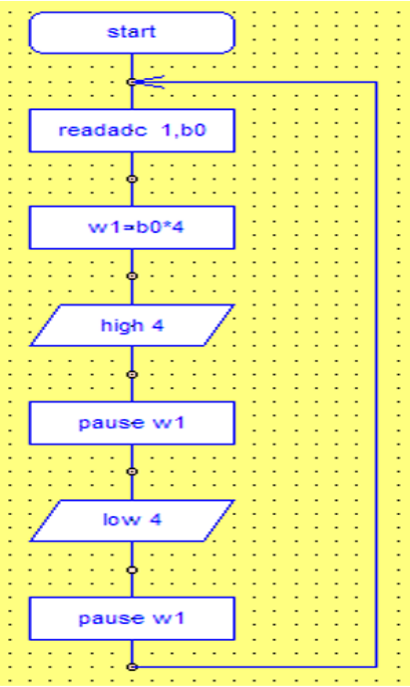
- channel is a variable or a constant that sets the input pin (1,2,or 4)
- variable is the name of the variable that holds the converted data.

Function:

Read the ADC channel (8 bit resolution) contents into variable.

Information:

The readadc command is used to read the analogue value from the micro-controller input pins 1, 2, or 4. The readadc command converts this value to an 8-bit variable. An 8-bit resolution analogue input will provide 256 different analogue readings (0 to 255) over the full voltage range (e.g. 0 to 4.5V). Note that not all inputs have internal ADC capability. Use the readadc10 command to read the full 10-bit value.



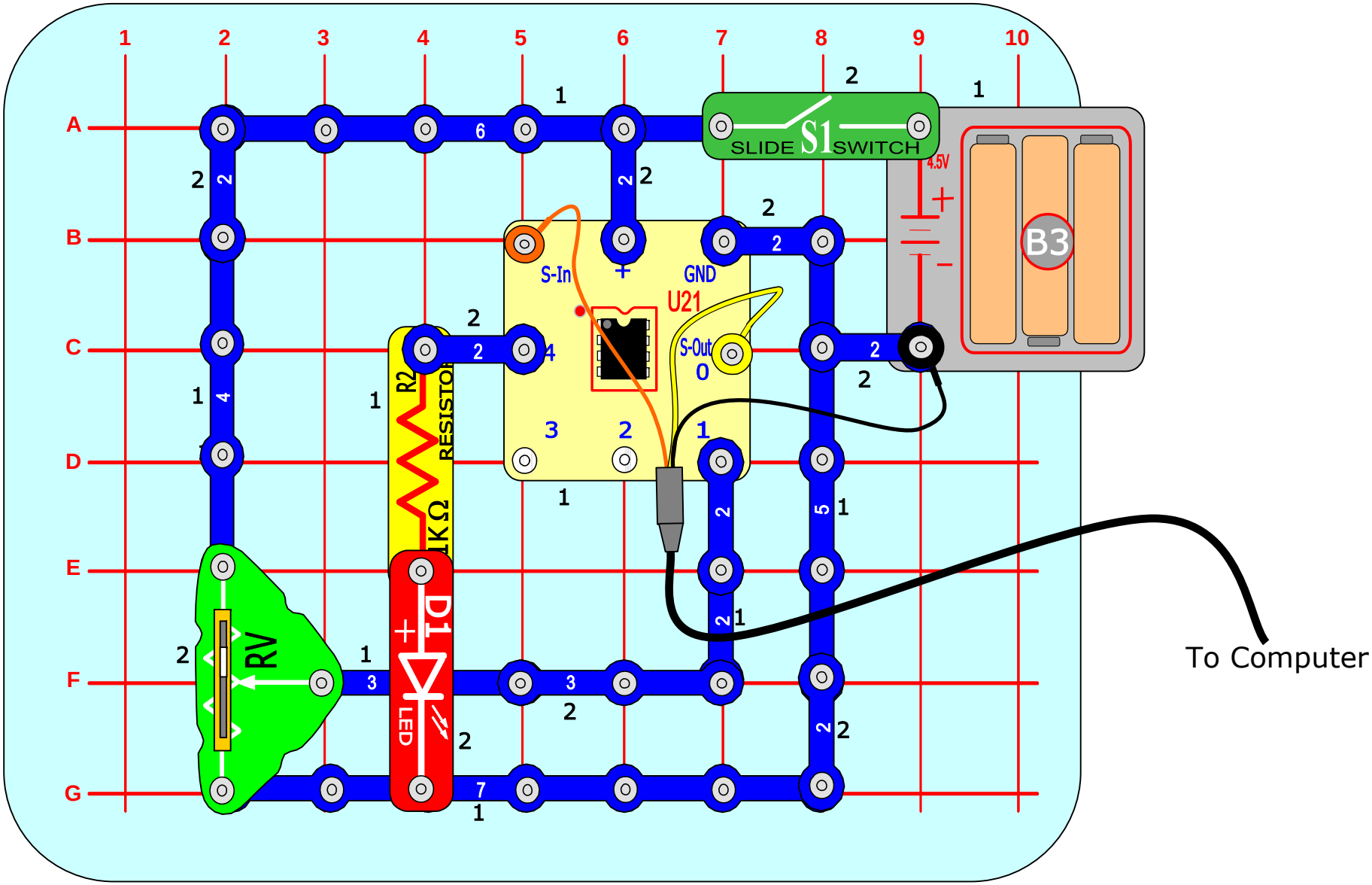
Convert to Basic,

Program:

```
main:
        readadc  1, b0
        w1=b0*4
        high 4
        pause w1
        low 4
        pause w1
        goto main
```

Enter the flow chart shown here into the program editor then build the circuit shown on the next page.

Download program and adjust RV for LED blinking rate.



Edit the previous program to use 10-bit accuracy as shown here.

readadc10

Syntax:

READADC10 channel,wordvariable

- channel is a variable or a constant specifying the input pin (1, 2, or 4)
- wordvariable is the name of the wordvariable that holds the converted data.

Function:

Read the ADC channel (10 bit resolution, 0 to 1023) contents into a wordvariable.

Information:

The readadc10 command is used to read the analogue value into the micro-controller with 10-bit accuracy. Since the result is a 10-bit number, a wordvariable must be used. Note that only input pins 1, 2, or 4 have internal ADC functionality.

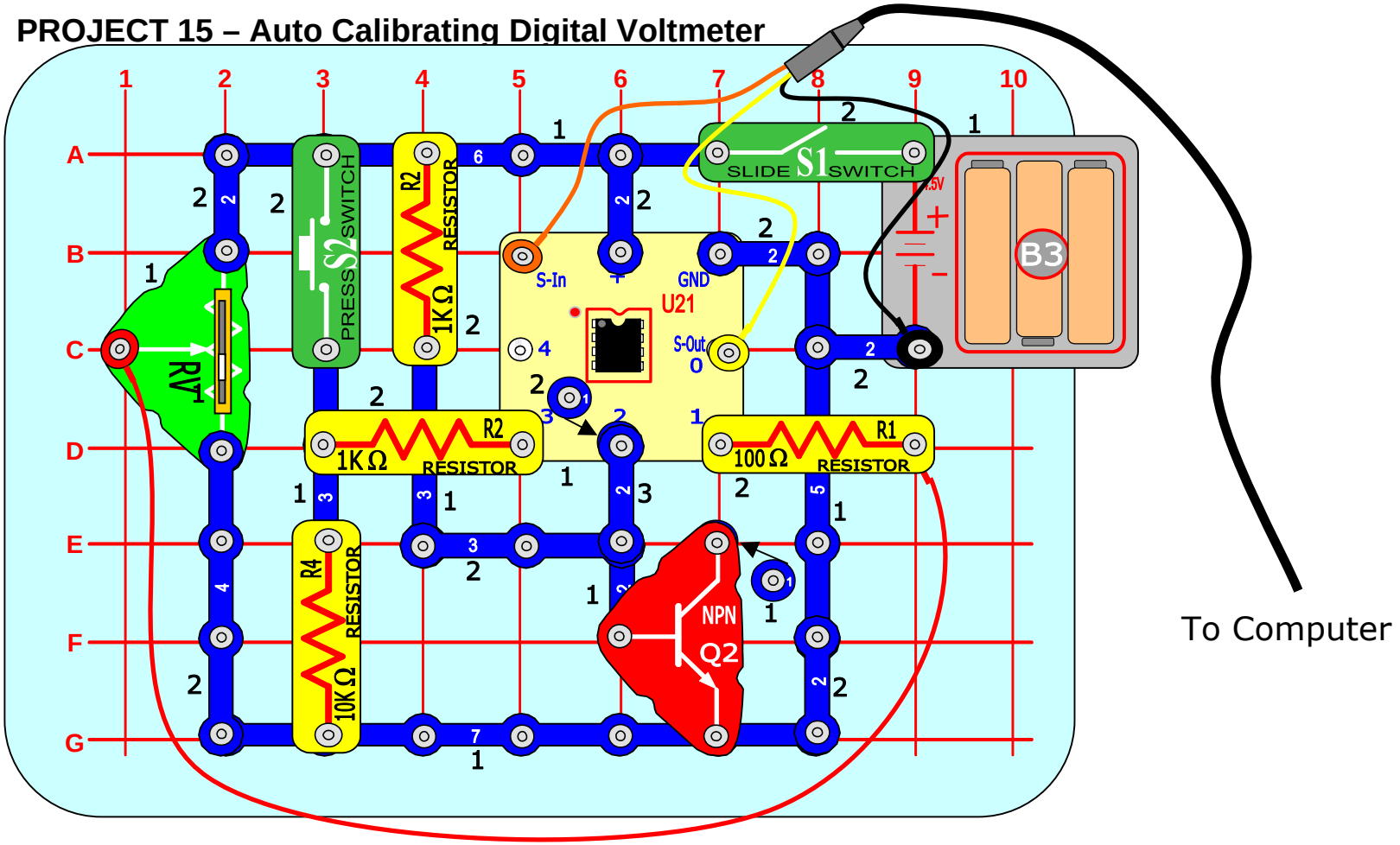
```
1   'BASIC converted from flowchart:
2   'C:\SC MICRO PROJECTS\PROJECT14.CAD
3   'Converted on DATE at TIME
4
5
6   main:
7           readadc10  1,w0     'check RV position
8           high 4              'turn LED on
9           pause w0            'delay
10          low 4               'turn LED off
11          pause w0            'delay
12          goto main           'repeat
```

Download this new program and note the difference in program length and functionality.

8 Bit Conversion Program	10 Bit Conversion Program
18 bytes	13 bytes
Delay between LED flashing increments by 4, for example; 0,4,8,12,16, ……..,1012,1016,1020	Delay between LED flashing increments by 1, for example; 0,1,2,3,4,5,6, …… 1021,1022,1023

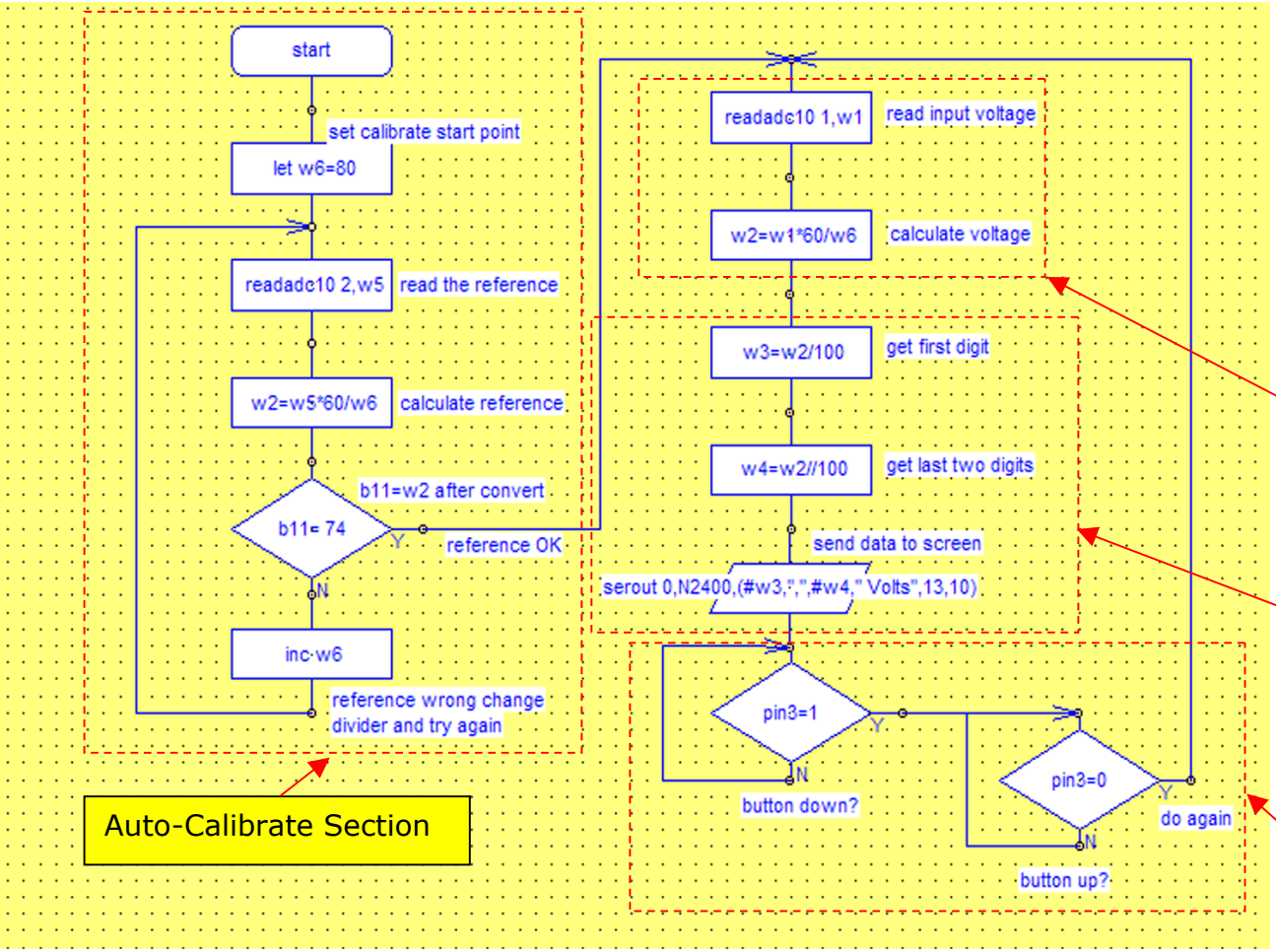
In some cases the 8 bit resolution is adequate to do the job, but when a finer resolution is required, use the readadc10 function.

PROJECT 15 – Auto Calibrating Digital Voltmeter



The Snap Circuit® shown above uses the base-emitter junction of transistor Q2 as a reference and calibrates the internal A to D for correct voltage readings. This self-calibration technique eliminates the error that would occur as batteries discharge. It also eliminates A to D differences from circuit to circuit. After building this circuit, use the flow chart and download the program shown on the next page. Open the terminal window by pressing F8 or using the drop down under the PICAXE® menu. Adjust the RV slider for different voltages and press the S2 pushbutton to get a reading. When the slider is all the way up, the voltage will equal the battery voltage.

Voltmeter Flowchart;



In the Auto-Calibrate section the base-emitter voltage on Q2 is read on pin 2 of the micro-controller. Variable w6 is then adjusted for the correct reading. This is only calculated once each time the program is started.

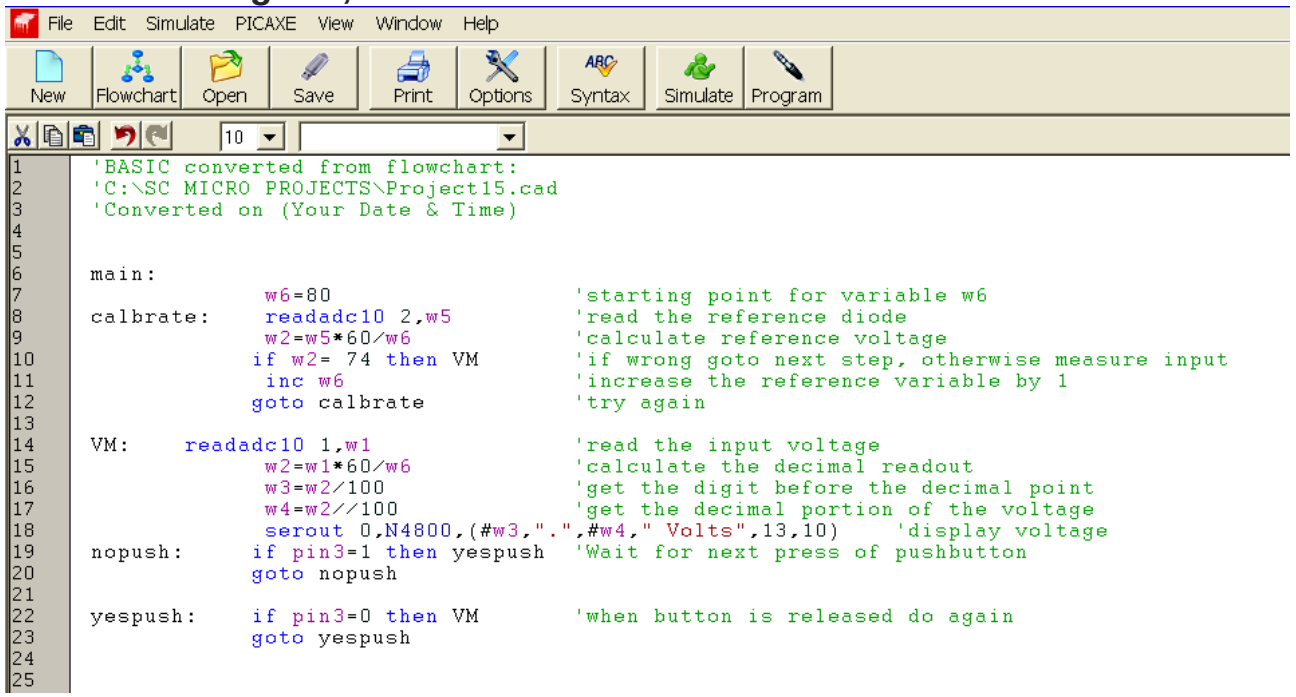
The RV voltages are then read on pin 1 of the micro-controller and calculated using variable w6.

