

Introduction

Thank you for purchasing the BASIC Stamp. At first glance, the Stamp may seem fairly minimal – especially in its new 14-pin SIP form. But, with software that fully utilizes all available hardware, the Stamp gives you useful and powerful features – without the cost of hardware. Its low cost and simplicity make the Stamp perfect for many prototyping and control applications.

Hardware: the Stamp discussed in this manual is the “BS1-IC.” It’s a complete BASIC-programmable computer, packaged as a 14-pin SIP module. The brain of the Stamp is an 18-pin BASIC interpreter chip. A 256-byte EEPROM holds a tokenized version of your program, which is executed by the interpreter. The remainder of the circuit board is taken up by a 4-MHz resonator, 5-volt regulator, and brown-out circuit.

The BS1-IC has a corresponding “carrier board,” which provides the usual battery clips, prototyping area, programming connector, and reset button. By plugging the BS1-IC into its carrier board, you create a traditional BASIC Stamp. The extra features of the carrier board are most useful for programming and prototyping.

BASIC Language: the Stamp is programmed in our simple BASIC language (sometimes called *PBASIC*). The language includes familiar instructions, such as GOTO, FOR...NEXT, and IF...THEN, as well as SBC instructions, such as SERIN (serial input), PWM, and BUTTON.

Each instruction takes 2-3 bytes of space, resulting in a maximum program size of 80-100 instructions. As for speed, the interpreter executes about 2,000 instructions per second.

Programming: to write programs for the Stamp, you’ll need the Stamp Programming Package. The package includes the software, cable, and documentation necessary to program Stamps. Purchasing the package also entitles you to free technical support.

Important Information

Warranty

Parallax warrants its products against defects in materials and workmanship for a period of 90 days.

If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Please contact us before shipping a return, since we may need to issue an RMA number. After repairing or replacing your product, we will ship it back to you using the same shipping method used to ship the product to Parallax (for instance, if you ship your product via overnight express, we will do the same).

This warranty does not apply if the product has been modified or damaged by accident, abuse, or misuse.

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 refund. Parallax will refund the purchase price, excluding shipping/handling costs. If the product has been altered or damaged, a partial refund will be given.

Copyrights and Trademarks

Copyright © 1995 by Parallax, Inc. All rights reserved. Parallax, the Parallax logo, and BASIC Stamp are registered trademarks of Parallax, Inc. PIC is a registered trademark of Microchip Technology, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Disclaimer of Liability

Parallax 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 for recovering, reprogramming, or reproducing any data stored in or used with Parallax products.

Special Note about Reseller Sales

If you purchased your Parallax product through a reseller, you should contact the reseller if there is a problem. They may have policies that differ from those of Parallax. Also, especially if you are outside the United States, they can usually offer faster assistance.

Parallax BBS and Internet Access

Parallax has a 24-hour BBS for your convenience. Customers can call our BBS to obtain the latest versions of Parallax software or to try software before purchasing the product.

The BBS telephone number is (916) 624-7101.

Settings are: 300-14400 baud, 8 data bits, 1 stop bit, no parity.

To reach us on Internet, send e-mail or ftp to *parallaxinc.com*.

Table of Contents

Stamp Programming	5
System Requirements.....	5
Packing List	5
Connecting to the PC	6
Stamp Hardware	7
BS1-IC Pin-Out.....	7
Carrier Board Features.....	7
General Stamp Schematic.....	8
Regulator Current Limits	8
I/O Port & Variable Space	9
Common Questions	11
Example Application	13
Using the Editor	14
Starting the Editor	14
Program Formatting.....	14
Entering & Editing Programs	18
Editor Function Keys	18
Running Your Program	20
Loading a Program from Disk.....	20
Saving a Program on Disk.....	20
Using Cut, Copy, and Paste	21
Using Search & Replace.....	21
Instruction Set Summary	23
BASIC Instructions	25
BRANCH	25
BUTTON	26
DEBUG.....	28
EEPROM	29
END	30
FOR...NEXT	31
GOSUB	33
GOTO	34
HIGH.....	35
IF...THEN.....	36
INPUT	37
LET.....	38
LOOKDOWN.....	40
LOOKUP.....	41

Table of Contents

LOW	42
NAP	43
OUTPUT	45
PAUSE.....	46
POT	47
PULSIN	49
PULSOUT	50
PWM.....	51
RANDOM.....	53
READ.....	54
RETURN	55
REVERSE	56
SERIN	57
SEROUT	61
SLEEP	64
SOUND	65
TOGGLE	66
WRITE	67

