



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com


Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



StampWorks

Experiments and BASIC Stamp Source Code

Version 2.1

PARALLAX 

WARRANTY

Parallax Inc. warrants its products against defects in materials and workmanship for a period of 90 days from receipt of product. If you discover a defect, Parallax Inc. will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to Parallax, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to Parallax. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem. Parallax will return your product or its replacement using the same shipping method used to ship the product to Parallax.

14-DAY MONEY BACK GUARANTEE

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax Inc. will refund the purchase price of the product, excluding shipping/handling costs. This guarantee is void if the product has been altered or damaged. See the Warranty section above for instructions on returning a product to Parallax.

COPYRIGHTS AND TRADEMARKS

This documentation is copyright 2005 by Parallax Inc. By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with Parallax products. Any other uses are not permitted and may represent a violation of Parallax copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by Parallax Inc. Duplication for educational use is permitted, subject to the following Conditions of Duplication: Parallax Inc. grants the user a conditional right to download, duplicate, and distribute this text without Parallax's permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax products, and the user may recover from the student only the cost of duplication.

This text is available in printed format from Parallax Inc. Because we print the text in volume, the consumer price is often less than typical retail duplication charges.

BASIC Stamp, Stamps in Class, Boe-Bot SumoBot, SX-Key and Toddler are registered trademarks of Parallax, Inc. If you decide to use registered trademarks of Parallax Inc. on your web page or in printed material, you must state that "(registered trademark) is a registered trademark of Parallax Inc." upon the first appearance of the trademark name in each printed document or web page. HomeWork Board, Parallax, and the Parallax logo are trademarks of Parallax Inc. If you decide to use trademarks of Parallax Inc. on your web page or in printed material, you must state that "(trademark) is a trademark of Parallax Inc.", "upon the first appearance of the trademark name in each printed document or web page. Other brand and product names are trademarks or registered trademarks of their respective holders.

ISBN 1-928982-35-2

DISCLAIMER OF LIABILITY

Parallax Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax Inc. is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp application, no matter how life-threatening it may be.

INTERNET DISCUSSION LISTS

We maintain active web-based discussion forums for people interested in Parallax products. These lists are accessible from www.parallax.com.

- [Propeller Chip](#) – This list is specifically for our customers using Propeller chips and products.
- [BASIC Stamp](#) – This list is widely utilized by engineers, hobbyists and students who share their BASIC Stamp projects and ask questions.
- [Stamps in Class[®]](#) – Created for educators and students, subscribers discuss the use of the Stamps in Class curriculum in their courses. The list provides an opportunity for both students and educators to ask questions and get answers.
- [Parallax Educators](#) – A private forum exclusively for educators and those who contribute to the development of Stamps in Class. Parallax created this group to obtain feedback on our curricula and to provide a place for educators to develop and obtain Teacher's Guides.
- [Robotics](#) – Designed for Parallax robots, this forum is intended to be an open dialogue for robotics enthusiasts. Topics include assembly, source code, expansion, and manual updates. The Boe-Bot[®], Toddler[®], SumoBot[®], HexCrawler and QuadCrawler robots are discussed here.
- [SX Microcontrollers and SX-Key](#) – Discussion of programming the SX microcontroller with Parallax assembly language SX – Key[®] tools and 3rd party BASIC and C compilers.
- [Javelin Stamp](#) – Discussion of application and design using the Javelin Stamp, a Parallax module that is programmed using a subset of Sun Microsystems' Java[®] programming language.

ERRATA

While great effort is made to assure the accuracy of our texts, errors may still exist. If you find an error, please let us know by sending an email to editor@parallax.com. We continually strive to improve all of our educational materials and documentation, and frequently revise our texts. Occasionally, an errata sheet with a list of known errors and corrections for a given text will be posted to our web site, www.parallax.com. Please check the individual product page's free downloads for an errata file.

ACKNOWLEDGEMENTS

Many thanks to fellow Parallaxians Jen Jacobs for cover and title page art and Chris Savage for technical review of this edition.