The digital number is formatted to read as a decimal and sent to the terminal display.

The process will repeat after the pushbutton is pressed and released.

Since the “if” box in the flow chart does not support word variables, b11 was used with a note to make it “w2” after converting to basic. The number “74” in the “if” box is a guess at the base-emitter voltage (.74 volts) and can be changed to the actual voltage measured from base to ground when circuit is active. After converting and editing, the basic program should read as shown on next page.

Voltmeter Program;



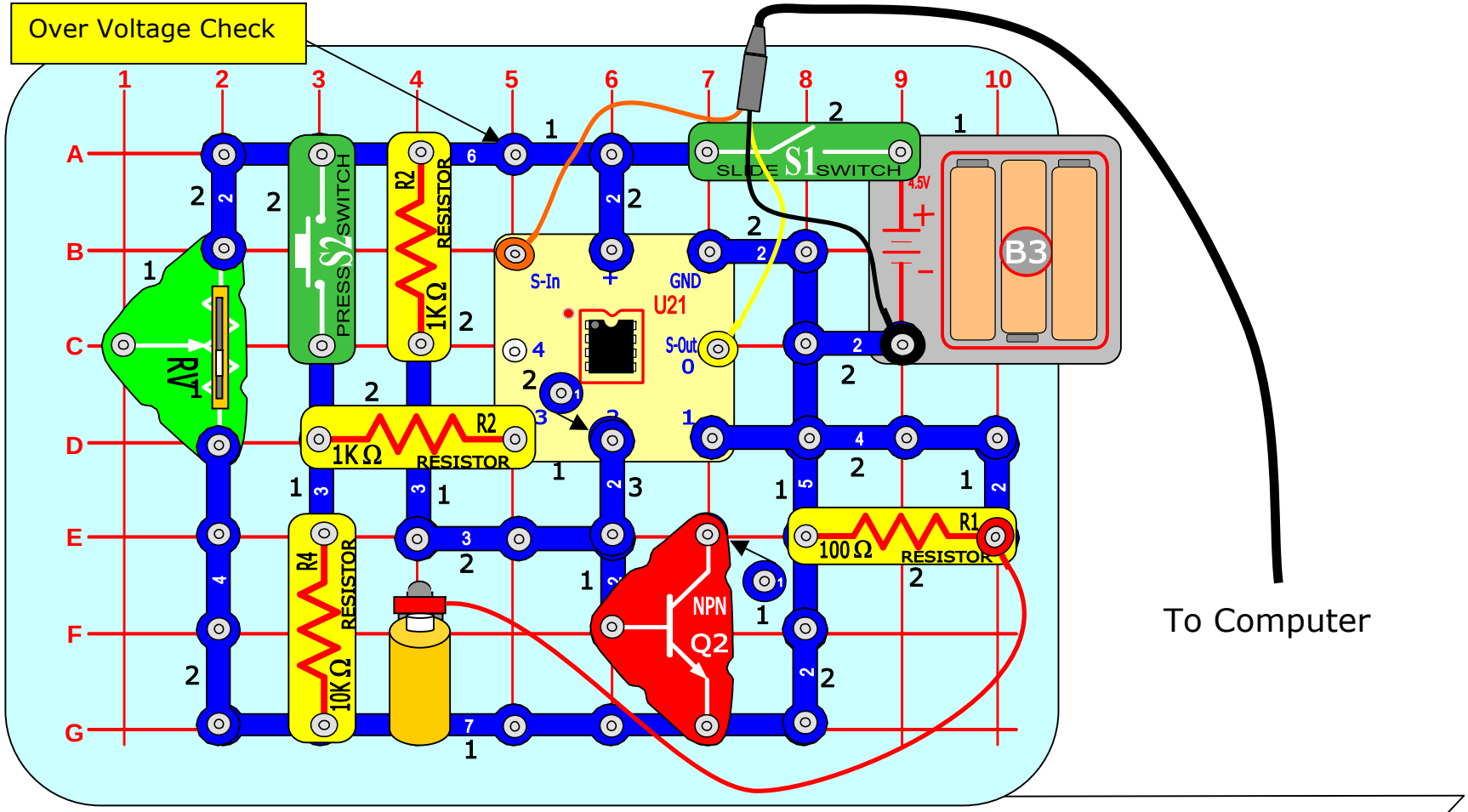

```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project15.cad
'Converted on (Your Date & Time)


main:
              w6=80                    'starting point for variable w6
calbrate:     readadc10 2,w5           'read the reference diode
              w2=w5*60/w6              'calculate reference voltage
             if w2= 74 then VM         'if wrong goto next step, otherwise measure input
              inc w6                   'increase the reference variable by 1
             goto calbrate             'try again

VM:     readadc10 1,w1                 'read the input voltage
              w2=w1*60/w6              'calculate the decimal readout
              w3=w2/100                'get the digit before the decimal point
              w4=w2//100               'get the decimal portion of the voltage
              serout 0,N4800,(#w3,".",#w4," Volts",13,10)    'display voltage
nopush:      if pin3=1 then yespush    'Wait for next press of pushbutton
             goto nopush

yespush:     if pin3=0 then VM         'when button is released do again
             goto yespush
```

Make sure baud rate is at 4800 when using terminal window.

The variable w5 stores the reading between 0000 and 1023 that represents the reference voltage. If the battery voltage equals 4.5 volts and the reference voltage equals .74 volts the w5 variable should read (.74/4.5)x1023 or 168 (decimals not allowed). The highest number the micro-controller can use mathematically is 65,335. If the highest number for w5 is 1023, multiplying by 60 will not exceed this limit (1023x60=61,380). These larger numbers allow for two decimals in the final reading. The program then calculates the voltage at pin 2 using the number 80 for w6 as follows … w2=(w5 x 60) / w6 or VR = (168 x 60) / 80 or 126. Since the reference voltage is 74 (or .74 volts), the variable w6 is increased by 1 and the calculation is repeated. When the 74 number is calculated the variable w6 has been found and the micro-controller uses the 1 pin to measure voltages.

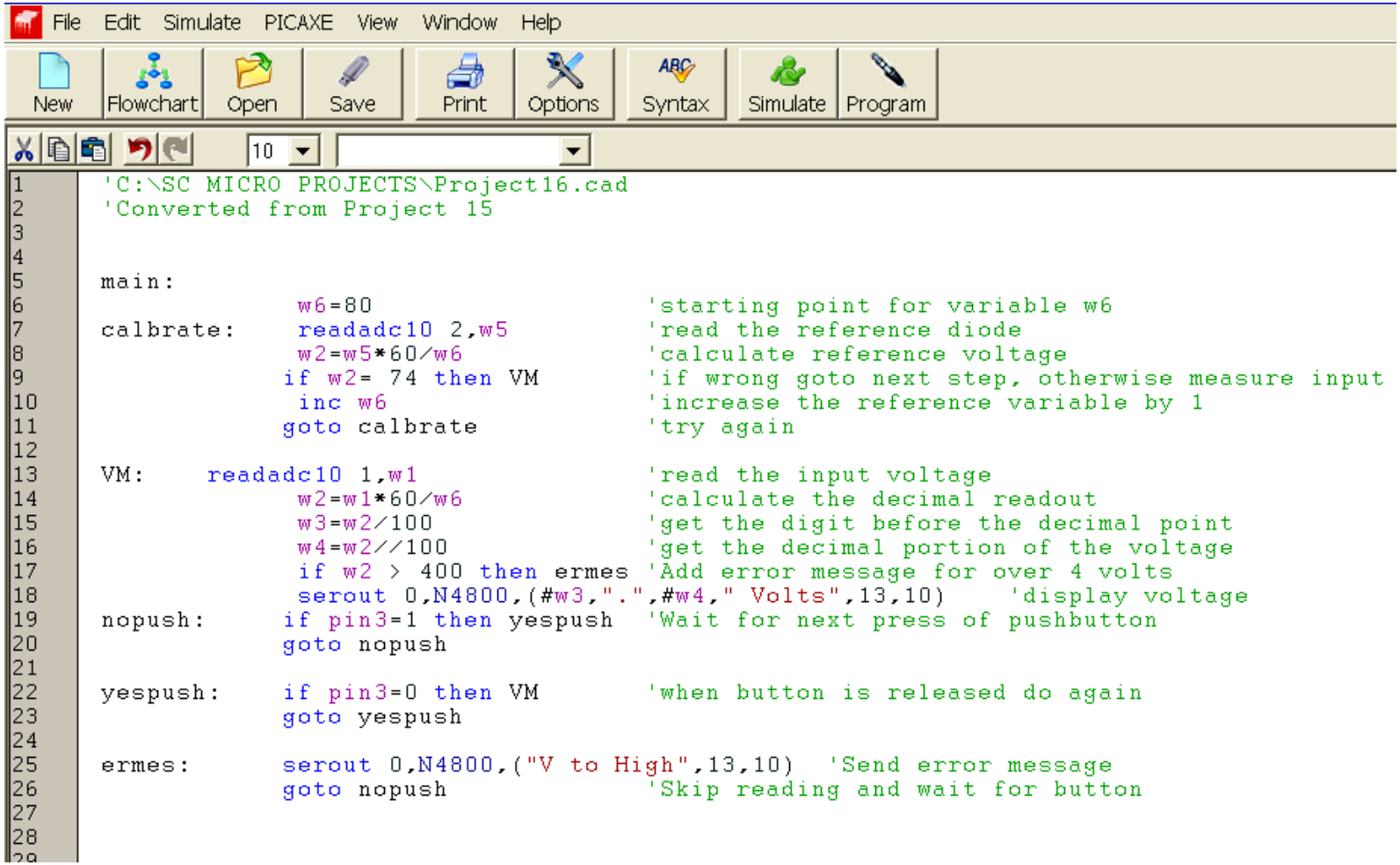
By comparing the input voltages to a known reference many undesired variables can be eliminated.

PROJECT 16 – Battery Tester (Batteries under 4 volts)



The Snap Circuit® shown above uses the voltmeter program to check batteries up to 4 volts. An error message is added for voltages over 4 volts. Hold the battery to be tested in the position shown here, then press the S2 pushbutton to get a reading. Make sure the bottom of the red snap on the wire touches the + terminal of the battery, and the other side of the battery is pressed onto the ground snap. The battery will be loaded at 10 milliamps per volt during the test.

Battery Tester Program;

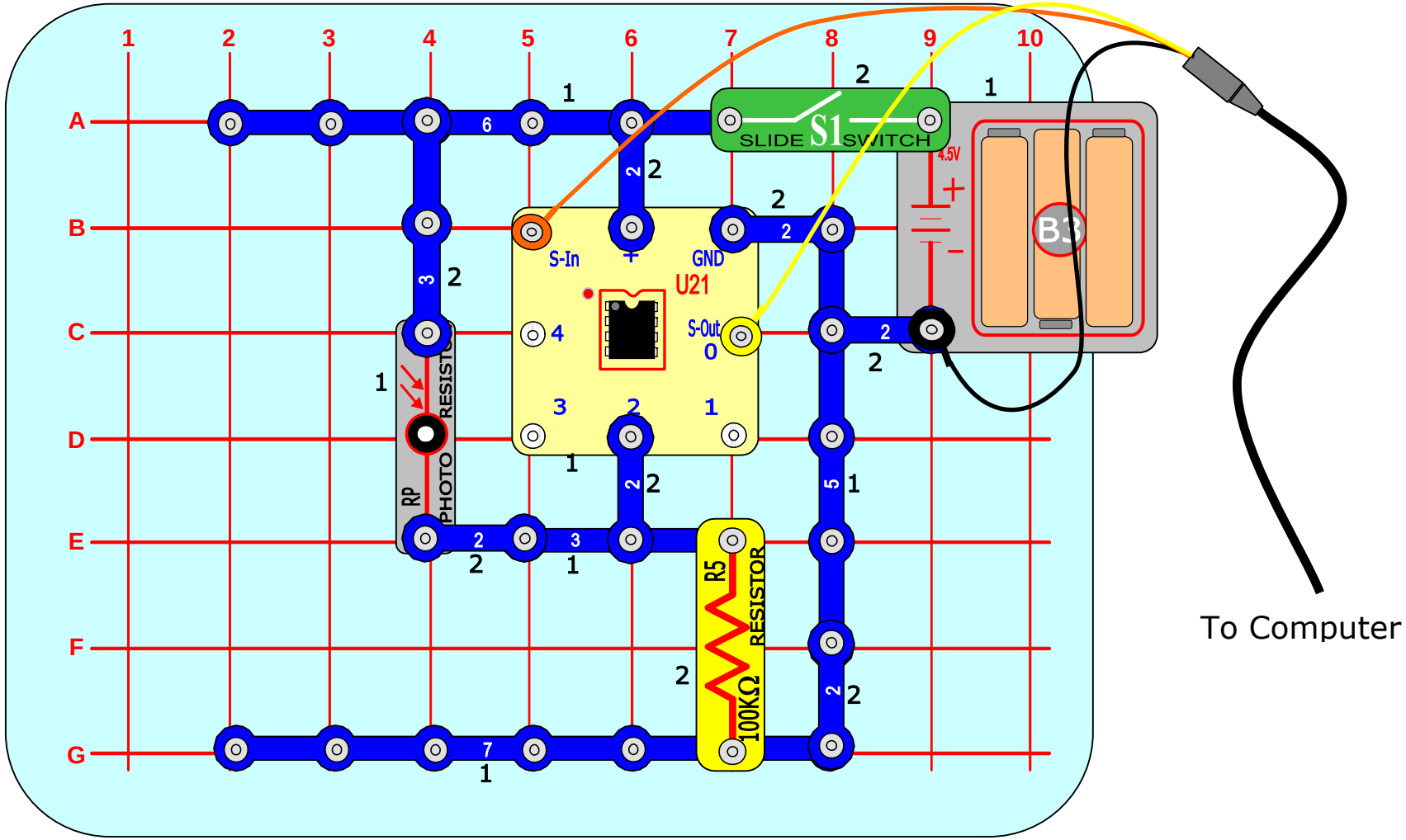



```
'C:\SC MICRO PROJECTS\Project16.cad
'Converted from Project 15


main:
                w6=80                   'starting point for variable w6
calbrate:       readadc10 2,w5          'read the reference diode
                w2=w5*60/w6             'calculate reference voltage
               if w2= 74 then VM        'if wrong goto next step, otherwise measure input
                inc w6                  'increase the reference variable by 1
               goto calbrate            'try again

VM:     readadc10 1,w1                  'read the input voltage
                w2=w1*60/w6             'calculate the decimal readout
                w3=w2/100               'get the digit before the decimal point
                w4=w2//100              'get the decimal portion of the voltage
                if w2 > 400 then ermes  'Add error message for over 4 volts
                serout 0,N4800,(#w3,".",#w4," Volts",13,10)     'display voltage
nopush:        if pin3=1 then yespush   'Wait for next press of pushbutton
               goto nopush

yespush:       if pin3=0 then VM        'when button is released do again
               goto yespush

ermes:         serout 0,N4800,("V to High",13,10)  'Send error message
               goto nopush              'Skip reading and wait for button
```

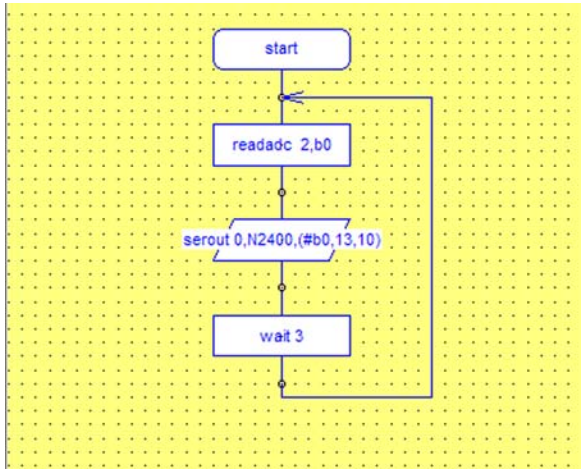
Modify the voltmeter program in project 15 to be as shown above. This program adds an error message if the battery voltage being checked gets close to or greater than the voltage level of the micro-controller. If fresh batteries are installed in the Snap Circuit® battery holder, the battery checker circuit can be used to check the voltage on any battery up to 4.0 volts.