Application Notes 69

Note #1: LCD user-interface terminal.....	69
Note #2: Interfacing an A/D convertor.....	75
Note #3: Hardware solution for keypads.....	79
Note #4: Controlling and testing servos.....	83
Note #5: Practical pulse measurements.....	89
Note #6: A serial stepper-motor controller	97
Note #7: Using a thermistor	101
Note #8: Sending Morse code	107
Note #9: Constructing a dice game	111
Note #10: Humidity and temperature	113
Note #11: Infrared communication	117
Note #12: Sonar rangefinding	121
Note #13: Using serial EEPROMs.....	127
Note #14: Networking multiple Stamps.....	133
Note #15: Using PWM for analog output.....	139
Note #16: Keeping Stamp programs private	143
Note #17: Solar-powered Stamp.....	147
Note #18: One pin, many switches.....	153
Note #19: Using the button instruction effectively.....	157
Note #20: An accurate timebase	165
Note #21: Fun with trains	169

Parallax Distributors xxx

Stamp Programming

System Requirements

To program Stamps, you'll need the following computer system:

- IBM PC or compatible computer
- 3.5-inch disk drive
- Parallel port
- 128K of RAM
- MS-DOS 2.0 or greater

To power the Stamps, you can use a 9-volt battery (this is the most convenient method). You can also use a 5-15 volt power supply, but you should be careful to connect the supply to the appropriate part of the Stamp. A 5-volt supply should be connected directly to the Stamp's +5V pin, but a higher voltage should be connected to the Stamp's PWR pin.

Connecting a high voltage supply (greater than 6 volts) to the 5-volt pin can permanently damage the Stamp.

Packing List

If you purchased the Stamp Programming Package, you should have received the following items:

- Programming cable
- 3.5-inch diskette
- Application notes (back of this manual)
- Stamp manual (this manual)

If any items are missing, please let us know.

Stamp Programming

Connecting to the PC

To program a Stamp, you'll need to connect it to your PC and then run the editor/downloader software. In this manual, it's assumed that your Stamp is a BS1-IC, and that you have the corresponding carrier board.

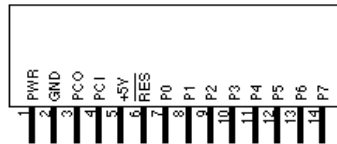
To connect the Stamp to your PC, follow these steps:

- 1) Plug the BS1-IC onto the carrier board. The BS1-IC plugs into a 14-pin SIP socket, located near the battery clips on the carrier. When plugged onto the carrier board, the components on the BS1-IC should face the battery clips.
- 2) In the Stamp Programming Package, you received a cable to connect the Stamp to your PC. The cable has two ends, one with a DB25 connector and the other with a 3-pin connector. Plug the DB25 end into an available **parallel** port on your PC.
- 3) Plug the remaining end of the cable onto the 3-pin header on the carrier board. On the board and the cable, you'll notice a double-arrow marking; the markings on the cable and board should match up.
- 4) Supply power to the carrier board, either by connecting a 9-volt battery or by providing an external power source.

With the Stamp connected and powered, run the editor/downloader software as described later in this manual.