Table of Contents

Preface	iii
Author's Note	iii
Getting the Most from StampWorks.....	v
Steps to Success	v
Preparing the StampWorks Lab	1
StampWorks Kit Contents.....	1
Setting Up the Hardware and Software	2
Notes on Using Integrated Circuits in StampWorks Experiments.....	9
Programming Essentials	11
Contents of a Working Program	11
Branching – Redirecting Program Flow	12
Looping – Running Code Again and Again.....	14
Subroutines – Reusable Code that Saves Program Space.....	16
The Elements of PBASIC Style	19
Time to Experiment	25
Learn the Programming Concepts	25
Building the Projects	25
What to do Between Projects	25
Experiment #1: Flash an LED	26
Experiment #2: Flash an LED (Advanced)	29
Experiment #3: Display a Counter with LEDs.....	33
Experiment #4: Science Fiction LED Display	36
Experiment #5: LED Graph (Dot or Bar).....	40
Experiment #6: A Simple Game	46
Experiment #7: A Lighting Controller	51
Building Circuits on Your Own	57
Using 7-Segment LED Displays	59
Experiment #8: A Single-Digit Counter	60
Experiment #9: A Digital Die	63
Experiment #10: A Digital Clock	67
Using Character LCDs	73
Experiment #11: Basic LCD Demonstration	75
Experiment #12: Creating Custom LCD Characters.....	82
Experiment #13: Reading the LCD RAM	88

Moving Forward	93
Experiment #14: Scanning and Debouncing Multiple Inputs	94
Experiment #15: Counting Events	98
Experiment #16: Frequency Measurement	101
Experiment #17: Advanced Frequency Measurement	106
Experiment #18: A Light Controlled Theremin.....	109
Experiment #19: Sound Effects (SFX).....	112
Experiment #20: Infrared Object Detection	119
Experiment #21: Analog Input with PULSIN	123
Experiment #22: Analog Output with PWM	126
Experiment #23: Expanded Digital Outputs with Shift Registers	130
Experiment #24: Expanded Digital Inputs with Shift Registers.....	137
Experiment #25: Mixed IO with Shift Registers	143
Experiment #26: Hobby Servo Control	146
Experiment #27: Stepper Motor Control	150
Experiment #28: Voltage Measurement	156
Experiment #29: Temperature Measurement.....	161
Experiment #30: High Resolution Temperature Measurement	168
Experiment #31: Advanced 7-Segment Multiplexing.....	173
Experiment #32: I2C Communications	179
Experiment #33: Using a Real-Time Clock.....	188
Experiment #34: Serial Communications with a PC	197
Experiment #35: (BONUS) BS2px ADC	206
Power PBASIC	211
Striking Out on Your Own	219

Preface

AUTHOR'S NOTE

Dear friends,

It seems like ages ago that Ken Gracey handed me a new prototyping and development board and asked, “What do you think we could do with this?” That board, of course, was the original NX-1000 and what we went on to create together was the first edition of the book you’re now reading: *StampWorks*.

A lot of things have changed since then, and yet many things remain comfortably constant: there are still many ways to learn microcontroller programming and one of the best – in our opinion – is to do so using the BASIC Stamp[®] microcontroller. Our philosophy has always been rooted in the belief that learning by doing provides the fastest, deepest, most satisfying results. We teach theory by putting it into practice. That’s what *StampWorks* is all about.

Most of you that find your way to *StampWorks* will have had some applicable experience; perhaps you’ve worked your way through our excellent Stamps in Class student guides and are looking to build on that experience. Perhaps you have an electronics and/or programming background and are looking to apply those skills with the BASIC Stamp microcontroller. Either way, this book will teach you to apply the skills that you have and develop new ones along the way so that you can confidently translate your ideas into working projects. Microcontrollers are a part of our daily lives – whether we see them or not – so learning to design with and program them is a very valuable skill.