Project 17, The Photo Resistor (RP) or Light Dependent Resistor

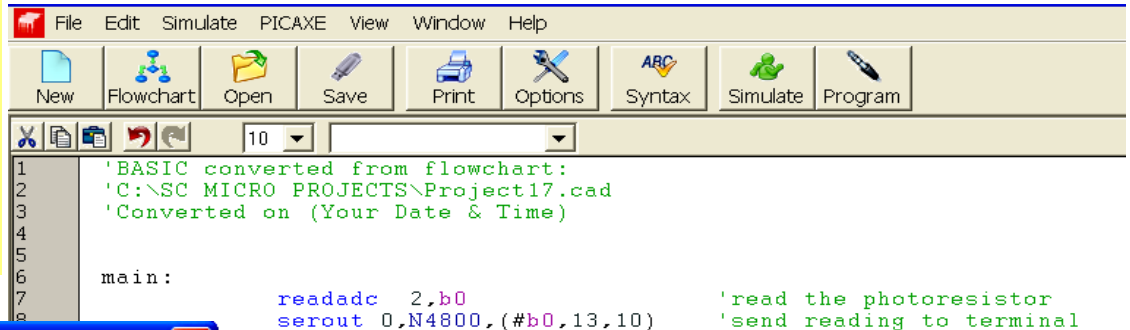
RP is an example of an analogue sensor that drops from a very high resistance to a low resistance as light is increased. It is connected between the micro-controller input pin 2 and ground. A 100kΩ resistor from ground to pin 2 allows the voltage on pin 2 to fall when it is dark and rise when there is light on RP. Dark should be close to 0 and bright should be close to 256. Build the snap circuit shown here.



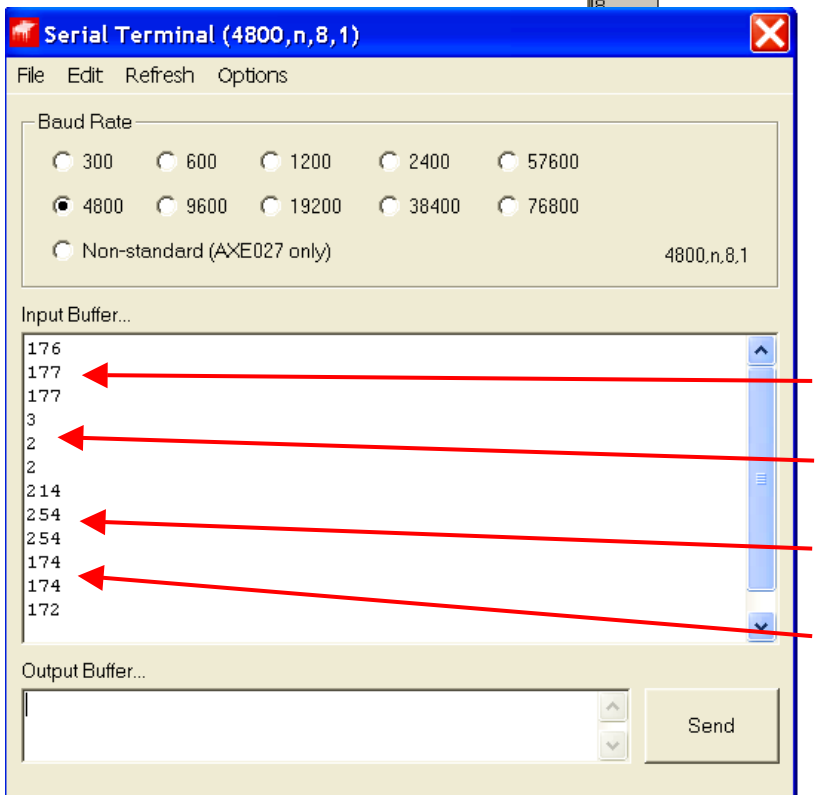
Next page shows program and flowchart.



Draw the flowchart at the left and use the PICAXE® drop down menu to convert the flowchart to the basic program shown below. Download the program into the micro-controller, clear the message window, and press f8 button to open the terminal window.



Change from N2400 to N4800



By placing the Snap Circuit® in normal room light you should get readings similar to the ones shown here.

Normal room light.

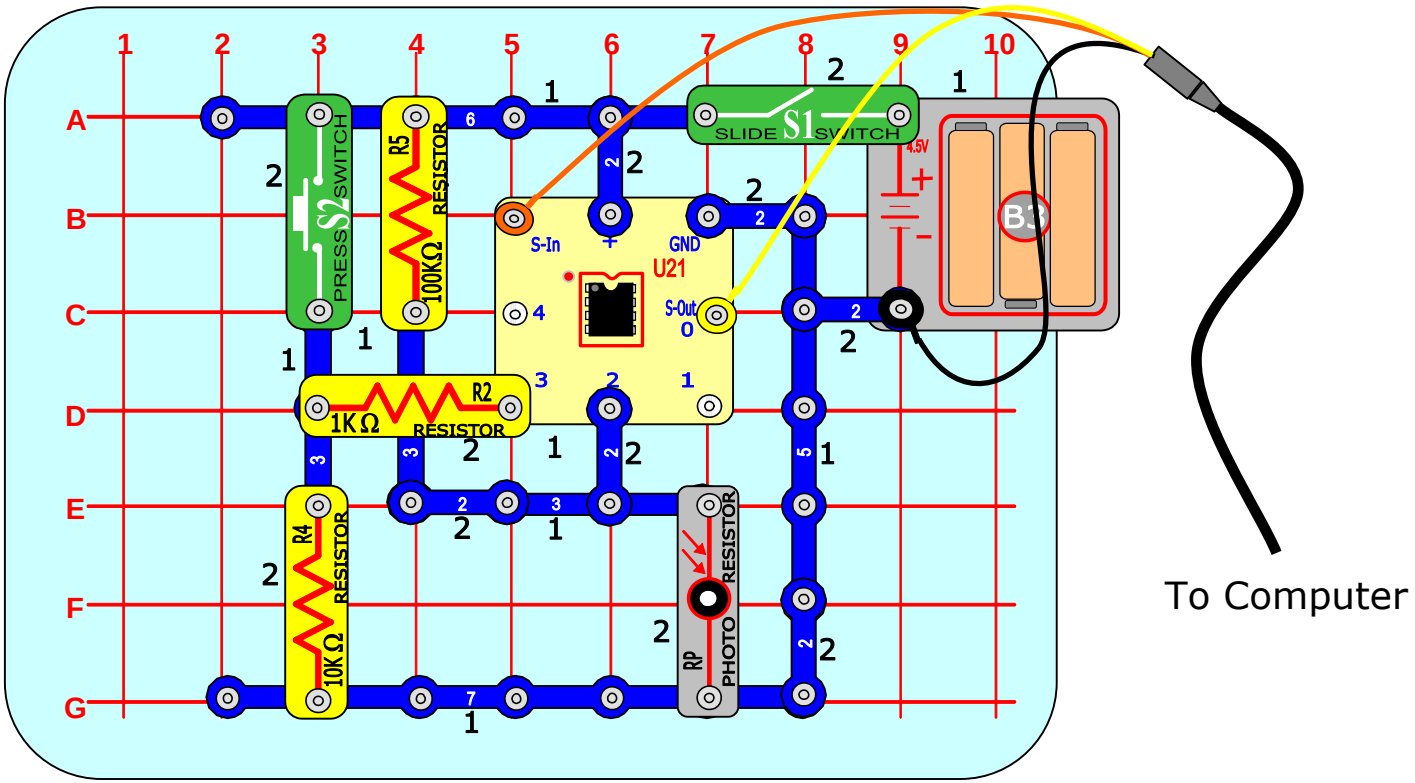
Hand over the RP sensor (Dark).

Flashlight onto RP sensor (Bright).

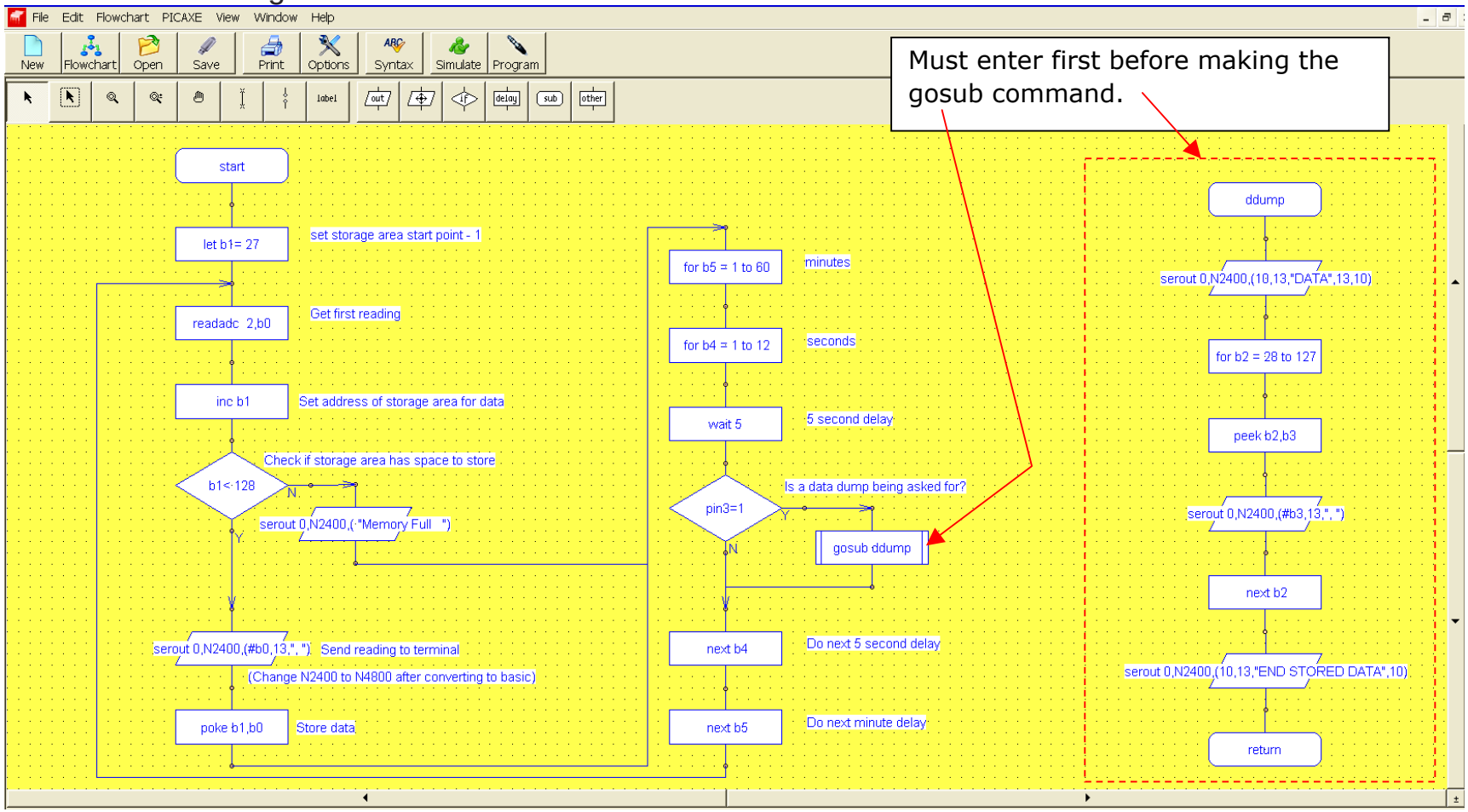
Normal room light.

Project 18, Introduction to Data Loggers.

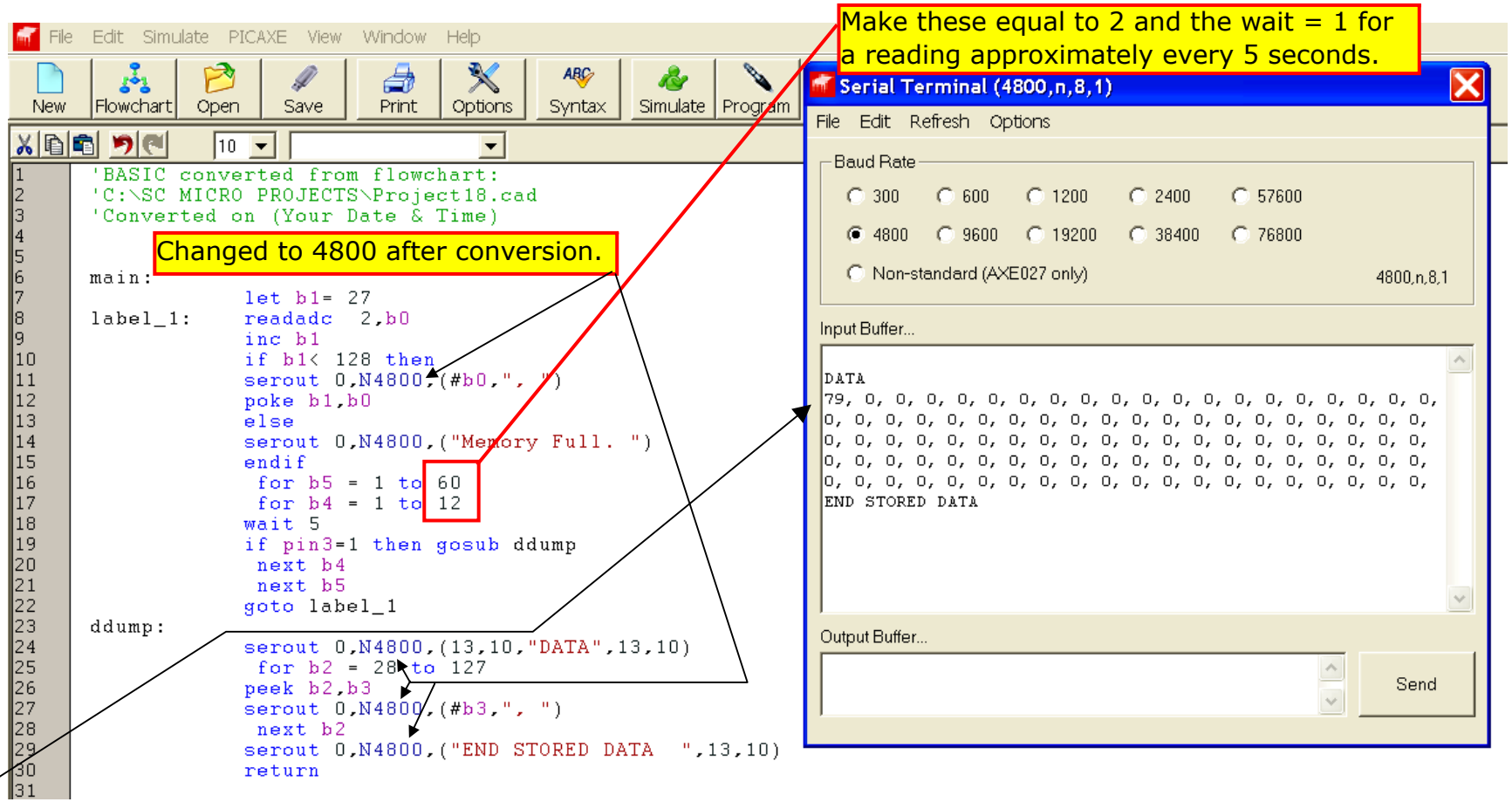
Technically speaking, a data logger is any device that can be used to store data. This includes many data acquisition devices such as plug-in boards or serial communication systems, which use a computer as a real time data recording system. However, most instrument manufacturers consider a data logger a stand alone device that can read various types of electrical signals and store the data in internal memory for later download to a computer.
The advantage of data loggers is that they can operate independently of a computer, unlike many other types of data acquisition devices. Data loggers are available in various shapes and sizes. The range includes simple economical single channel fixed function loggers to more powerful programmable devices capable of handling hundreds of inputs. The Snap Circuit below is a single channel fixed light intensity logger. Build this circuit.