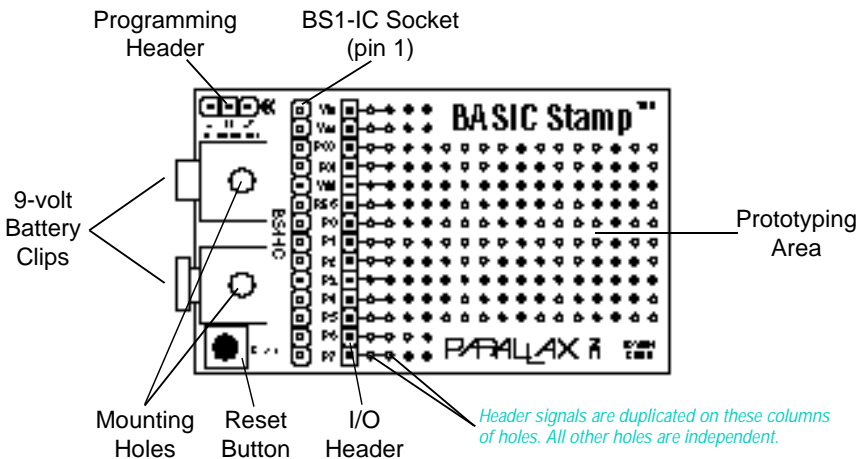
Stamp Hardware

BS1-IC



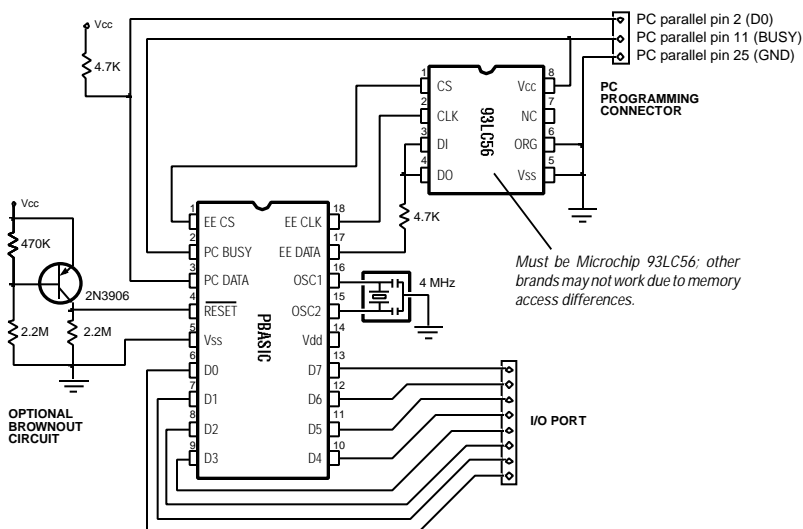
*Shown at 125%
of actual size*

- PWR** *Unregulated power in.* Accepts 6-15 VDC, which is then regulated to 5 volts. May be left unconnected if 5 volts is applied to the +5V pin.
- GND** *System ground.* Connects to PC parallel port pin 25 (GND) for programming.
- PCO** *PC Out.* Connects to PC parallel port pin 11 (BUSY) for programming.
- PCI** *PC In.* Connects to PC parallel port pin 2 (D0) for programming.
- +5V** *5-volt input/output.* If an unregulated voltage is applied to the PWR pin, then this pin will output 5 volts. If no voltage is applied to the PWR pin, then a regulated voltage between 4.5V and 5.5V should be applied to this pin.
- RES** *Reset input/output.* Goes low when power supply is less than 4 volts, causing the BS1-IC to reset. Can be driven low to force a reset.
- P0-P7** *General-purpose I/O pins.* Each can sink 25 mA and source 20 mA. However, the total of all pins should not exceed 50 mA (sink) and 40 mA (source).



Stamp Hardware

General Stamp Schematic*:



* The BS1-IC has a slightly different schematic. For reset and brownout circuits, the BS1-IC uses two components from Seiko, rather than the transistor circuit shown above. However, this schematic is the same in all other respects, and serves as an example of how simple the Stamp circuit is to implement.

Current Limits of the On-Board Regulator

In some cases, you may want to know how much current the Stamp can handle with its on-board regulator. Basically, at higher supply voltages, the regulator can handle less current. The Stamp itself takes 1-2 mA, so any current “left over” can be used to drive external circuits. The table below shows the approximate current limits at various voltages:

Power Supply (volts)	Total Current (mA)
5-9	50
12	40
25	10
40	2-3

We recommend a lower supply voltage (5-15 VDC). However, the Stamp will run at higher voltages (with low current consumption).

I/O Port & Variable Space

The BASIC Stamp has 16 bytes of RAM devoted to I/O and the storage of variables. The first two bytes are used for I/O (1 for actual pins, 1 for direction control), leaving 14 bytes for data. This arrangement of variable space is shown below:

Word Name	Byte Names	Bit Names	Special Notes
Port	Pins Dirs	Pin0-Pin7 Dir0-Dir7	<i>I/O pins; bit addressable. I/O pin direction control; bit addressable.</i>
W0	B0 B1	Bit0-Bit7 Bit8-Bit15	<i>Bit addressable. Bit addressable.</i>
W1	B2 B3		
W2	B4 B5		
W3	B6 B7		
W4	B8 B9		
W5	B10 B11		
W6	B12 B13		<i>Used by GOSUB instruction. Used by GOSUB instruction.</i>

The Stamp's BASIC language allows a fair amount of flexibility in naming variables and I/O pins. Depending upon your needs, you can use the variable space and I/O pins as bytes (*Pins, Dirs, B0-B13*) or as 16-bit words (*Port, W0-W6*). Additionally, the I/O pins and the first two data bytes can be used as individual bits (*Pin0-Pin7, Dir0-Dir7, Bit0-Bit15*). In many cases, a single bit may be all you need, such as when storing a status flag.