Like earlier editions, this book assumes that you’re ready to work – ready to read component documentation, willing to open the BASIC Stamp IDE help file for details on a PBASIC command, that you’re unafraid to do a web search if necessary to obtain data that will be required for a challenge; in short, whatever it takes to succeed. We’ll push a bit harder this time, but we’ll do it together. My goal is that even if this isn’t your first exposure to *StampWorks*, it will be a worthwhile and pleasurable experience.

Among the changes that affect this edition of *StampWorks* is an updated PBASIC language: PBASIC 2.5. For those that come from a PC programming background, PBASIC 2.5 will make the transition to embedded programming a bit easier to deal with. And what I'm especially excited about is a new development platform: the Parallax Professional Development Board. My colleague, John Barrowman, with feedback from customers and Parallax staff alike, put about all of the features we would ever want into one beautiful product. For those of you have an NX-1000 (any of the variants), don't worry; most of the experiments will run on it without major modification.

Finally, as far as the text goes, many of the project updates are a direct result of those that have come before you, and you, my friend, have the opportunity to affect future updates. Please, if you ever have a question, comment, or suggestion, feel free to e-mail them to Editor@parallax.com.

A handwritten signature in black ink that reads "Jon Williams". The signature is written in a cursive style with a large, sweeping initial 'J' and a long, trailing flourish at the end.

GETTING THE MOST FROM STAMPWORKS

Before you get started, you'll want to have a copy of the *BASIC Stamp Syntax and Reference Manual* (version 2.1 or higher) handy – either printed or in PDF (available as a free download from www.parallax.com). Through the course of this book I will ask you to review specific sections of the *BASIC Stamp Manual* in preparation for an experiment. At other times I may ask you to go to the Internet to download a datasheet; by doing this we can focus on the details of the experiment and not have to print a lot of redundant information.

STEPS TO SUCCESS

Read (or review if you have previous BASIC Stamp programming experience) sections 1 – 4 of the *BASIC Stamp Syntax and Reference Manual*. This will introduce you to the BASIC Stamp microcontroller, its programming IDE, and its memory organization. And if you've *never* worked with microcontrollers or programming of any kind, I strongly suggest that you download and work your way through our *What's A Microcontroller?* student guide. This outstanding resource is used in schools all over the world and is considered the best introduction to microcontroller principals and programming available anywhere.

The focus of *StampWorks* is on embedded programming and circuit integration. That said, this is not a text on electronics principles. If you are new to the world of electronics, a great beginning text is *Getting Started in Electronics* by renowned electronics author, Forrest M. Mims. You can find this at your favorite bookseller.

Read “Preparing the StampWorks Lab” in the next section. This will introduce you to the Parallax Professional Development Board (PDB) and get it ready for the experiments that follow.

Finally, work your way through the experiments, referring to the *BASIC Stamp Syntax and Reference Manual* (or online Help file) as needed. This is the fun part – and the part that is the most work. Don't allow yourself to be satisfied with simply loading and running the code – dig in and work with it, modify it, make it your own.

By the time you've completed the experiments in this book I believe you will be ready and will have the confidence to take on your own BASIC Stamp microcontroller projects; from projects that may be very simple to those that are moderately complex. The real key is to make sure you truly understand an experiment before

moving on to another. Oftentimes we will rely on what we've previously worked through as support for a new experiment. Taken one at a time, the experiments are not difficult and if you work through them methodically, you'll find your confidence and abilities increasing at a very rapid pace.

Preparing the StampWorks Lab

STAMPWORKS KIT CONTENTS

Before getting to the experiments, let's start by taking inventory of the kit and then preparing the PDB for use in the experiments that follow. Once this is done, you'll be able to move through the experiments smoothly, and when you've completed StampWorks you'll be ready for just about any project you can imagine.