Create the following flowchart:

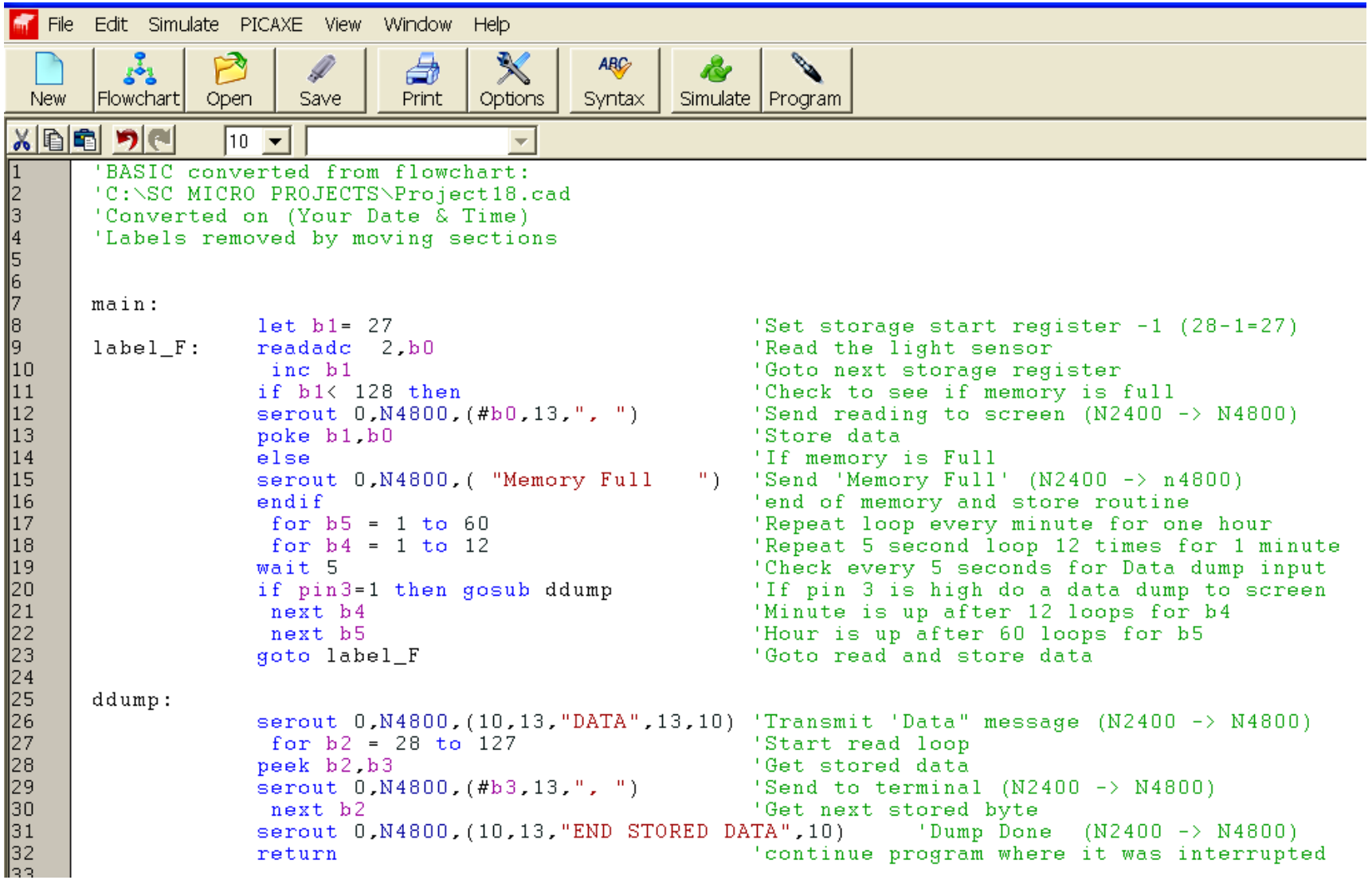


In the above flowchart the data is stored in registers 28 to 127 with the “poke” command and retrieved by using a subroutine called ddump with the “peek” command. The “gosub” command only stops the time keeping process to retrieve the data and send it to the terminal. Time keeping is then resumed where it left off by the return command at the end of the ddump subroutine. Use the PICAXE® drop down menu and convert the flow chart to a program similar to the one shown on the next page. Be sure to add your own notes and change labels to make your program easy to understand when you revisit this program at a later date.



Download the above program into the Snap Circuit Project 18 Light Intensity Logger. Press the F8 key to open the terminal window on your display. Press and hold the S2 switch until data starts dumping into the terminal window then release switch. The window should display data similar to the picture above and on the right.

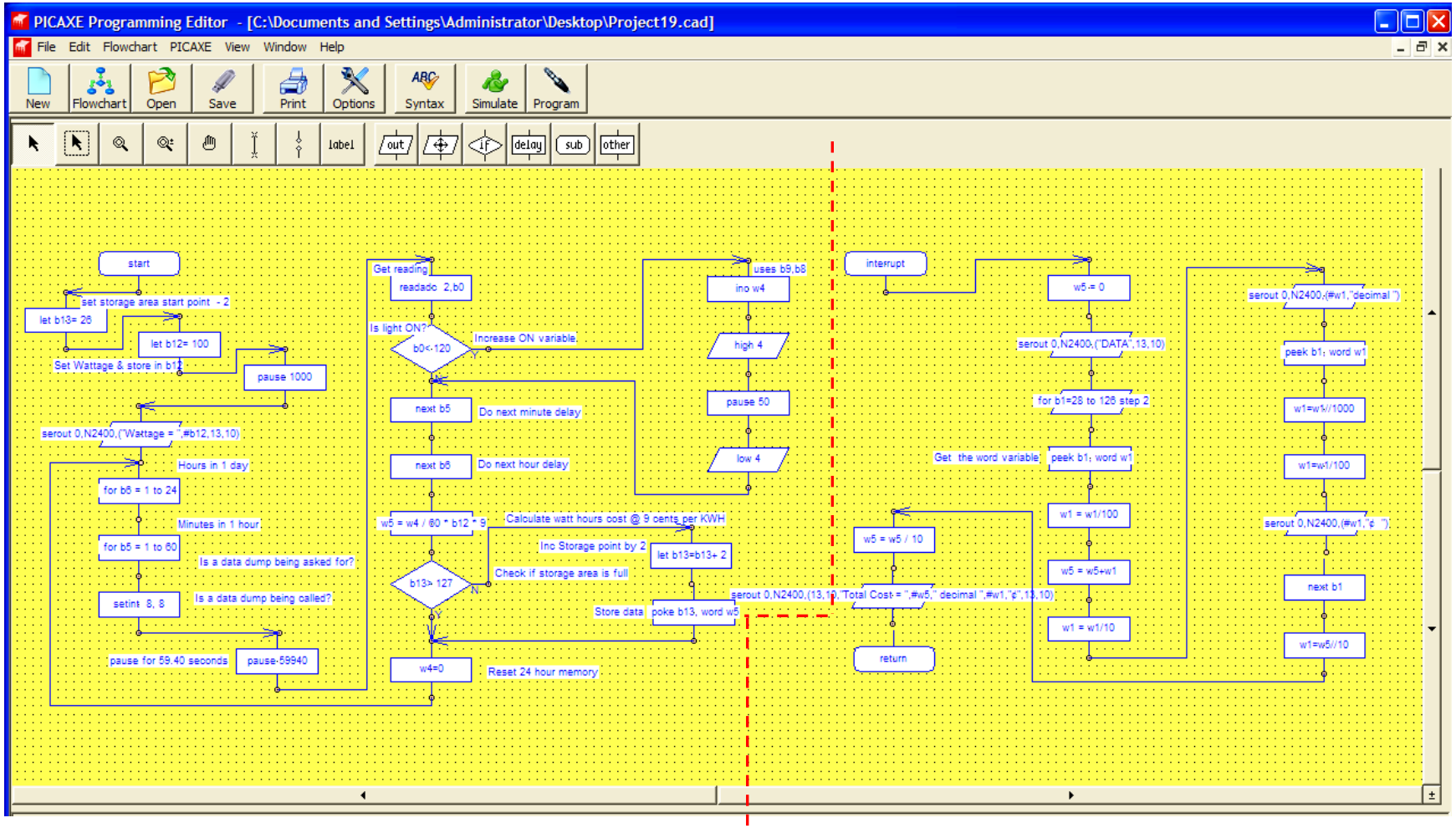
There will only be one reading since the light sensor data logger was just turned on. The rest of the data should be zero. It will take approximately one hour before the second reading is taken. Placing this circuit near a window for a few days will record the light levels for that area every hour for 99 hours. To make the readings faster, change the program as shown in the red boxes above and repeat the process. The next page shows program with notes for future reference.



By moving groups around some labels can be removed and program is easier to read. Notes are important to keep a clear picture of each lines purpose and changes like N2400 changed to N4800 five times in program.

Project 19, Green Power Meter or An Energy Cost Data Logger.

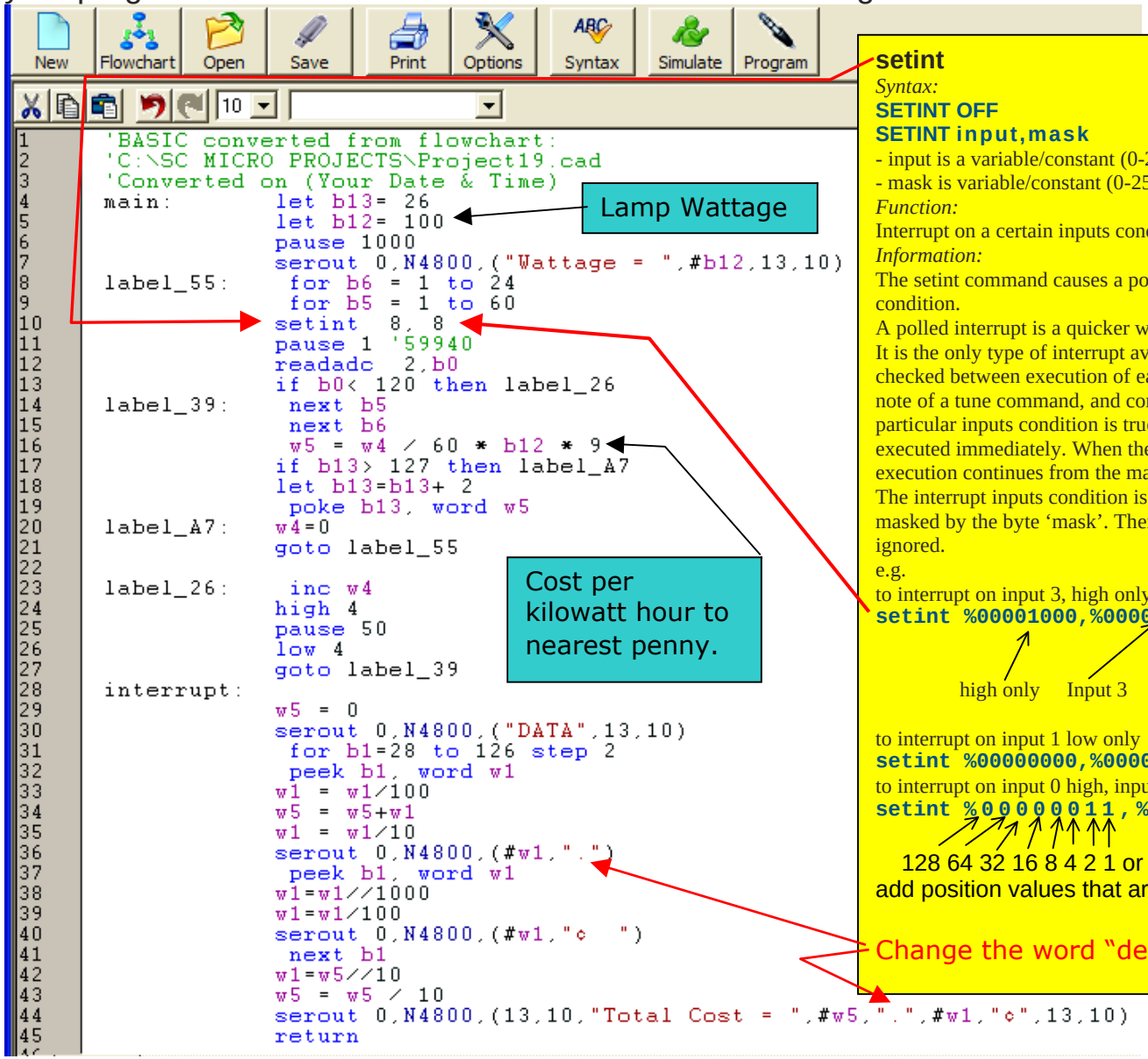
A data logger can be used to store data on how much electricity is being used by a device. For example, the light sensor is placed by a lamp in a room. When the lamp is turned on the micro-controller records the time. When the light is turned off, the time elapsed is calculated and stored. The time on is then used to calculate the kilowatt hours of energy used and the cost based on the current price of electricity. Total cost per day is then stored for displaying when requested. PICAXE® timing is approximate & can vary by ±1%. Consider the flow chart below.



This section gathers and stores data.

This section calculates and displays cost.

Since the flow chart editor will not allow a decimal, write the word “decimal” instead and convert it to a decimal point after transforming the flow chart to a basic program. The ‘¢’ symbol can be entered by holding down the <alt> key and entering 0162 on the number pad. Open the program editor and enter the flowchart as shown above. Be sure to save it before converting to basic. After converting to basic your program should be similar to the one shown here. Program in Basic for Green Power Meter.



setint

Syntax:

SETINT OFF

SETINT input,mask

- input is a variable/constant (0-255) which specifies input condition.
- mask is variable/constant (0-255) which specifies the mask

Function:

Interrupt on a certain inputs condition.

Information:

The setint command causes a polled interrupt on a certain input pin / flags condition.

A polled interrupt is a quicker way of reacting to a particular input combination. It is the only type of interrupt available in the PICAXE system. The inputs port is checked between execution of each command line in the program, between each note of a tune command, and continuously during any pause command. If the particular inputs condition is true, a ‘gosub’ to the interrupt sub-procedure is executed immediately. When the sub-procedure has been carried out, program execution continues from the main program.

The interrupt inputs condition is any pattern of ‘0’s and ‘1’s on the input port, masked by the byte ‘mask’. Therefore any bits masked by a ‘0’ in byte mask will be ignored.

e.g.

to interrupt on input 3, high only in binary format

setint %00001000,%00001000 (In decimal = 8,8)

high only Input 3 Input 2 Input 1 Input 0

to interrupt on input 1 low only

setint %00000000,%00000010 (In decimal = 0,2)

to interrupt on input 0 high, input 1 high and input 2 low

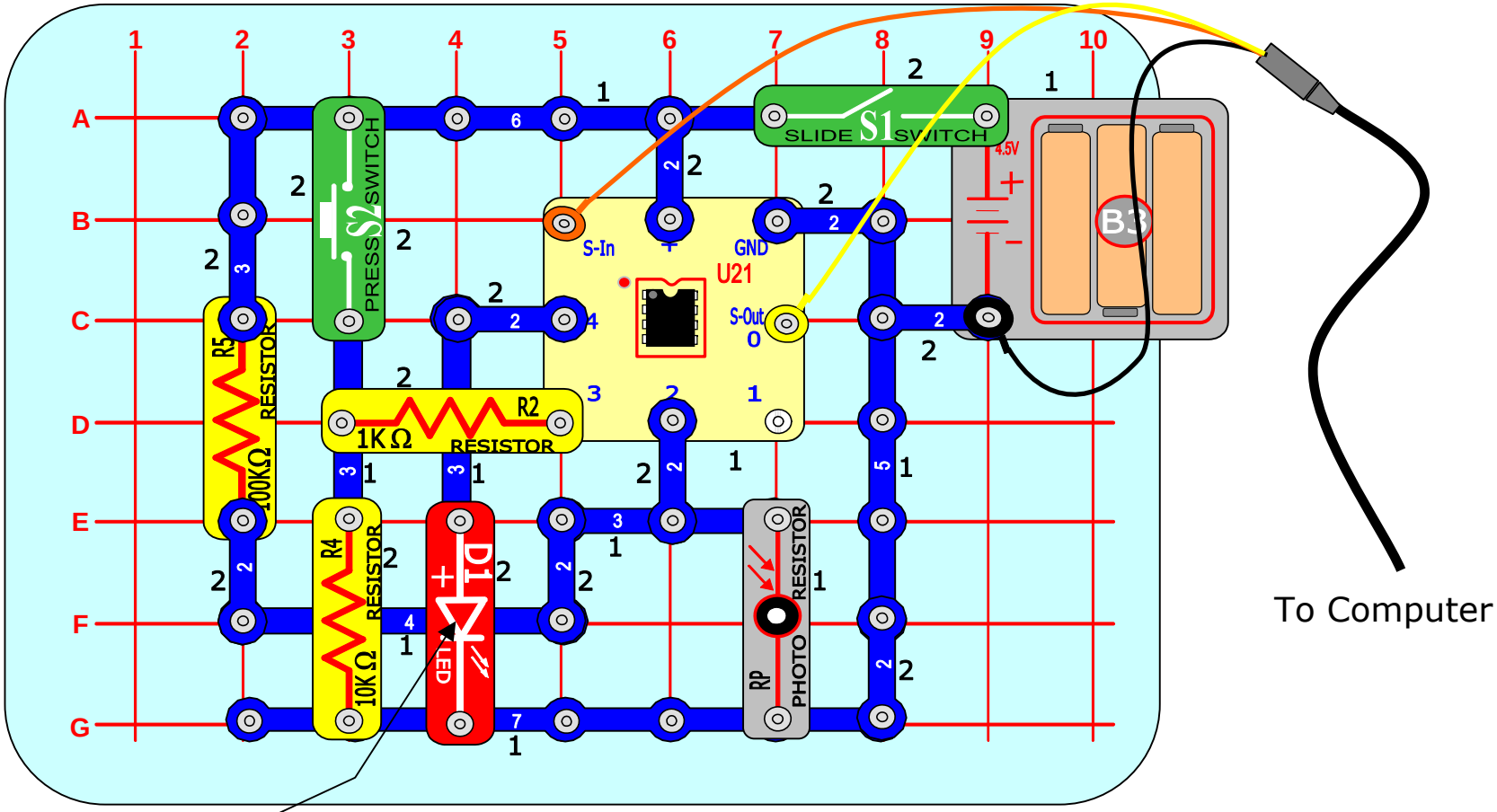
setint %00000011,%00000111 (In decimal = 3,7)

128 64 32 16 8 4 2 1 or 2+1=3 4+2+1=7

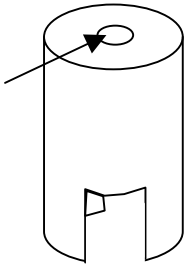
add position values that are ones to convert from binary to decimal.