I/O Port & Variable Space

Port is a 16-bit word, which is composed of two bytes, **Pins** and **Dirs**:

Pins (byte) and **Pin0-Pin7** (corresponding bits) are the I/O port pins. When these variables are read, the I/O pins are read directly. When these variables are written to, the corresponding RAM is written to, which is then transferred to the I/O pins before each instruction.

Dirs (byte) and **Dir0-Dir7** (corresponding bits) are the I/O port direction bits. A “0” in one of these bits causes the corresponding I/O pin to be an input; a “1” causes the pin to be an output. This byte of data is transferred to the I/O port’s direction register before each instruction.

When you write your BASIC programs, you’ll use the symbols described above to read and write the Stamp’s 8 I/O pins.

Normally, you’ll start your program by defining which pins are inputs and which are outputs. For instance, “dirs = %00001111” sets bits 0-3 as outputs and bits 4-7 as inputs (right to left).

After defining which pins are inputs and outputs, you can read and write the pins. The instruction “pins = %11000000” sets bits 6-7 high. For reading pins, the instruction “b2 = pins” reads all 8 pins into the byte variable b2.

Pins can be addressed on an individual basis, which may be easier. For reading a single pin, the instruction “Bit0 = Pin7” reads the state of I/O pin 7 and stores the reading in bit variable Bit0. The instruction “if pin3 = 1 then start” reads I/O pin 3 and then jumps to start (a location) if the pin was high (1).

The Stamp’s editor software recognizes the variable names shown on the previous page. If you’d like to use different names, you can start your program with instructions to define new names:

symbol switch = pin0
symbol flag = bit0
symbol count = b2

'Define label "switch" for I/O pin 0
'Define label "flag" for bit variable bit0
'Define label "count" for byte variable b2

Common Questions

Can I expand the Stamp's BASIC program memory?:

No. The BASIC interpreter only addresses 8 bits of program space, which results in the 256-byte limitation. Using a larger EEPROM, such as the Microchip 93LC66, won't make any difference.

What voltage range can I use to power the Stamp:

We encourage people to use a 9-volt battery to power the Stamp, especially if they have the carrier board. The battery is simple and can power the Stamp for days, even weeks if sleep mode is used.

However, if you want to use an external power supply, you can use anything that supplies 5-15 volts DC at a minimum of 2 mA (not including I/O current needs).

If you have a 5-volt supply, connect it to the Stamp's +5V pin. This will route power directly to the Stamp circuit, bypassing the voltage regulator.

If you have a 6-15 volt supply, connect it to the Stamp's PWR pin. This will route the power through the on-board 5-volt regulator.

Can I use the Stamp to power external circuits?:

Yes. If you need to supply 5 volts, connect your circuit to the Stamp's +5V pin. If you need the unregulated input voltage (6-15 volts), connect your circuit to the Stamp's PWR pin.

How long can the Stamp run on a 9-volt battery?:

This depends on what you're doing with the Stamp. If your program never uses sleep mode and has several LED's connected to I/O lines, then the Stamp may only run for several hours. If, however, sleep mode is used and I/O current draw is minimal, then the Stamp can run for weeks.

What are the sink and source capabilities of the Stamp's I/O lines?:

The Stamp's I/O pins can each sink 25 mA and source 20 mA. However, the total sink and source for all 8 I/O lines should not exceed 50 mA (sink) and 40 mA (source).

Common Questions

Does the BASIC language used in the Stamp support floating point math?:

No. The Stamp only works with integer math, which means that no fractions are allowed. Expressions must be given as integers, and any results are given as integers. For instance, if you gave the Stamp an instruction to divide 5 by 2, it would return a result of 2, not 2.5; the remainder (.5) is simply lost.