StampWorks Lab Kit Contents #27297			
(parts and quantities subject to change without notice)			
Stock Code #	Description	Marking	Qty
27218	<i>BASIC Stamp Syntax and Reference Manual</i>		1
27220	<i>StampWorks Manual v2.1</i>		1
23138	Professional Development Board		1
BS2-IC	BASIC Stamp 2 module		1
750-00007	Power supply, 12 vdc, 1 amp		1
800-00003	Serial cable		1
805-00006	USB cable, Mini-A to Mini-B		1
700-00050	22-gauge wire, solid, red		1
700-00051	22-gauge wire, solid, white		1
700-00052	22-gauge wire, solid, black		1
200-01030	0.01 μ F capacitor	103	2
200-01040	0.1 μ F capacitor	104	2
150-02210	220 ohm resistor	Red-Red-Brn	3
150-04710	470 ohm resistor	Yel-Vio-Brn	3
150-01020	1 k-ohm resistor	Brn-Blk-Red	3
150-04720	4.7 k-ohm resistor	Yel-Vio-Red	3
150-01030	10 k-ohm resistor	Brn-Blk-Org	3
350-00009	CdS photoresistor		2
350-00003	IR LED		1
350-90000	LED stand-off (for IR LED)		1
350-90001	LED shield (for IR LED)		1
350-00014	IR receiver		1
603-00006	Parallel LCD module		1
604-00009	LM555 timer		1
602-00015	LM358 dual op-amp		1
602-00009	74HC595, serial-in-parallel-out shift register		2
602-00010	74HC165, parallel-in-serial-out shift register		2
ADC0831	ADC0831, 8-bit A/D converter		1
604-00002	DS1620, digital thermometer		1
603-00014	MC14489 LED multiplexer		1
604-00020	24LC32 EEPROM		1
900-00005	Servo, Parallax Standard		1
27964	Stepper motor, 12 vdc, unipolar		1

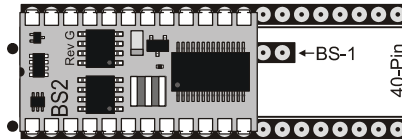
SETTING UP THE HARDWARE AND SOFTWARE

To set up the StampWorks lab for experiments, you'll need the following items:

- Professional Development Board
- BASIC Stamp 2 module
- 12-volt wall pack (2.1 mm, center-positive plug)
- Programming cable (serial or USB)
- Red and black hook-up wire (22-gauge, solid)
- Wire cutters/strippers (not included in the StampWorks Kit)

Installing the BASIC Stamp Module

Start by removing the BASIC Stamp 2 module from its protective foam and carefully inserting it into the 40-pin DIP socket on the PDB (upper-left, near the DB-9 programming connector). You'll notice that the BASIC Stamp 2 module and the PDB socket are marked with semi-circle alignment guides. The BASIC Stamp 2 module should be inserted into the socket so that the alignment guides match. Ensure that the BASIC Stamp 2 module is fully left-aligned in the socket as shown in the illustration below.



Make the Programming Connection

Use a programming cable (either serial or USB, but not both at the same time) to connect the PDB to your PC. It is best to select a serial (COM) port that is not already in use. If, however, you're forced to unplug another device, for example, a PDA or electronic organizer from your computer, make sure that you also disable its communication software before attempting to program your BASIC Stamp microcontroller.



Note: For USB programming, make sure that you have the latest FDTI VCP (Virtual Com Port) driver. Step-by-step installation instructions of the VCP driver may be obtained via the StampWorks Product Page [http at www.parallax.com](http://www.parallax.com).

Computer System Requirements

You will need either a desktop or laptop PC to run the BASIC Stamp Editor software. For the best experience with the StampWorks experiments, check that your computer system meets the following requirements:

- Microsoft Windows® 2000/XP or newer operating system
- An available serial or USB port (with VCP driver installed)
- World Wide Web access



Note: While third-party developers have made BASIC Stamp editors available for operating systems other than Windows, these editors are not supported by Parallax. This text assumes that you're running the official Parallax BASIC Stamp Editor on a Windows computer. If you're using another operating system and editor, you may need to make adjustments in editor-specific instructions.

Installing the BASIC Stamp Editor

Download the latest version of the BASIC Stamp Editor for Windows (version 2.1 or later) from www.parallax.com. Run the program installer, following the on-screen prompts.