Change the word “decimal” to a “.” before downloading.

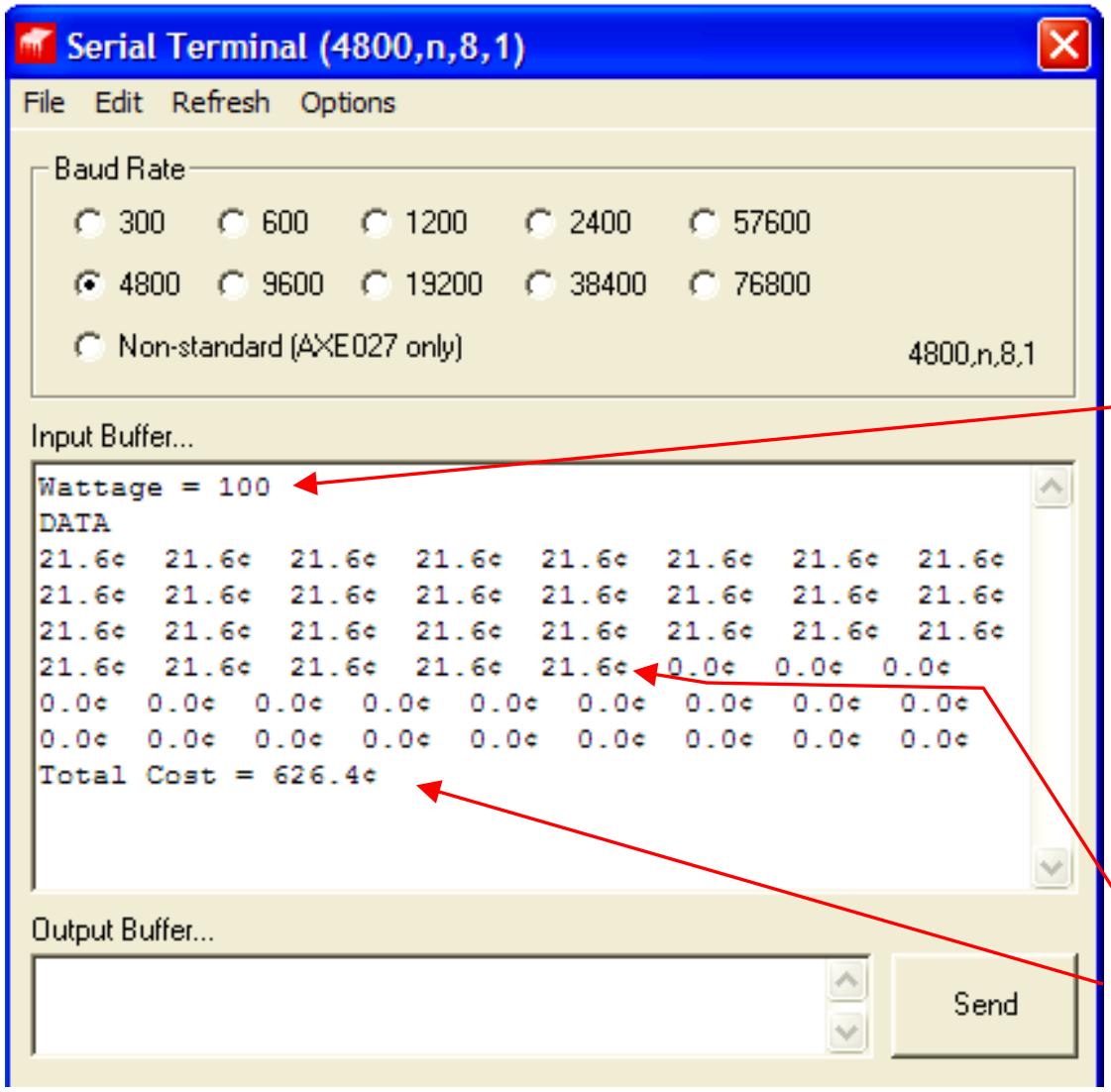
The program will take 24 hours before it records the first day's cost. To speed up program for testing purposes, change the second pause from 59940 to 1. After testing replace original value.
Build the Green Power Meter Circuit shown here.



To shield the photo resistor RP from daylight and other light sources you should take a tube from a roll of paper towels and cover one end with a piece of paper as shown here. Punch a pencil size hole in the paper and cut the other end of the tube so it fits over the photo resistor RP in the circuit above. Aim the hole in the tube at the lamp being measured so only light from that lamp hits the photo resistor.



To test the Green Power Meter (GPM) change the second pause in the program from 59940 to 1 and download into the Snap Circuit® on the previous page. After removing successful download window, quickly open the terminal by pressing <F8> key. Restart the GPM and your terminal window should display the 'Wattage = 100' message.



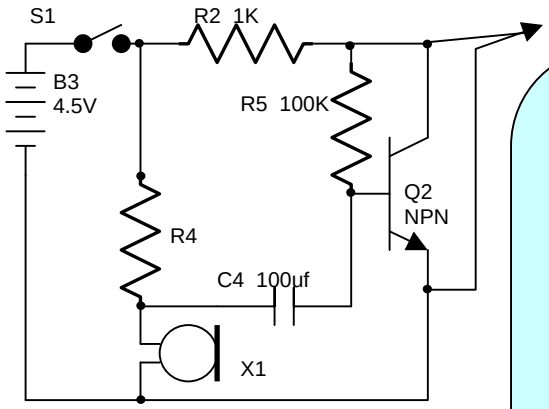
The red LED should be flashing very fast if enough light is present to trigger the record part of the program. Place your hand over the photo resistor and the flashing should stop. Press the S2 pushbutton to see recorded data. It will take more than one minute to emulate one day of recording with light on.

29 Days recorded, out of 50 possible before memory is full. For 29 Days Recorded the lamp at 9 cents per kilowatt hour, would have cost the user $6.26.

Be sure to change the wattage setting to the value of the lamp you will test and the price per kilowatt hour to the nearest penny rate on your electric bill. Replace original pause settings and download the program. Remove computer leads and install 10K resistor. Place it under the light to be tested. Make sure the LED flashes once every minute the lamp is on, and does not flash when lamp is off. Wait a month to get a good reading on the cost of normal use of the lamp. Although the GPM can only measure up to 255 watts directly, any device tied to a light source can be calculated. For example, a 1200 watt heater with a light would cost 12 times the number calculated above or 12 x $6.26 = $75.12 for 29 days.

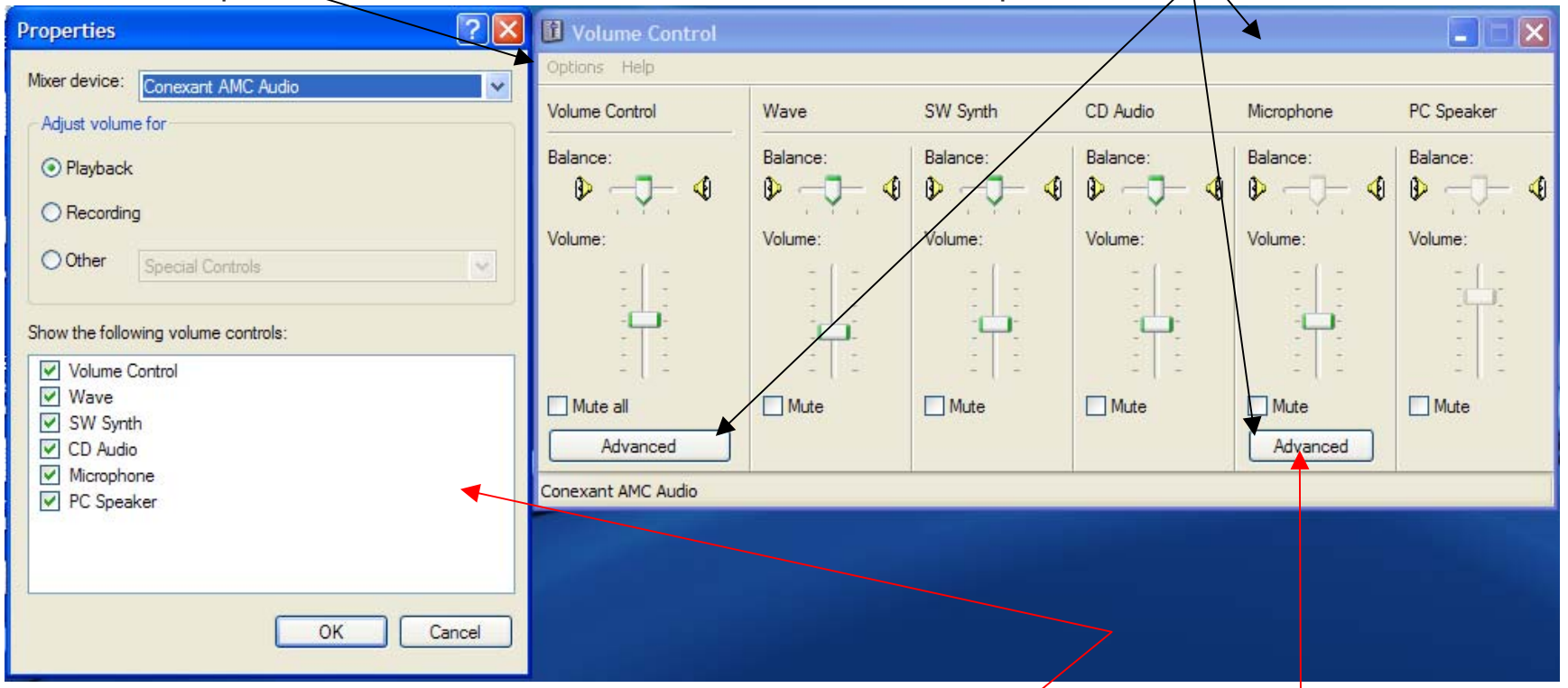
PROJECT 20, Audio Amplifier and the Microphone (X1)

Build the Snap Circuit shown below. For those familiar with electronic circuits and schematics a schematic drawing of this circuit is also included.



Schematic Drawing

Make sure the computer speakers are turned on and the volume or loudness control is not at zero. Also make sure the microphone input is on by checking the controls as shown below.
Go to control panel and double click the “Sounds and Audio Devices” icon. In the panel that opens, click the “Advanced” bar under the device volume section to open the window below. {May appear different}
Click on the “options” menu and check the “Advanced” section to open these.



Click on “options” again, and then “properties” to open this window.
Make sure the Microphone & Volume Control boxes are checked. Click OK.

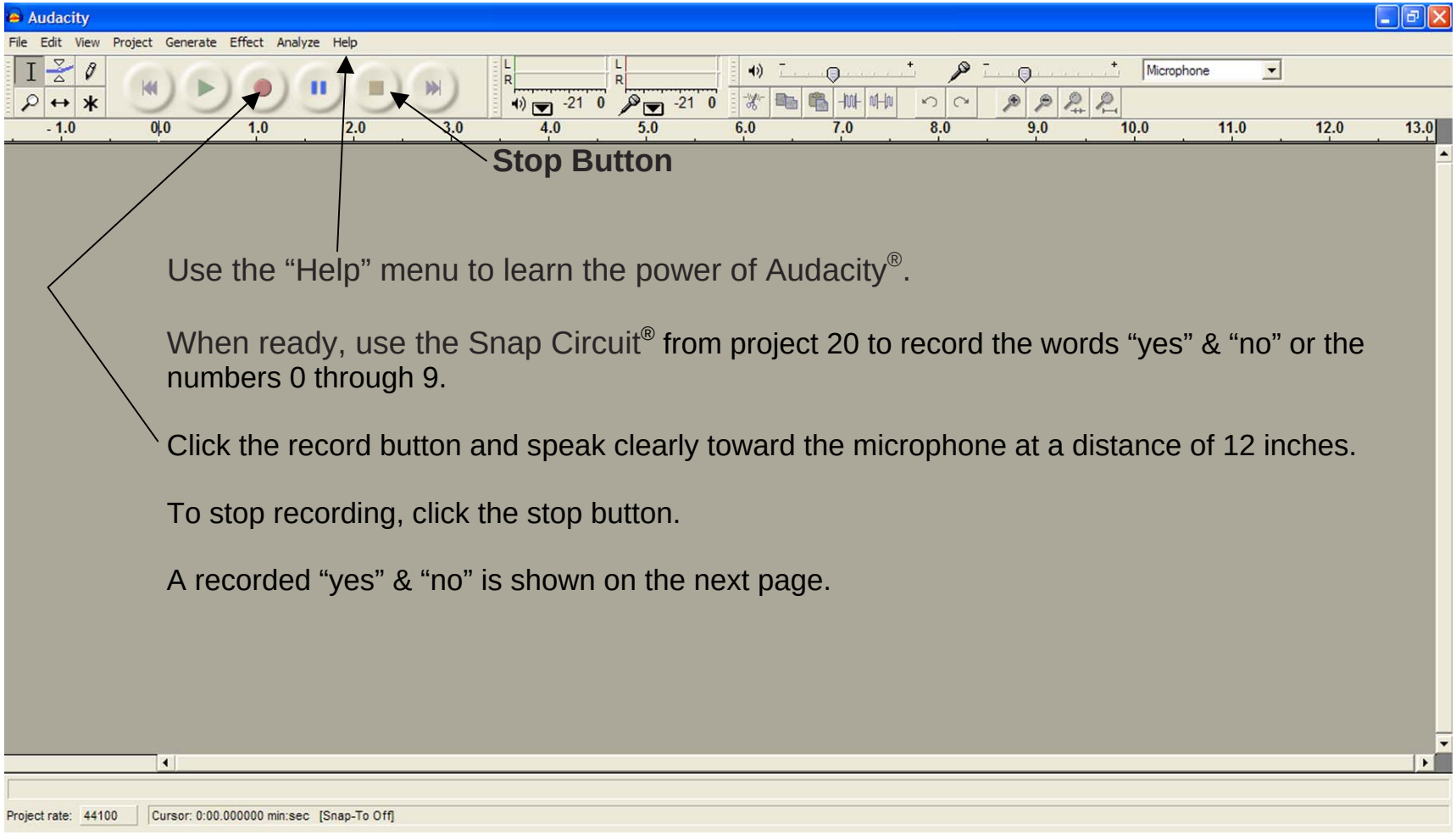
For best sensitivity, click on the advanced button under the microphone column and make sure the “Microphone Boost” box is checked in the window that opens. Not all versions of windows will have these “Advanced” buttons and windows may appear different. Turn S1 switch to on and you should be able to hear amplified sounds from the microphone. Test by blowing on the microphone. If feedback occurs, reduce the speaker volume or the microphone input setting. Keep this circuit for the next project.