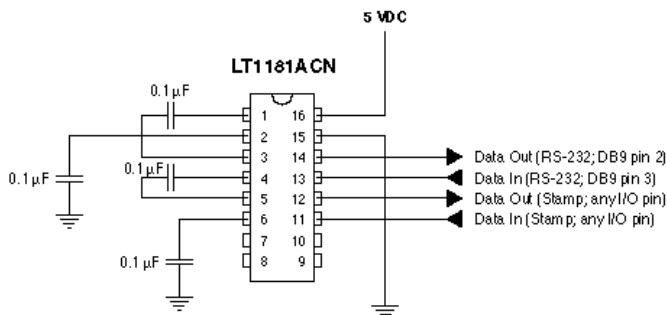
How does the Stamp evaluate mathematical expressions?:

Mathematical expressions are evaluated strictly left to right. This is important, since you may get different results than you expect. For instance, under normal rules, the expression $2 + 3 \times 4$ would be solved as $2 + (3 \times 4)$, since multiplication takes priority over addition. The result would be 14. However, since the Stamp solves expressions from left to right, it would be solved as $(2 + 3) \times 4$, for a result of 20.

When writing your programs, please remember that the left-to-right evaluation of expressions may affect the results.

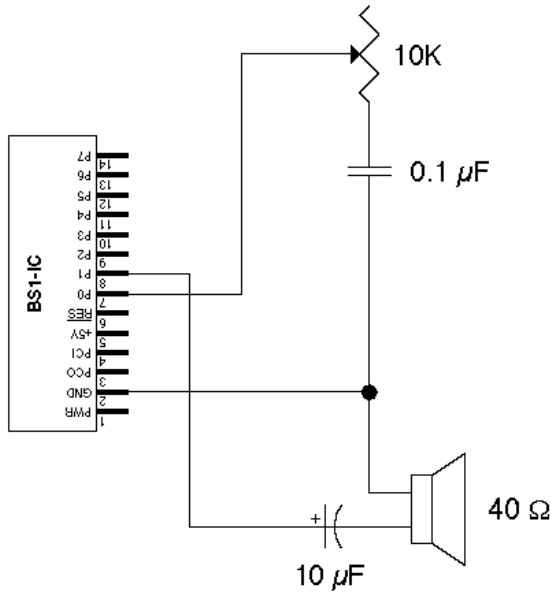
What do I need to make the Stamp support real RS-232 voltages?

The Stamp's I/O pins operate at TTL voltages (0-5 volts), so the SERIN and SEROUT instructions operate at these voltages. This is fine for most applications, such as Stamps communicating with Stamps. However, some PCs may not accept TTL voltages, especially when the PC is receiving data. If you need real RS-232 voltages, you can use the circuit shown below. The LT1181ACN is available from various distributors, including Digi-Key (call 800-344-4539).



Example Application

This page shows a simple application for a Stamp (in this case, the BS1-IC). The purpose of the application is to read the value of the potentiometer and then generate a corresponding tone on the speaker. As the potentiometer value changes, so does the tone. For interesting variations, the potentiometer could easily be changed to a thermistor or photocell.



loop:

pot 0,100,b2

'Read potentiometer on pin 0 and
'store result in variable b2.

b2=b2/2

'Divide result so highest value
'will be 128.

sound 1,(b2,10)

'Generate a tone using speaker
'on pin 1. Frequency is set by
'value in b2. Duration of tone
'is set to 10.

goto loop

'Repeat the process.

Using the Editor

Starting the Editor

With the Stamp connected and powered, run the editor software by typing the following command from the DOS prompt:

STAMP

Assuming you're in the proper directory, the Stamp software will start running after several seconds. The editor screen is dark blue, with one line across the top that names various disk and downloading functions. Except for the top line, the entire screen is available for entering and editing BASIC programs.

Program Formatting

There are few restrictions on how programs are entered. However, you should know the rules for entering constants, labels, and comments, as described in the following pages:

- **Constants:** constant values can be declared in four ways: decimal, hex, binary, and ASCII.

Hex numbers are preceded with a dollar sign (\$), binary numbers are preceded with a percent sign (%), and ASCII values are enclosed in double quotes ("). If no special punctuation is used, then the editor will assume the value is decimal. Following are some examples:

100	'Decimal
\$64	'Hex
%01100100	'Binary
"A"	'ASCII "A" (65)
"Hello"	'ASCII "H", "e", "l", "l", "o"

Most of your programs will probably use decimal values, since this is most common in BASIC. However, hex and binary can be useful. For instance, to define pins 0-3 as outputs and pins 4-7 as inputs, you could use any of the following, but the binary example is the most readable:

dirs = 15	'Decimal
dirs = \$0F	'Hex
dirs = %00001111	'Binary

Using the Editor

- **Address Labels:** the editor uses labels to refer to addresses (locations) within your program. This is different from some versions of BASIC, which use line numbers.