Download the StampWorks Program Files

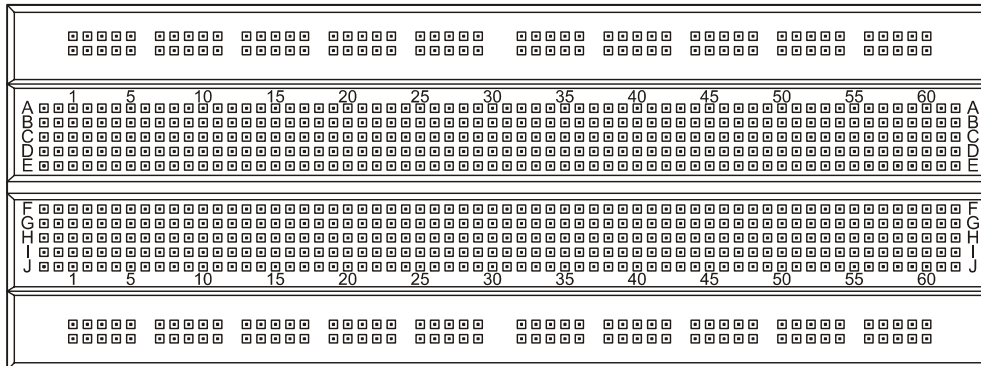
The sample programs listed in this book, with the exception of Experiment 35, were written for the BASIC Stamp 2. These programs and some additional bonus programs are available for free download from www.parallax.com. Many of them contain additional code to support conditional compilation with different BASIC Stamp models.

Preparing the Breadboard

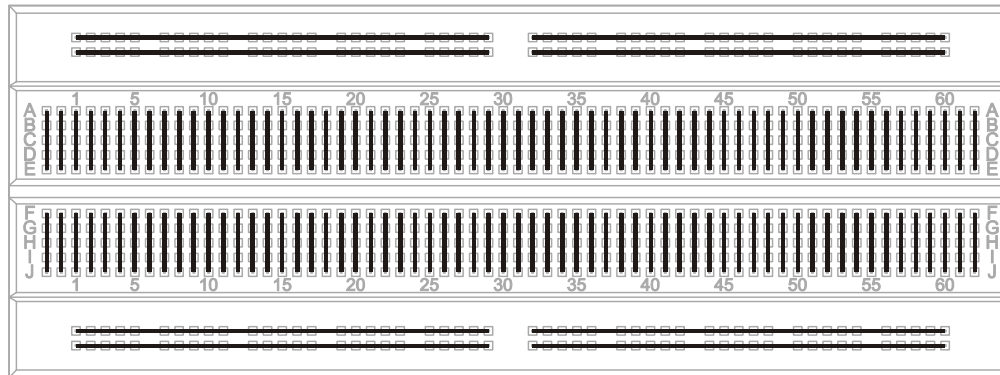
In the center of the PDB is a solderless breadboard where we will build circuits that are not integral to the PDB lab board itself (a variety of components are included in the StampWorks kit). It's important to understand how this breadboard works. With a little bit of preparation, it will be even easier to use with the experiments that follow.

The innermost portion of the breadboard is where we will connect the components. This section of the breadboard consists of several columns of sockets (there are numbers printed along the top for reference). For each column there are two sets of rows. The rows are labeled A through E and F through J, respectively. For any column, sockets A through E are electrically connected. The same holds true for rows F through J.

Above and below the main section of breadboard are two horizontal rows of sockets, each divided in the center. These horizontal rows (often called "rails" or "buses") will be used to carry +5 volts (Vdd) and Ground (Vss). The preparation of the breadboard involves connecting the rails so that they run from end-to-end, connecting the top and bottom rails together and, finally, connecting the rails to the Vdd and Vss connections of the PDB power supply. Here's what the breadboard looks like on the outside:



If the breadboard was X-Rayed, we would see the internal connections and the breaks in the Vdd and Vss rails that need to be connected. Here's a view of the breadboard's internal connections:



Start by setting your wire stripper for 22 gauge (0.34 mm^2). Take the spool of black wire and strip a $\frac{1}{4}$ -inch (6 mm) length of insulation from the end of the wire. With your needle-nose pliers, carefully bend the bare wire 90 degrees so that it looks like this:



Now push the bare wire into the topmost (ground) rail, into the socket that is just above breadboard column 29 (this socket is just left of the middle of the breadboard, near the top). Hold the wire so that it extends to the right. Mark the insulation by lightly pinching it with the wire cutters at the socket above column 32. Be careful not to cut the wire.