SECTION 4: AUDACITY® & SOUND

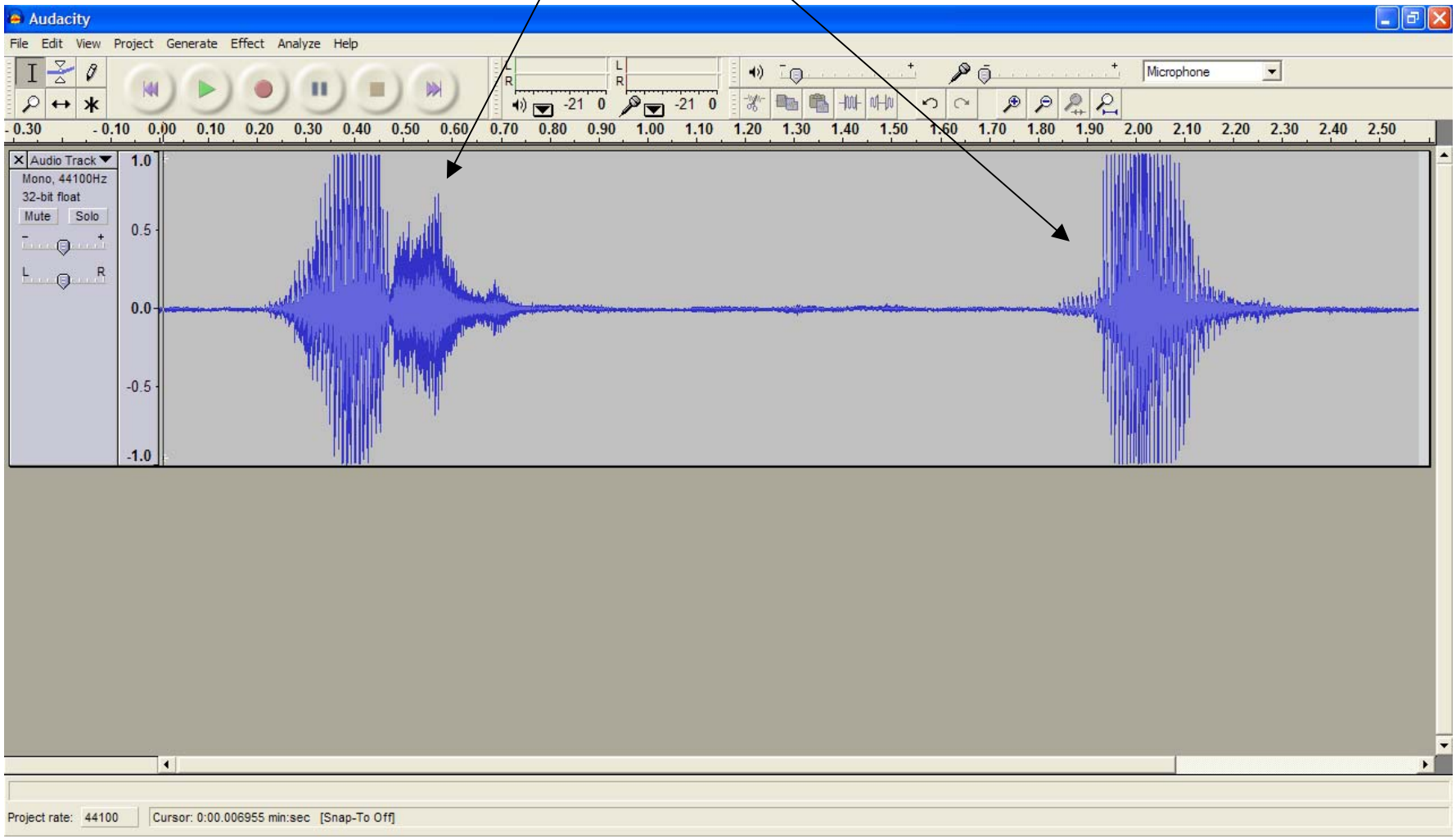
Project 21, Audacity®

audacity-win-1.2.6.exe

Run the installer program “audacity®-win-1.2.6.exe” from the Elenco® disc.
Follow the instructions to install the program. After installation run the program and a window similar to the one below should open;



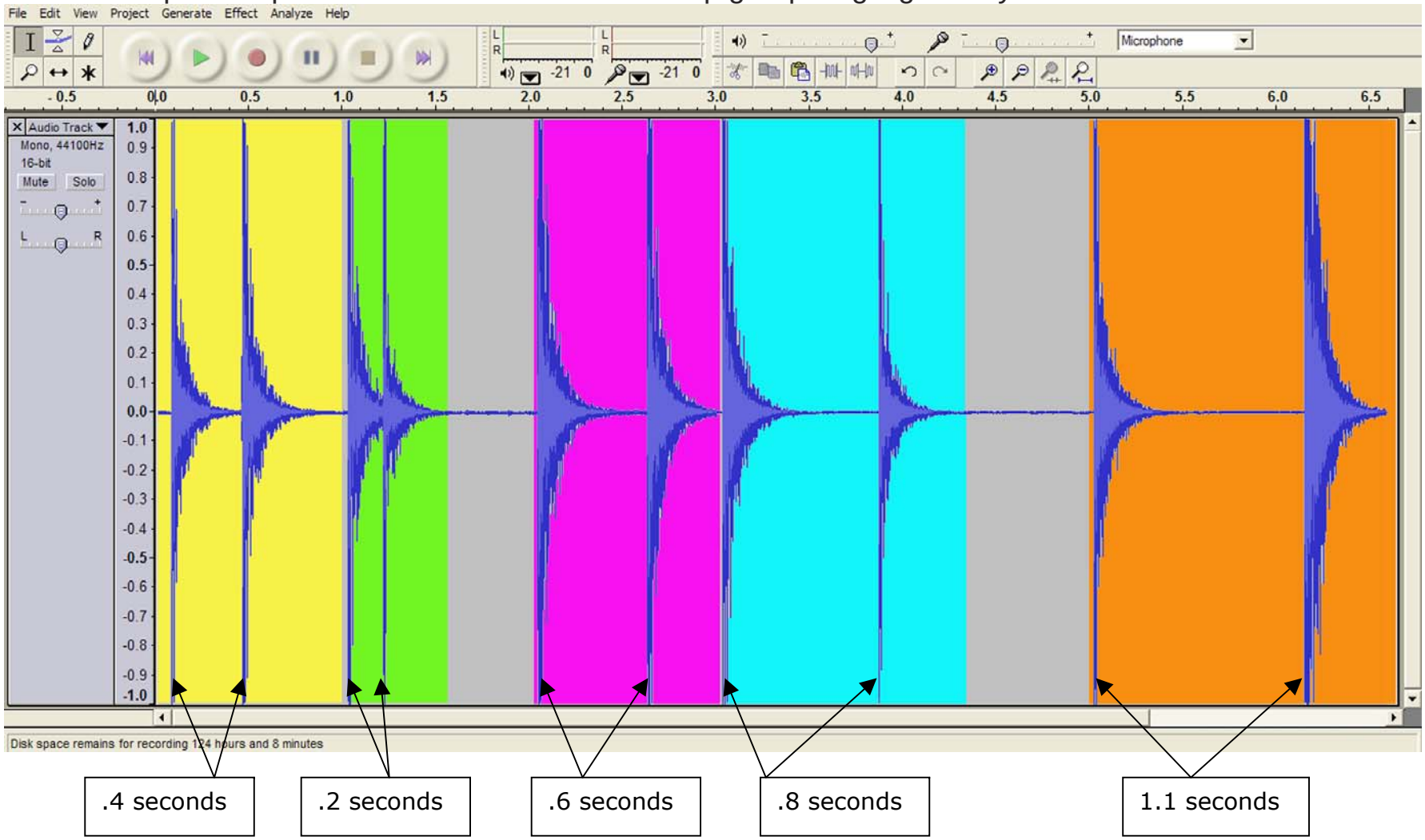
An Audacity® recording of the words “yes” and “no”.



In speech recognition programs, computers analyze the digital data from words and use the common points to determine the word being spoken. The next project will use the sound of clapping to control an LED.

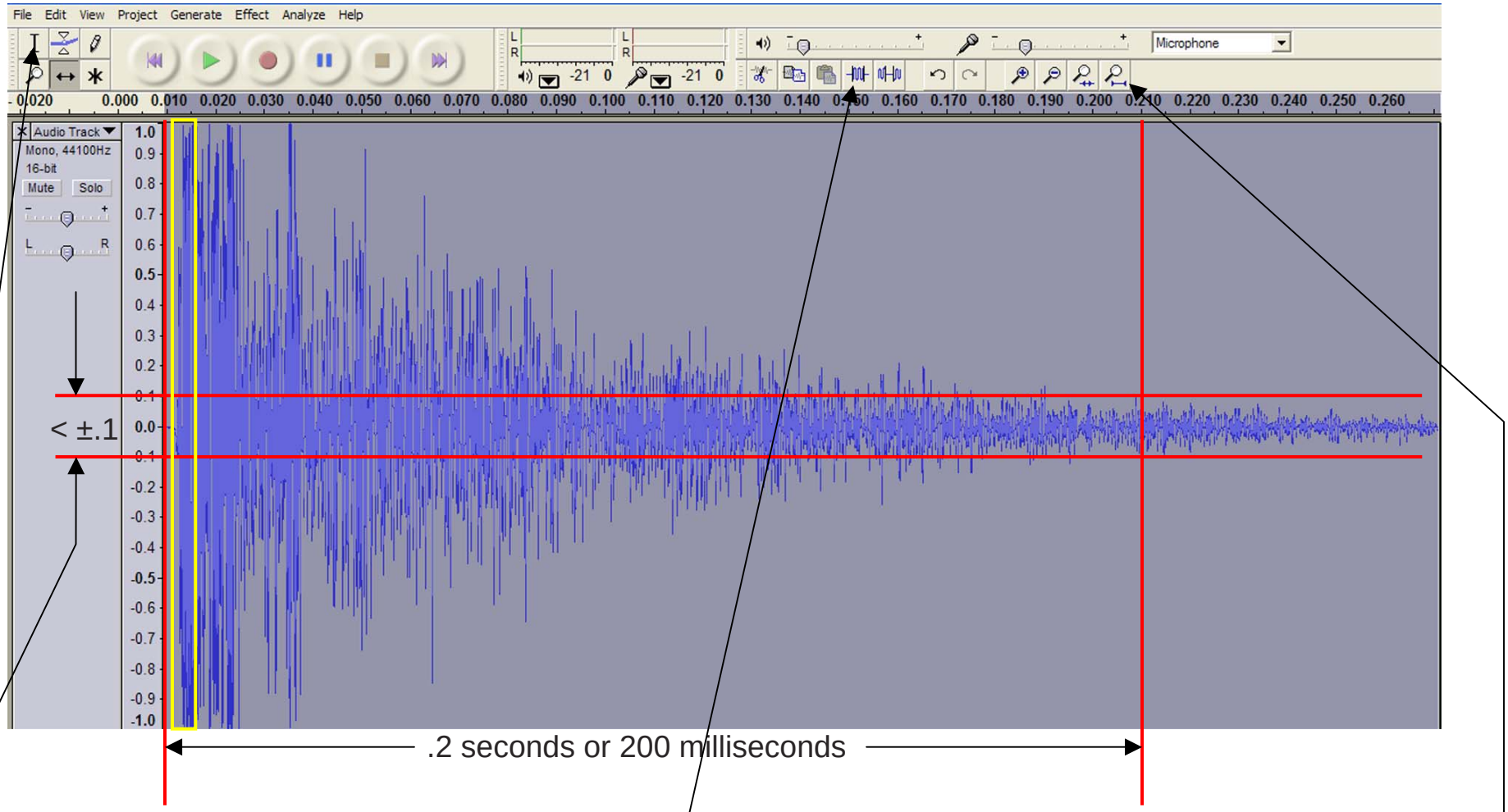
Project 22, Investigating Sound of Clapping.

Use previous setup to record the sound of two claps. Try recording two claps with different delays between claps. The picture below shows different clap groups highlighted by different colors.



Peaks reach both +1 and –1 levels, then decay to a level less than .1 within .2 seconds.

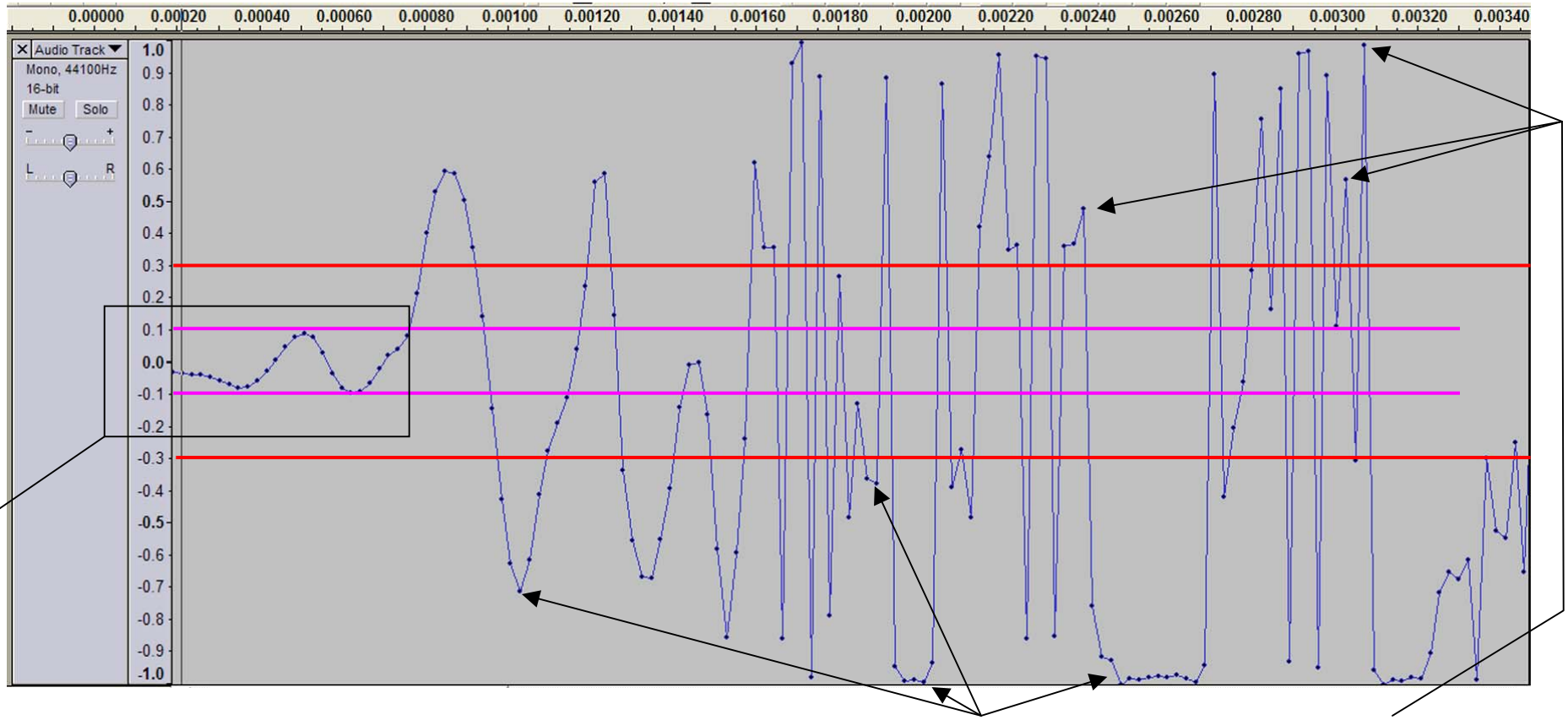
A single clap from the previous page is shown here.



Amplitude window for less than 10% of a positive or a negative peak.
Expanding the area inside the yellow box will show the first few milliseconds of clap. This area could be used to trigger an interrupt and start the process to analyze the sound. This area is expanded by;

1. Highlight area using this button.
2. Eliminate rest of curve using this button.
3. Expand using this button.

Expanded start of a clap. Data points highlighted by clicking “pencil” button.