Generally speaking, label names can be any combination of letters, numbers, and underscores (_), but the first character of the name must not be a number. Also, label names cannot use reserved words, such as instruction names (serin, toggle, goto, etc.) and variable names (port, w2, b13, etc.)

When first used, label names must end with a colon (:). When called elsewhere in the program, labels are named without the colon. The following example illustrates how to use a label to refer to an address:

```
loop:      toggle 0                'Toggle pin 0

           for b0 = 1 to 10
           toggle 1                'Toggle pin 1 ten times
           next

           goto loop              'Repeat the process
```

- **Value Labels:** along with program addresses, you can use labels to refer to variables and constants. Value labels share the same syntax rules as address labels, but value labels don't end with a colon (:), and they must be defined using the "symbol" directive. The following example shows several value labels:

```
symbol start = 1                'Define two constant
symbol end = 10                  'labels

symbol count = b0                'Define a label for b0

loop:    for count = start to end
         toggle 1                'Toggle pin 1 ten times
         next
```

Using the Editor

- **Comments:** comments can be added to your program to make it more readable.

Comments begin with an apostrophe (') and continue to the end of the line. Alternatively, you can designate a comment using the standard REM statement found in many versions of BASIC. An example follows:

```
symbol relay = 3           'Make label for I/O pin 3
symbol length = w2        'Make label for w2

      dirs = %11111111    'Make all pins outputs
      pins = %00000000    'Make all pins low

REM this is the main loop

main:   length = length + 10      'Increase length by 10
        gosub sub                'Call pulse out routine
        goto main                'Loop back

sub:    pulsout relay,length : toggle 0 : return
```

- **General Format:**

The editor is not case sensitive, except when processing strings (such as “hello”).

Multiple instructions and labels can be combined on the same line by separating them with colons (:).

The following example shows the same program as separate lines and as a single-line...

Multiple-line version:

```
      dirs = 255           'Make all pins outputs
      for b2 = 0 to 100    'Count from 0 to 100
      pins = b2            'Make pins = count (b2)
      next                 'Continue counting til 100
```

Single-line version:

```
      dirs = 255 : for b2 = 0 to 100 : pins = b2 : next
```


Using the Editor

- **Mathematical Operators:** the following operators may be used in mathematical expressions:

+	add
-	subtract
*	multiply (returns low word of result)
**	multiply (returns high word of result)
/	divide (returns quotient)
//	divide (returns remainder)
min	keep variable greater than or equal to value
max	keep variable less than or equal to value
&	logical AND
	logical OR
^	logical XOR
&/	logical AND NOT
/	logical OR NOT
^/	logical XOR NOT

Some examples:

count = count + 1	'Increment count
timer = timer * 2	'Multiply timer by 2
b2 = b2 / 8	'Divide b2 by 8
w3 = w3 & 255	'Isolate lower byte of w3
b0 = b0 + 1 max 99	'Increment b0, but don't 'allow b0 to exceed 99
b3 = b3 - 1 min 10	'Decrement b3, but don't 'allow b3 to drop below 10

Using the Editor

Entering & Editing Programs

As covered in the previous pages, there are some rules to remember about the use of constants, labels, and comments. However, for the most part, you can format your programs as you see fit.

We've tried to make the editor as intuitive as possible: to move up, press the *up arrow*; to highlight one character to the right, press *shift-right arrow*; etc.

Most functions of the editor are easy to use. Using single keystrokes, you can perform the following common functions:

- Load, save, and run programs.
- Move the cursor in increments of one character, one word, one line, one screen, or to the beginning or end of a file.
- Highlight text in blocks of one character, one word, one line, one screen, or to the beginning or end of a file.
- Cut, copy, and paste highlighted text.
- Search for and/or replace text.