Remove the wire from the breadboard and cut it about $\frac{1}{4}$ -inch (6 mm) beyond the mark you just made. With your wire strippers, remove the insulation at the mark. Now bend the second bare end 90 degrees so that the wire forms a squared “U” shape with the insulation in the middle.

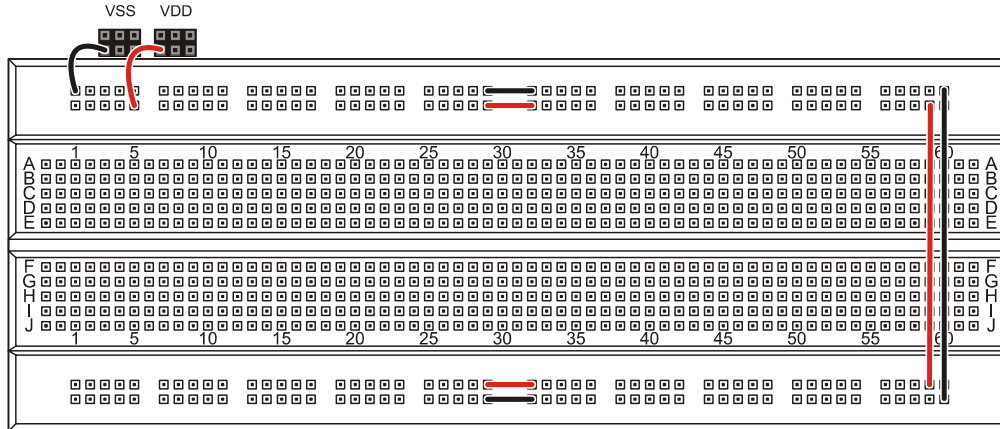


If you've measured and cut carefully, this "U" shaped wire will plug comfortably into the ground rail at sockets 29 and 32. This will create a single ground rail. Repeat this process with black wire for the bottom-most rail. Then, connect the two rails together using the same process at column 60 (right-most sockets on each rail).

With the red wire, connect the top and bottom inside rail halves together. These rails will carry +5 volts, or Vdd. Connect the Vdd rails together at column 59.

Now take a 1½ -inch (4 cm) section of black wire and a 1½ -inch (4 cm) section of red wire and strip ¼ -inch (6 mm) insulation from the ends of both. Bend each wire into a rounded "U" shape. These wires are not designed to lie flat like the other connections, making them easy to remove from the StampWorks lab board if necessary.

Carefully plug one end of the red wire into any of the terminal sockets of the VDD block (near pin 1 of the BASIC Stamp socket) and the other end into the Vdd (+5) rail at column 5. Then, plug one end of the black wire into any of the sockets of the VSS block and other end into the ground rail at column 1. *Be very careful* with these last two connections. If the Vdd and Vss rails get connected together damage may occur when power is applied to the PDB. When completed, the PDB breadboard will look like this:



Final Checkout

With the BASIC Stamp module installed and the breadboard prepared it is time for a final checkout before proceeding to the experiments. If you haven't done so already, connect a programming cable (serial or USB) between your PC and the PDB. Connect a 12-volt DC power supply to the PDB power connector. Move the PDB power switch to ON; a blue LED next to the power switch should illuminate. If it doesn't, move the power switch to OFF and recheck all connections, as well as the power supply.

Start the BASIC Stamp Editor and enter the following short program:

```
' {$STAMP BS2}

Main:
  DEBUG "Ready for StampWorks 2.1!"
  END
```

Now run the program. If all went well the program will be downloaded to the BASIC Stamp module and a Debug Terminal window will appear.