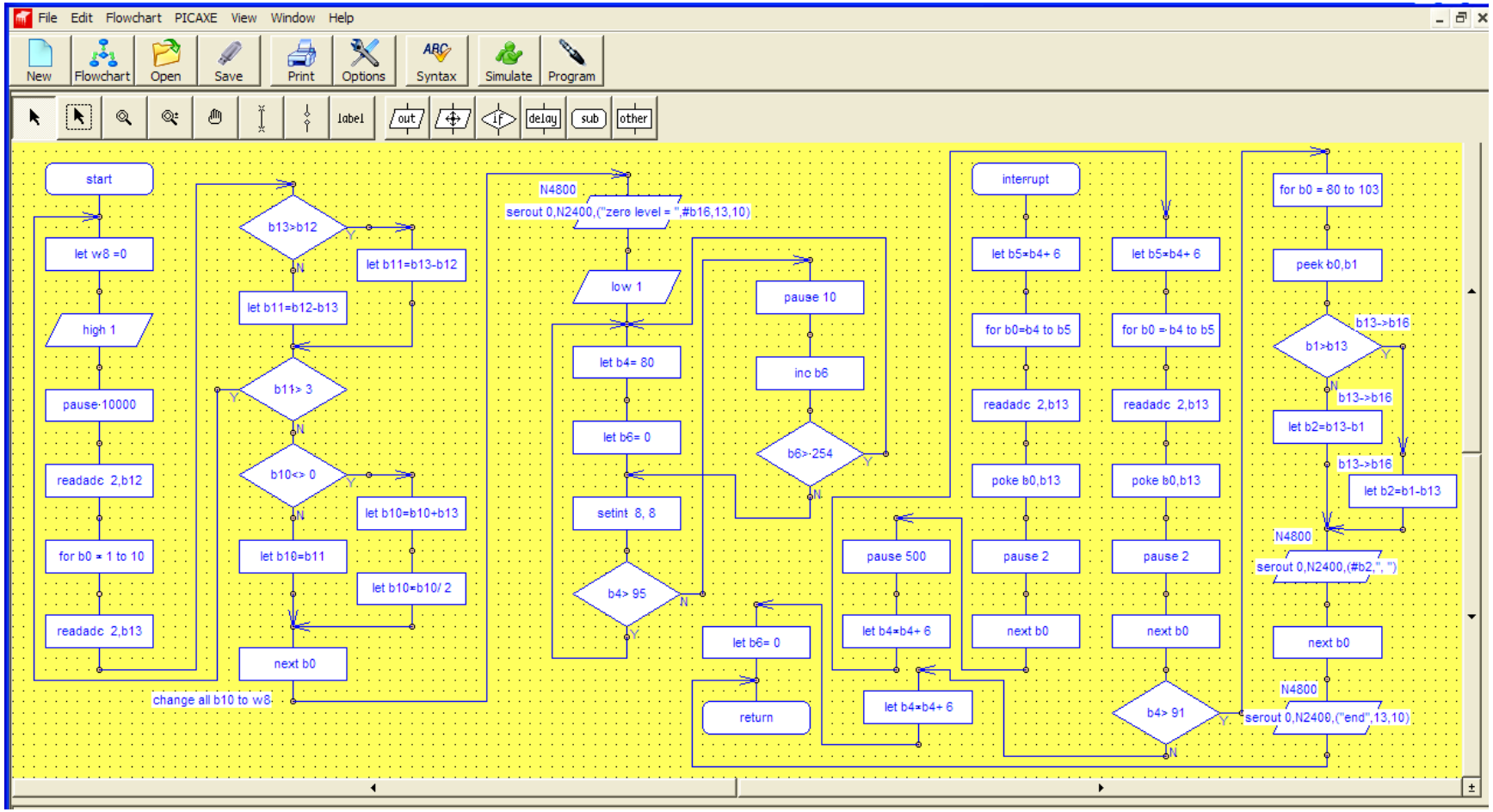


If –1 = 0, +1 = 255, and 0.0 = 127 then the dots below the –0.3 red line and the dots above the 0.3 red line show digital data points for loud sharp sounds that are less than 88 or greater than 166. This data can be used to detect the start of a clap sound.

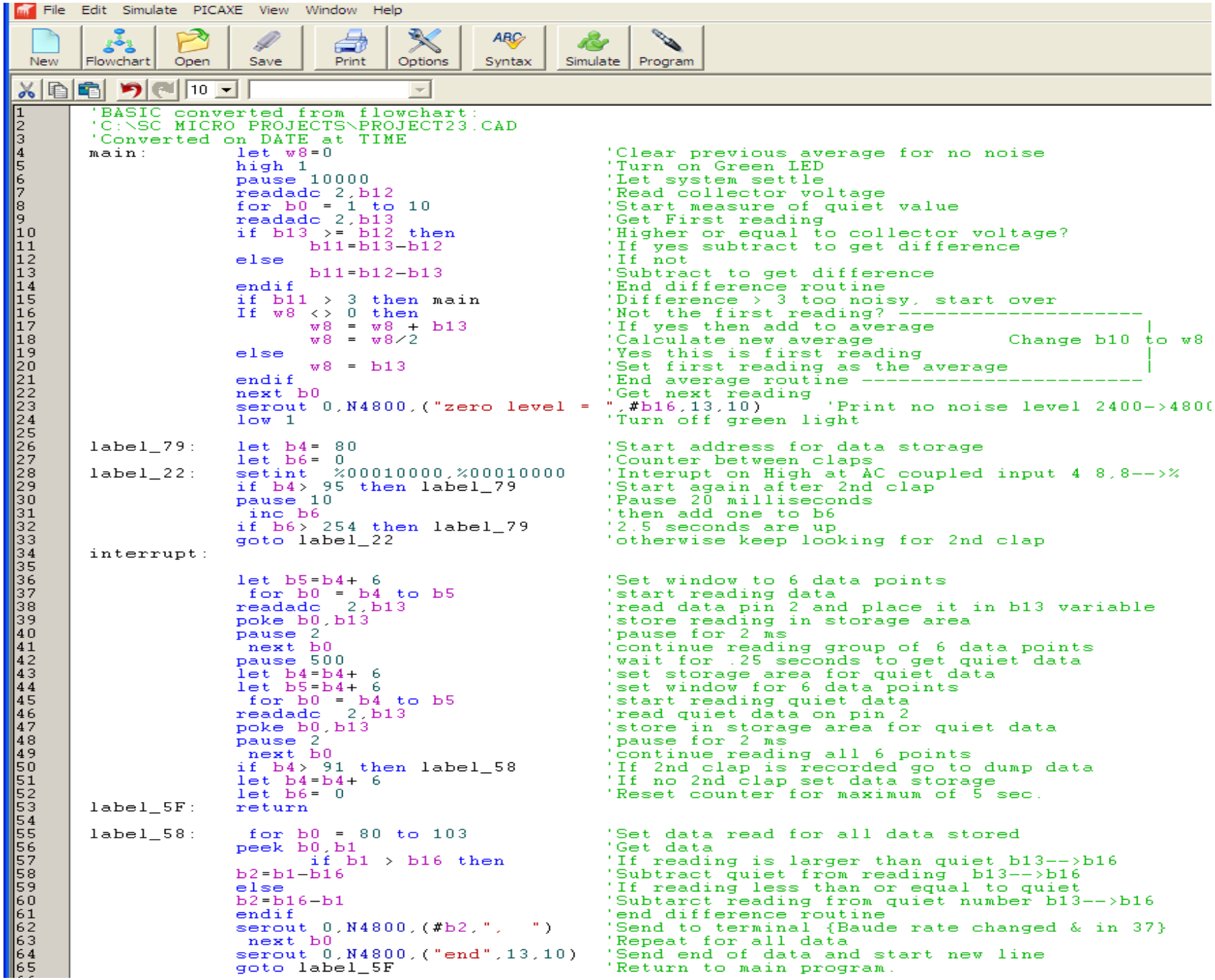
After .25 seconds or 250 milliseconds all the data should be between the two center purple lines as shown by the dots at the beginning of the curve. This data can be used to detect the end of a clap sound. Use the Audacity® program and the circuit from project 20 to verify these facts.

Project 23, The Clap-Data Program

Consider the flow chart shown here as one method of recording and displaying data from the sound of two claps within a 2.5 second window.

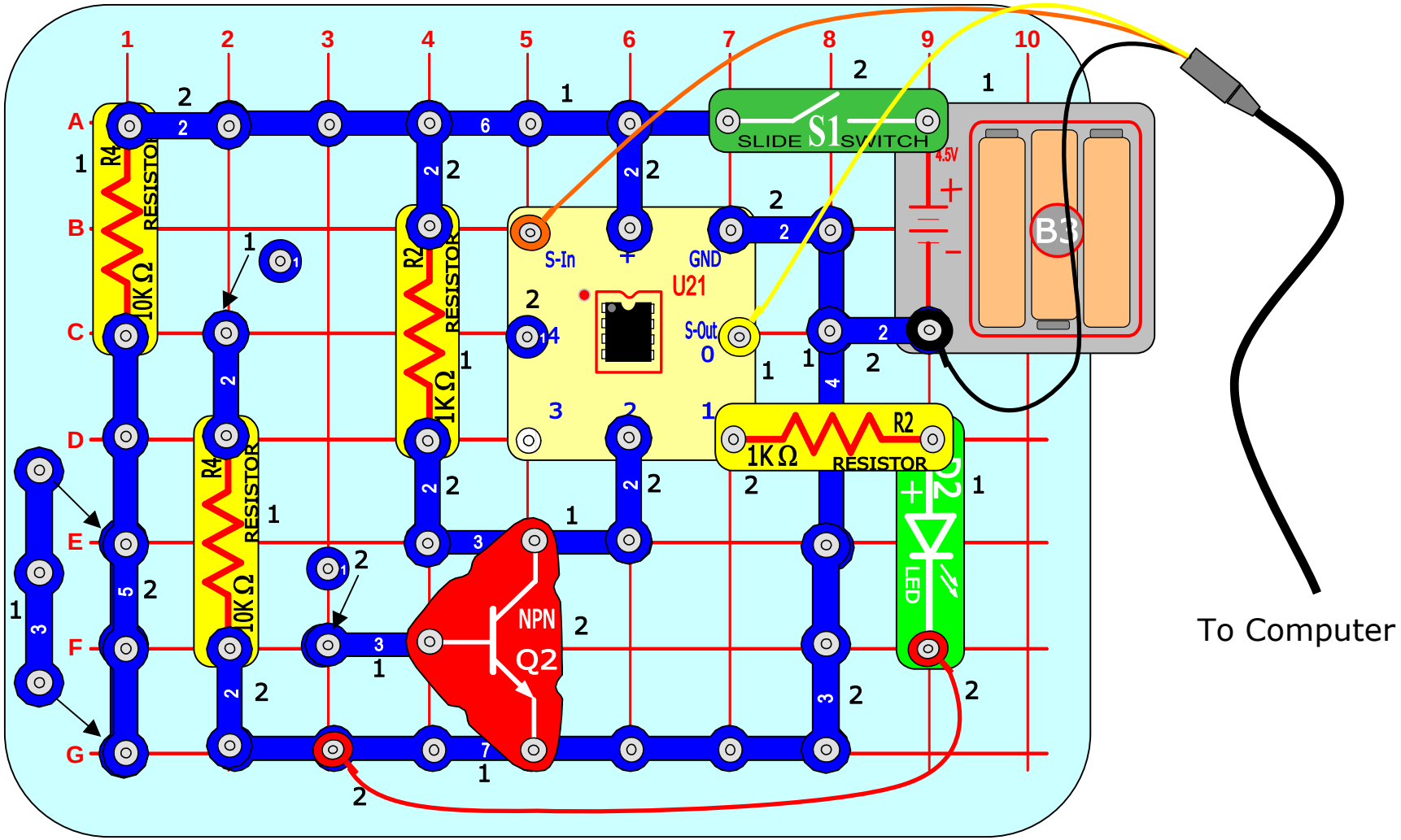


Converting the above flowchart to a basic program and adding notes can produce the Basic program shown on the next page.

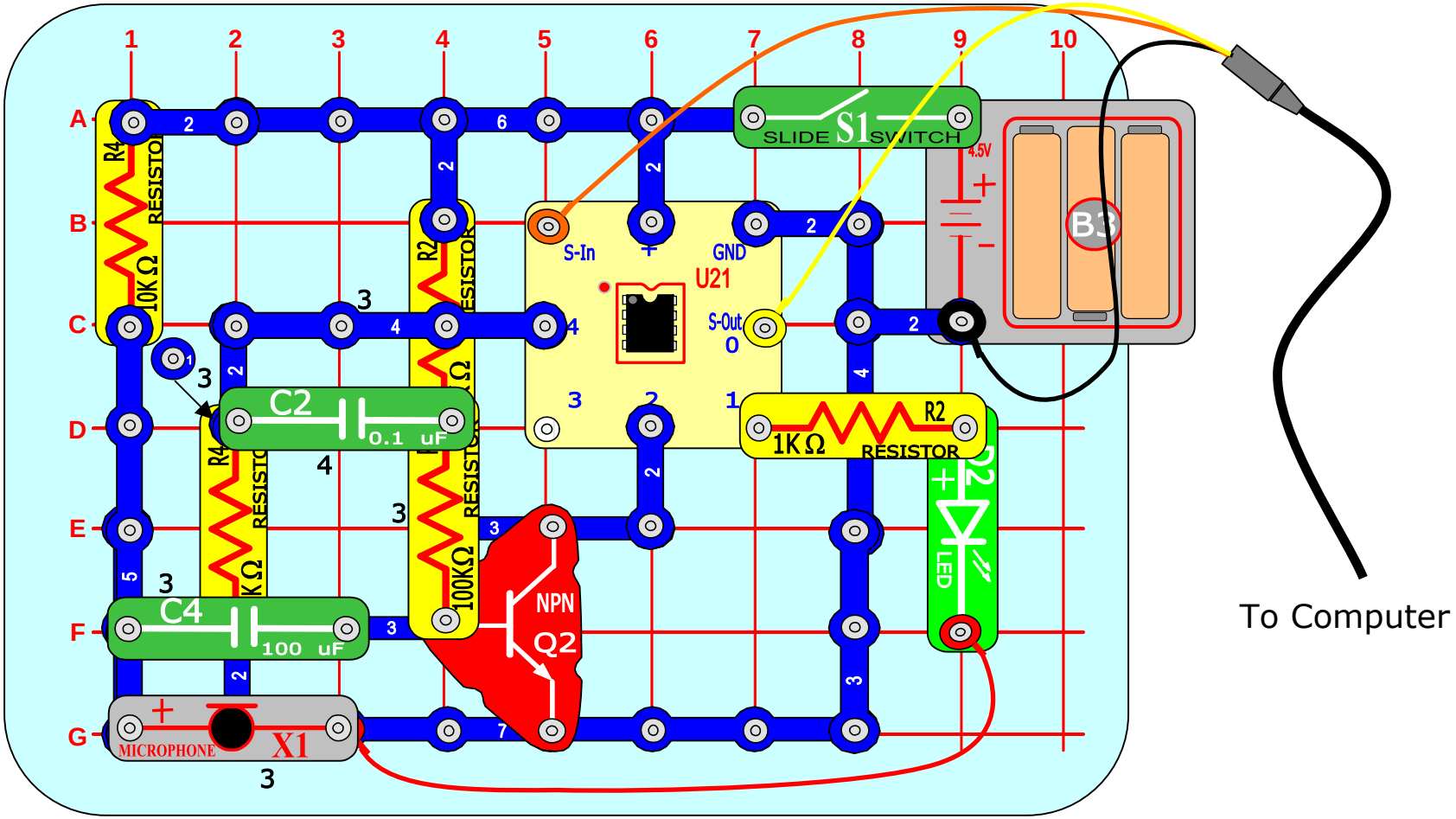

File Edit Simulate PICAXE View Window Help
New Flowchart Open Save Print Options Syntax Simulate Program
10
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\PROJECT23.CAD
'Converted on DATE at TIME
main: let w8=0 'Clear previous average for no noise
high 1 'Turn on Green LED
pause 10000 'Let system settle
readadc 2,b12 'Read collector voltage
for b0 = 1 to 10 'Start measure of quiet value
readadc 2,b13 'Get First reading
if b13 >= b12 then 'Higher or equal to collector voltage?
b11=b13-b12 'If yes subtract to get difference
else 'If not
b11=b12-b13 'Subtract to get difference
endif 'End difference routine
if b11 > 3 then main 'Difference > 3 too noisy, start over
If w8 <> 0 then 'Not the first reading?
w8 = w8 + b13 'If yes then add to average
w8 = w8/2 'Calculate new average Change b10 to w8
else 'Yes this is first reading
w8 = b13 'Set first reading as the average
endif 'End average routine
next b0 'Get next reading
serout 0,N4800,("zero level = ",#b16,13,10) 'Print no noise level 2400->4800
low 1 'Turn off green light
label_79: let b4= 80 'Start address for data storage
let b6= 0 'Counter between claps
label_22: setint %00010000,%00010000 'Interupt on High at AC coupled input 4 8,8-->%
if b4> 95 then label_79 'Start again after 2nd clap
pause 10 'Pause 20 milliseconds
inc b6 'then add one to b6
if b6> 254 then label_79 '2.5 seconds are up
goto label_22 'otherwise keep looking for 2nd clap
interrupt:
let b5=b4+ 6 'Set window to 6 data points
for b0 = b4 to b5 'start reading data
readadc 2,b13 'read data pin 2 and place it in b13 variable
poke b0,b13 'store reading in storage area
pause 2 'pause for 2 ms
next b0 'continue reading group of 6 data points
pause 500 'wait for .25 seconds to get quiet data
let b4=b4+ 6 'set storage area for quiet data
let b5=b4+ 6 'set window for 6 data points
for b0 = b4 to b5 'start reading quiet data
readadc 2,b13 'read quiet data on pin 2
poke b0,b13 'store in storage area for quiet data
pause 2 'pause for 2 ms
next b0 'continue reading all 6 points
if b4> 91 then label_58 'If 2nd clap is recorded go to dump data
let b4=b4+ 6 'If no 2nd clap set data storage
let b6= 0 'Reset counter for maximum of 5 sec.
label_5F: return
label_58: for b0 = 80 to 103 'Set data read for all data stored
peek b0,b1 'Get data
if b1 > b16 then 'If reading is larger than quiet b13-->b16
b2=b1-b16 'Subtract quiet from reading b13-->b16
else 'If reading less than or equal to quiet
b2=b16-b1 'Subtarct reading from quiet number b13-->b16
endif 'end difference routine
serout 0,N4800,(#b2,", ") 'Send to terminal {Baude rate changed & in 37}
next b0 'Repeat for all data
serout 0,N4800,("end",13,10) 'Send end of data and start new line
goto label_5F 'Return to main program.