Editor Function Keys

The following list shows the keys that are used to perform various functions:

Alt-R	Run program in Stamp (<i>download the program on the screen, then run it</i>)
Alt-L	Load program from disk
Alt-S	Save program on disk
Alt-Q	Quit editor and return to DOS
Enter	Enter information and move down one line
Tab	Same as Enter
Left arrow	Move left one character
Right arrow	Move right one character

Using the Editor

Up arrow	Move up one line
Down arrow	Move down one line
Ctrl-Left	Move left to next word
Ctrl-Right	Move right to next word
Home	Move to beginning of line
End	Move to end of line
Page Up	Move up one screen
Page Down	Move down one screen
Ctrl-Page Up	Move to beginning of file
Ctrl-Page Down	Move to end of file
Shift-Left	Highlight one character to the left
Shift-Right	Highlight one character to the right
Shift-Up	Highlight one line up
Shift-Down	Highlight one line down
Shift-Ctrl-Left	Highlight one word to the left
Shift-Ctrl-Right	Highlight one word to the right
Shift-Home	Highlight to beginning of line
Shift-End	Highlight to end of line
Shift-Page Up	Highlight one screen up
Shift-Page Down	Highlight one screen down
Shift-Ctrl-Page Up	Highlight to beginning of file
Shift-Ctrl-Page Down	Highlight to end of file
Shift-Insert	Highlight word at cursor
ESC	Cancel highlighted text
Backspace	Delete one character to the left
Delete	Delete character at cursor
Shift-Backspace	Delete from left character to beginning of line
Shift-Delete	Delete to end of line
Ctrl-Backspace	Delete line
Alt-X	Cut marked text and place in clipboard
Alt-C	Copy marked text to clipboard
Alt-V	Paste (insert) clipboard text at cursor
Alt-F	Find string (establish search information)
Alt-N	Find next occurrence of string
Alt-P	Calibrate potentiometer scale (see POT instruction for more information)

Using the Editor

Running Your Program

To run the program shown on the screen, press Alt-R. The editor software will check all available parallel ports, searching for a BASIC Stamp. If it finds one, it will download and run your program. Note that any program already in the Stamp will be overwritten. If the editor is unable to locate a Stamp, it will display an error message.

Assuming that you have a Stamp properly connected to your PC, the editor will display a bargraph, which shows how the download of your program is progressing. Typical downloads take only several seconds, so the graph will fill quickly.

As the graph fills, you'll notice that some of the graph fills with white blocks, while the remainder fills with red blocks. These colors represent how much of the Stamp's EEPROM space is used by the program. White represents available space, and red represents space occupied by the program. This provides an easy way to determine how much of the EEPROM remains available for additional instructions or data.

When the download is complete, your program will automatically start running in the Stamp. If you used the debug directive in your program, it will display its data when it's encountered in the program.

To remove the download graph from the screen, press any key.

Loading a Program from Disk

To load a BASIC program from disk, press Alt-L. A small box will appear, prompting you for a filename. If you entered the filename correctly, the program will be loaded into the editor. Otherwise, an error message will be displayed.

If you decide not to load a program, press ESC to resume editing.

Saving a Program on Disk

To save a BASIC program on disk, press Alt-S. A small box will appear, prompting you for a filename. After the filename is entered, the editor will save your program.

Using the Editor

Using Cut, Copy, and Paste

Like most word processors, the editor can easily cut, copy, and paste text. If you need to make major changes to your program, or your program has many repetitive routines, these functions can save a lot of time.

The function of the cut, copy, and paste routines is to cut or copy highlighted text to the *clipboard* (the clipboard is an area of memory set aside by the editor). Text in the clipboard can later be pasted (inserted) anywhere in your program. Both *cut* and *paste* copy text to the clipboard, but *cut* also removes the text from its current location.

Please note that *cutting* text is different from *deleting* it. While both functions remove text from its current location, only *cut* saves the text to the clipboard – *delete* removes it entirely.

As an example of cut and paste, let's cut a section of text and then paste it elsewhere. The following steps will guide you through the process:

- First, you need to **highlight some text**. For this example, let's highlight everything from the cursor to the end of the line. To do this, press Shift-End (everything from the cursor to the end of the line should become highlighted).
- Second, with the line highlighted, **press Alt-X (cut)**. The text should disappear.
- Third, move the cursor to another location – anywhere is fine. Then, **press Alt-V (paste)**. The text should appear where the cursor was, pushing any following text down as necessary.

The first step could be replaced with copy (Alt-C), instead of cut (Alt-X). The only difference would be that the text would appear in its original location, as well as the pasted location.