If an error occurs, check the following items:

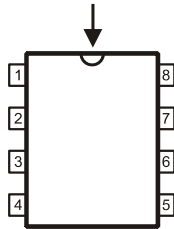
- Is the BASIC Stamp module plugged into the PDB correctly?
- Is the PDB power switch set to ON? Is the blue ON LED lit?
- Is the programming cable connected between the PC and the PDB?
- Have you (manually) selected the wrong PC com port?
- Is the PC com port being used by another program?
- If using USB, have you installed the FTDI VCP driver?

When the Debug Terminal window appears and tells you that the StampWorks lab is ready, it's time to talk about BASIC Stamp programming.

NOTES ON USING INTEGRATED CIRCUITS IN STAMPWORKS EXPERIMENTS

There are two ways to draw integrated circuits (ICs) in a schematic: One way is considered “chip-centric” in which I/O pins appear in the schematic according to their physical location on the device. *StampWorks* uses schematics drawn for efficiency, meaning that I/O pins are placed to make the schematic legible. I/O pins on all chips are counted according to their indicator, starting with Pin 1 and counting in a counter-clockwise direction as shown below:

Indicator denotes
top of device.



Programming Essentials

CONTENTS OF A WORKING PROGRAM

In Sections 1 - 4 of the *BASIC Stamp Syntax and Reference Manual* you were introduced to the BASIC Stamp, its architecture, and the concepts of variables and constants. In this section, we'll introduce the various elements of a program: linear code, branching, loops, and subroutines.

The examples in this discussion use *pseudo-code* to demonstrate and describe program structure. Italics are used to indicate the sections of pseudo-code that require replacement with valid programming statements in order to allow the example to compile and run correctly. You need not enter any of the examples here as all of these concepts will be used in the experiments that follow.

People often think of computers and microcontrollers as “smart” devices and yet, they will do nothing without a specific set of instructions. This set of instructions is called a program, and it is our job to write it. Programs for the BASIC Stamp are written in a language called PBASIC, a Parallax-specific version of the BASIC (Beginner's All-purpose Symbolic Instruction Code) programming language. BASIC is very popular because of its simplicity and English-like syntax. Since its creation at Dartmouth College in the mid 1960's it has become one of the dominant programming languages available for platforms as small as the BASIC Stamp microcontroller, and as large as mainframe computer systems.

A working program can be as simple as a list of statements. Like this:

```
statement 1  
statement 2  
statement 3  
END
```

This is a very simple, yet valid program structure. What you'll find, however, is that most programs do not run in a straight, linear fashion like the listing above. Program flow is often redirected with branching, looping, and subroutines, with short linear sections in between. The requirements for program flow are determined by the goal of the program and the conditions under which the program is running.

BRANCHING – REDIRECTING PROGRAM FLOW

A branching instruction is one that causes the flow of the program to change from its linear path. In other words, when the program encounters a branching instruction, it will, in almost all cases, not be running the next [linear] line of code. The program will usually go somewhere else, often creating a program loop. There are two categories of branching instructions: *unconditional* and *conditional*. PBASIC has two instructions, **GOTO** and **GOSUB** that cause unconditional branching.

Here's an example of an unconditional branch using **GOTO**:

```
Label:
  statement 1
  statement 2
  statement 3
  GOTO Label
```

We call this an unconditional branch because it always happens. **GOTO** redirects the program to another location. The location is specified as part of the **GOTO** instruction and is called an address. Remember that addresses start a line of code and are followed by a colon (:). You'll frequently see **GOTO** at the end of the main body of code, forcing the program statements to run again.

Conditional branching will cause the program flow to change under a specific set of circumstances. The simplest conditional branching is done with an **IF-THEN** construct. PBASIC includes two distinct versions of **IF-THEN**; the first is used specifically to redirect program flow to another point based on a tested condition.