Build the Snap Circuit® shown on the next two pages and download the above program into the micro-processor.

Clap data taking circuit levels 1 & 2.



Finish the Snap Circuit® shown below for the clap data taking circuit levels 3 & 4.



After downloading the program, clear the download window and open the Terminal window under the PICAXE® menu in the program editor or press F8. A quiet level will be calculated and displayed when the green LED goes off. Every two claps should produce a stream of data that is followed by the word “end”. Second clap should be .5 to 2.5 seconds after the first clap for best results. Data will vary from circuit to circuit, but certain characteristics will remain the same. It is these characteristics that will be used to control our final output.

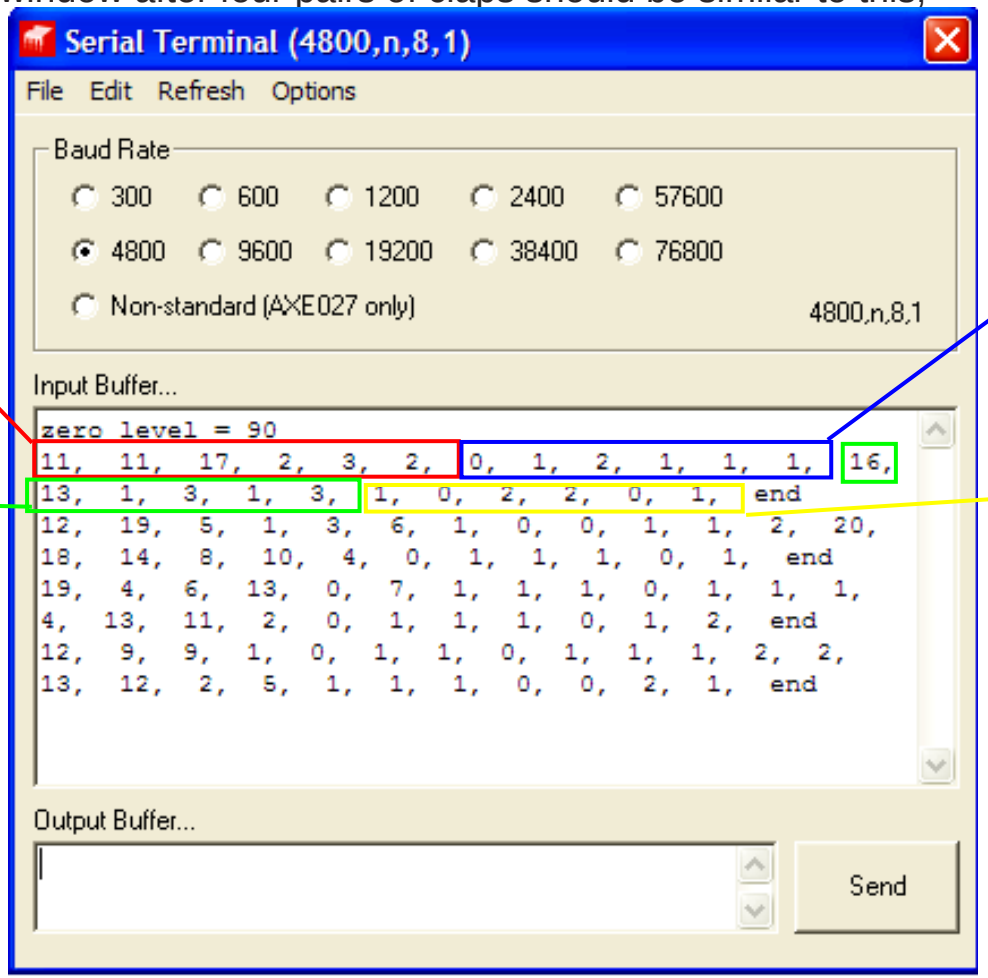
Project 24, Analyzing Clap Data

Data shown in terminal window after four pairs of claps should be similar to this;

Start of first clap
11
11
17
2
3
2
46/ 6 = 7

Start of 2nd clap
16
13
1
3
1
3
37 / 6 = 6

Adding the levels for the start of the first clap yields 7 times the sum of the data for the end of that clap. It is this ratio we will use to recognize the clapping sound.
Remainders are dropped,



End of first clap
0
1
2
1
1
1
6 / 6 = 1

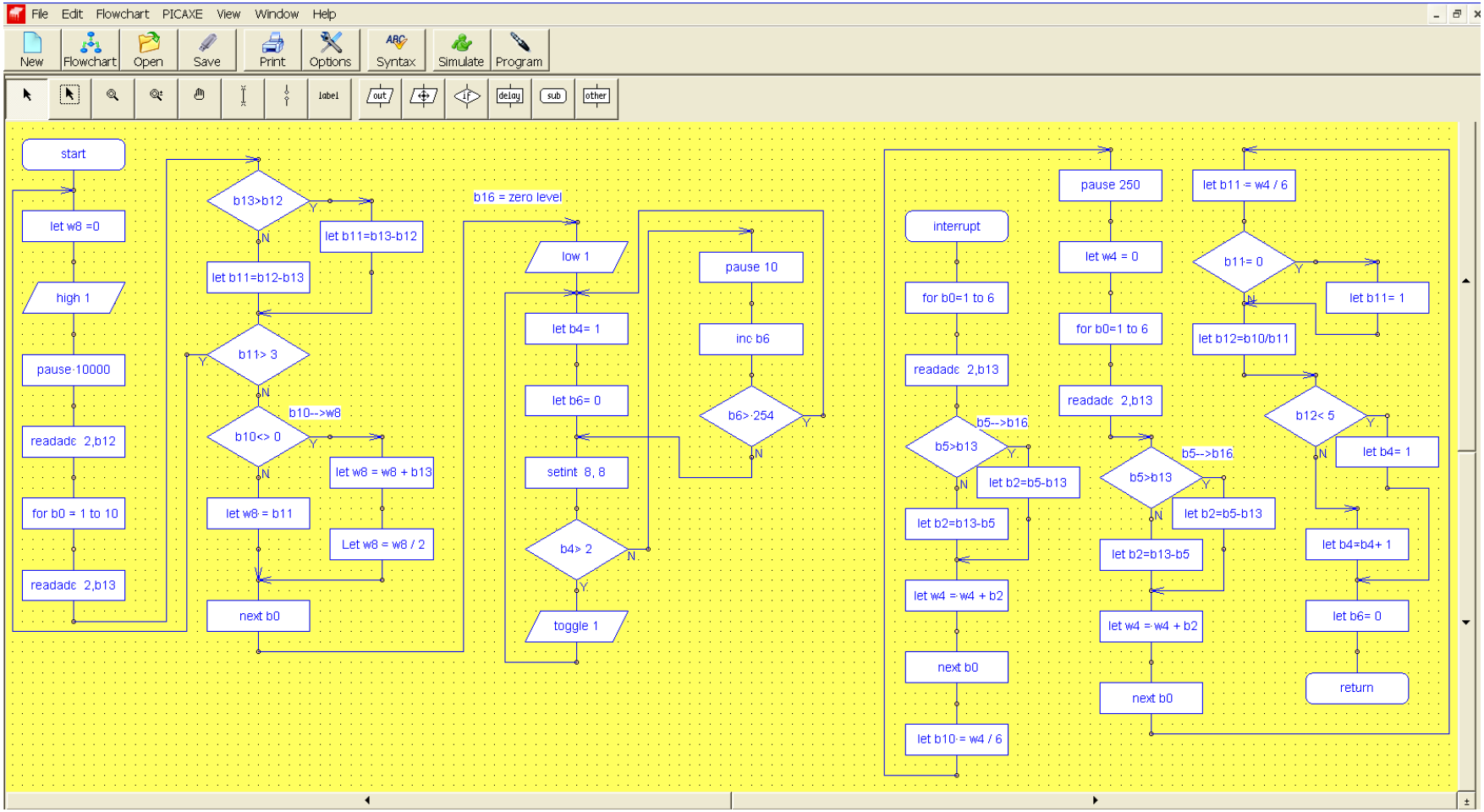
End of 2nd clap
1
0
2
2
0
1
6 / 6 = 1

Adding the levels for the start of the second clap yields 6 times the sum of the data for the end of that clap.
Remainders are dropped, only whole numbers

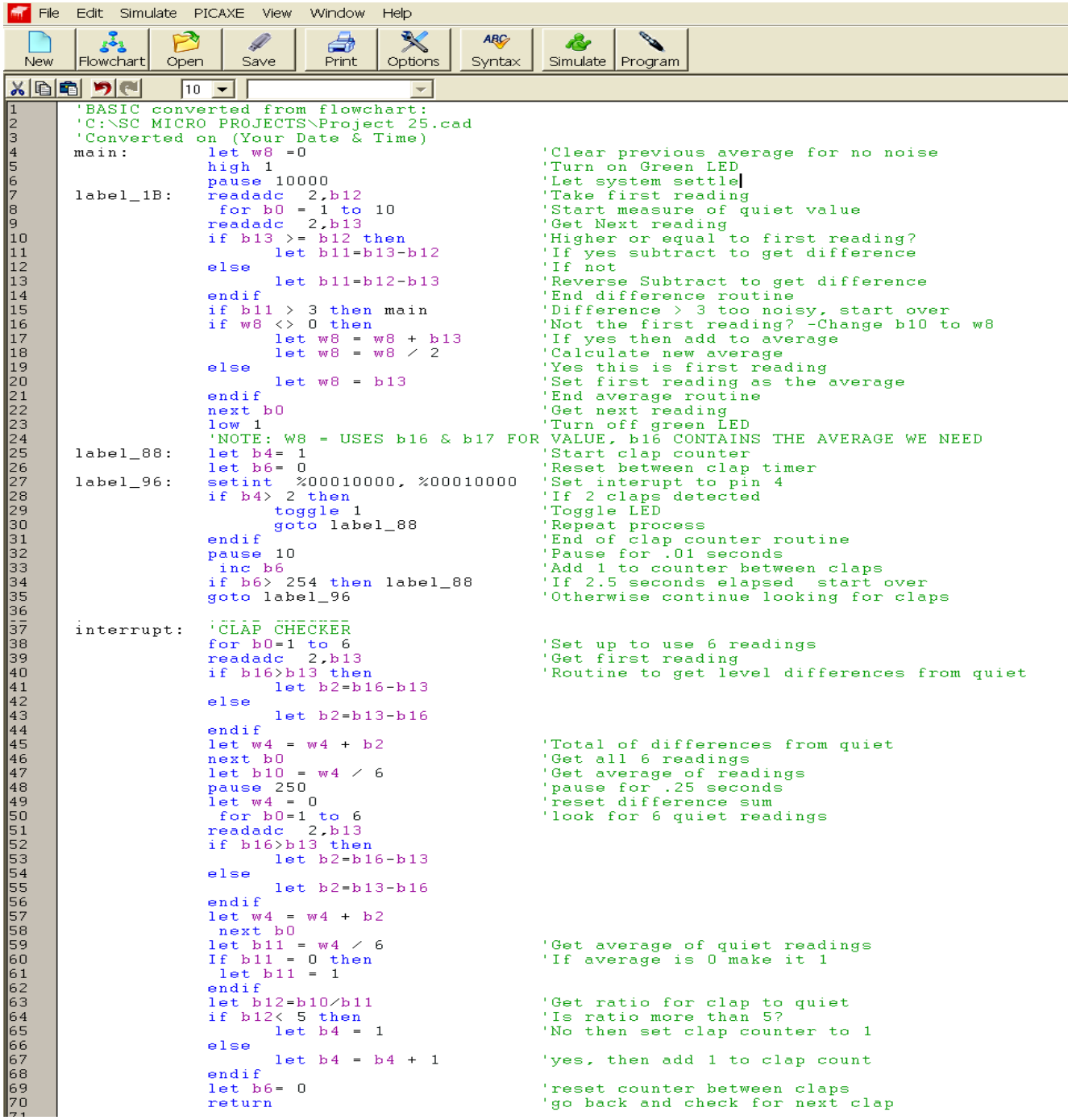
Clap Pair 1 ratios: 7 for first & 6 for second
Clap Pair 2 ratios: 7 for first & 12 for second {If average is less than 1, 1 is used}
Clap Pair 3 ratios: 8 for first & 5 for second
Clap Pair 4 ratios: 5 for first & 5 for second
Observing ratios lets us pick 5 as the minimum ratio for a clap sound.

Project 25, The Clap it ON, Clap it OFF Circuit

In the following flow chart the ratio of 5 is used to determine a clap sound and if two claps occur within a 2.5 second period the green LED changes state. Since there is no need to store the data and send it to the computer, the program can be a stand alone circuit that responds to a two clap command.



The beginning of the above flow chart the 10-second delay is necessary in order to allow all the circuit transients to settle and come to their operating levels. A green light indicates this process is running.




```
'BASIC converted from flowchart:
'C:\SC MICRO PROJECTS\Project 25.cad
'Converted on (Your Date & Time)
main:       let w8 =0                       'Clear previous average for no noise
            high 1                          'Turn on Green LED
            pause 10000                     'Let system settle
label_1B:   readadc  2,b12                  'Take first reading
             for b0 = 1 to 10               'Start measure of quiet value
            readadc  2,b13                  'Get Next reading
            if b13 >= b12 then              'Higher or equal to first reading?
                  let b11=b13-b12           'If yes subtract to get difference
            else                            'If not
                  let b11=b12-b13           'Reverse Subtract to get difference
            endif                           'End difference routine
            if b11 > 3 then main            'Difference > 3 too noisy, start over
            if w8 <> 0 then                 'Not the first reading? -Change b10 to w8
                  let w8 = w8 + b13         'If yes then add to average
                  let w8 = w8 / 2           'Calculate new average
            else                            'Yes this is first reading
                  let w8 = b13              'Set first reading as the average
            endif                           'End average routine
            next b0                         'Get next reading
            low 1                           'Turn off green LED
            'NOTE: W8 = USES b16 & b17 FOR VALUE, b16 CONTAINS THE AVERAGE WE NEED
label_88:   let b4= 1                       'Start clap counter
            let b6= 0                       'Reset between clap timer
label_96:   setint  %00010000, %00010000    'Set interupt to pin 4
            if b4> 2 then                   'If 2 claps detected
                  toggle 1                  'Toggle LED
                  goto label_88             'Repeat process
            endif                           'End of clap counter routine
            pause 10                        'Pause for .01 seconds
             inc b6                         'Add 1 to counter between claps
            if b6> 254 then label_88        'If 2.5 seconds elapsed  start over
            goto label_96                   'Otherwise continue looking for claps

interrupt:  'CLAP CHECKER
            for b0=1 to 6                   'Set up to use 6 readings
            readadc  2,b13                  'Get first reading
            if b16>b13 then                 'Routine to get level differences from quiet
                  let b2=b16-b13
            else
                  let b2=b13-b16
            endif
            let w4 = w4 + b2                'Total of differences from quiet
            next b0                         'Get all 6 readings
            let b10 = w4 / 6                'Get average of readings
            pause 250                       'pause for .25 seconds
            let w4 = 0                      'reset difference sum
             for b0=1 to 6                  'look for 6 quiet readings
            readadc  2,b13
            if b16>b13 then
                  let b2=b16-b13
            else
                  let b2=b13-b16
            endif
            let w4 = w4 + b2
             next b0
            let b11 = w4 / 6                'Get average of quiet readings
            If b11 = 0 then                 'If average is 0 make it 1
             let b11 = 1
            endif
            let b12=b10/b11                 'Get ratio for clap to quiet
            if b12< 5 then                  'Is ratio more than 5?
                  let b4 = 1                'No then set clap counter to 1
            else
                  let b4 = b4 + 1           'yes, then add 1 to clap count
            endif
            let b6= 0                       'reset counter between claps
            return                          'go back and check for next clap
```

When the light goes out the circuit is reading to use. A proper two-clap sound should toggle the green LED on or off. After conversion to Basic and cleanup the program should look like the one shown here.

Notes added will help when viewed at a later date.

All flow charts and basic programs are on disc provided.

Elenco® Electronics Inc.
150 Carpenter Ave
Wheeling, IL 60090
(847) 541-3800
www.elenco.com