Using the Editor

Using Search & Replace

The editor has a function that allows you to search for and/or replace text. In many instances, this function can be very useful. For example, you may decide to change a variable name throughout your program. Doing so manually would take a lot of time, but with search and replace, it takes just seconds.

To set the search criteria, **press Alt-F (find)**. A small box will appear in the center of the screen, requesting a search string and an optional replacement string. To perform the search, follow these steps:

- **Enter the search string.** If you want to search for a string that contains the *Tab* or *Return* keys, you can do so by typing *Ctrl-Tab* or *Ctrl-Return*; “•” will appear for each tab, “ ” for each return.
- **Enter the replacement string, if necessary.** If you enter a replacement string, it will be copied to the *clipboard* (the clipboard is an area of memory set aside by the editor). During the search, you will have the option to replace individual occurrences of the search string with the replacement string.

If you only want to search (without the option to replace), just press *Enter* for the replacement string.

- The editor will remove the search criteria box and highlight the **first occurrence** of the search string.

To replace the highlighted string with the replacement string, press **Alt-V (paste)**.

To find the next occurrence of the search string, **press Alt-N**.

Instruction Set Summary

BRANCHING

IF...THEN	<i>Compare and conditionally branch.</i>
BRANCH	<i>Branch to address specified by offset.</i>
GOTO	<i>Branch to address.</i>
GOSUB	<i>Branch to subroutine at address. Up to 16 GOSUB's are allowed.</i>
RETURN	<i>Return from subroutine.</i>

LOOPING

FOR...NEXT	<i>Establish a FOR...NEXT loop.</i>
------------	-------------------------------------

NUMERICS

{LET}	<i>Perform variable manipulation, such as $A=5$, $B=A+2$, etc. Possible operations are add, subtract, multiply, divide, max. limit, min. limit, and logical operations AND, OR, XOR, AND NOT, OR NOT, and XOR NOT.</i>
LOOKUP	<i>Lookup data specified by offset and store in variable. This instruction provides a means to make a lookup table.</i>
LOOKDOWN	<i>Find target's match number (0-N) and store in variable.</i>
RANDOM	<i>Generate a pseudo-random number.</i>

DIGITAL I/O

OUTPUT	<i>Make pin an output.</i>
LOW	<i>Make pin output low.</i>
HIGH	<i>Make pin output high.</i>
TOGGLE	<i>Make pin an output and toggle state.</i>
PULSOUT	<i>Output a timed pulse by inverting a pin for some time.</i>
INPUT	<i>Make pin an input</i>
PULSIN	<i>Measure an input pulse.</i>
REVERSE	<i>If pin is an output, make it an input. If pin is an input, make it an output.</i>
BUTTON	<i>Debounce button, perform auto-repeat, and branch to address if button is in target state.</i>

Instruction Set Summary

SERIAL I/O

SERIN	<i>Serial input with optional qualifiers and variables for storage of received data. If qualifiers are given, then the instruction will wait until they are received before filling variables or continuing to the next instruction. Baud rates of 300, 600, 1200, and 2400 are possible. Data received must be with no parity, 8 data bits, and 1 stop bit.</i>
SEROUT	<i>Send data serially. Data is sent at 300, 600, 1200, or 2400 baud, with no parity, 8 data bits, and 1 stop bit.</i>

ANALOG I/O

PWM	<i>Output PWM, then return pin to input. Used to output analog voltages (0-5V) using a capacitor and resistor.</i>
POT	<i>Read a 5-50K potentiometer and scale result.</i>

SOUND

SOUND	<i>Play notes. Note 0 is silence, notes 1-127 are ascending tones, and notes 128-255 are white noises.</i>
-------	--

EEPROM ACCESS

EEPROM	<i>Store data in EEPROM before downloading BASIC program.</i>
READ	<i>Read EEPROM byte into variable.</i>
WRITE	<i>Write byte into EEPROM.</i>

TIME

PAUSE	<i>Pause execution for 0–65536 milliseconds.</i>
-------	--

POWER CONTROL

NAP	<i>Nap for a short period. Power consumption is reduced.</i>
SLEEP	<i>Sleep for 1-65535 seconds. Power consumption is reduced to approximately 20 μA.</i>
END	<i>Sleep until the power cycles or the PC connects. Power consumption is reduced to approximately 20 μA.</i>

PROGRAM DEBUGGING

DEBUG	<i>Send variables to PC for viewing.</i>
-------	--