Take a look at this listing:

```
Start:
  statement 1
  statement 2
  statement 3
  IF (condition) THEN Start
```

In this example, statements 1- 3 will run at least once and then continue to run as long as the condition evaluates as True. When required, the condition can be tested prior to the code statements:

```

Start:
  IF (condition) THEN
    statement 1
    statement 2
    statement 3
  ENDIF

```

Note that the code statements are nested in an **IF-THEN-ENDIF** structure which does not require a branch label. If the condition evaluates as False, the program will continue at the line that follows **ENDIF**. Another use of this conditional structure is to add the **ELSE** clause:

```

Start:
  IF (condition) THEN
    statement 1
    statement 2
    statement 3
  ELSE
    statement 4
    statement 5
    statement 6
  ENDIF

```

If the condition evaluates as True then statements 1 – 3 will run, otherwise statements 4 – 6 will run.

As your requirements become more sophisticated, you'll find that you'll want your program to branch to any number of locations based on the value of a control variable. One approach is to use multiple **IF-THEN** constructs.

```

IF (index = 0) THEN Label_0
IF (index = 1) THEN Label_1
IF (index = 2) THEN Label_2

```

This approach is valid and does get used. Thankfully, PBASIC has a special command called **BRANCH** that allows a program to jump to any number of addresses based on the value of an index variable. **BRANCH** is a little more complicated in its setup, but very powerful in that it can replace multiple **IF-THEN** statements. **BRANCH** requires a control (index) variable and a list of addresses

The previous listing can be replaced with one line of code:

```

BRANCH index, [Label_0, Label_1, Label_2]

```

When *index* is zero, the program will branch to **Label_0**, when *index* is one the program will branch to **Label_1** and so on.

Related to **BRANCH** is **ON-GOTO**, in fact, it can serve as direct replacement:

```
ON index GOTO Label_0, Label_1, Label_2
```

Programmers coming from a PC background are probably more familiar with **ON-GOTO**, hence its inclusion in PBASIC 2.5.

LOOPING – RUNNING CODE AGAIN AND AGAIN

As demonstrated in the previous section, program loops can be created with conditional and unconditional branching instructions. Modern variants of BASIC, including PBASIC 2.5, simplify looping with the **DO-LOOP** structure. With **DO-LOOP** the branching label is no longer required. Here's how **DO-LOOP** is used to force *unconditional* looping of number of code statements:

```
DO
  statement 1
  statement 2
  statement 3
LOOP
```

As in the previous example, statements 1 - 3 will run in order, continuously.

The **DO-LOOP** construct can be made conditional by testing before or after the loop statements:

```
DO WHILE (condition)
  statement 1
  statement 2
  statement 3
LOOP
```

In this example the loop statements will only run if and while the condition evaluates as True.

```
DO
  statement 1
  statement 2
  statement 3
LOOP WHILE (condition)
```

In the second example, the loop statements will run at least once, even if the condition evaluates as False. As you can see, the strength of **DO-LOOP** is that it simplifies how and where the condition testing occurs.

DO-LOOP adds another type of testing with **UNTIL**.

```

DO
    statement 1
    statement 2
    statement 3
LOOP UNTIL (condition)

DO UNTIL (condition)
    statement 1
    statement 2
    statement 3
LOOP

```

By using **UNTIL**, the loop statements will run while the condition evaluates as False. And, as demonstrated earlier, placing the test at the end of the loop will cause the loop statements to run at least one time.

Another example of looping is the programmed loop using **FOR-NEXT**.

```

FOR controlVar = startVal TO endVal STEP stepSize
    statement 1
    statement 2
    statement 3
NEXT

```

The **FOR-NEXT** construct is used to run a section of code a specific number of times. **FOR-NEXT** uses a control variable to determine the number of loop iterations. The size of the variable will determine the upper limit of loop iterations. For example, the upper limit when using a byte-sized control variable would be 255. In the example below, *controlVar* could be defined as a Nib (4-bit) variable as the end value is less than 16:

```

FOR controlVar = 1 TO 10
    statement 1
    statement 2
    statement 3
NEXT

```