BASIC STAMP FAQS

(FREQUENTLY ASKED QUESTIONS AND ANSWERS)

What is the BASIC Stamp?

The BASIC Stamp is a microcontroller developed by Parallax, Inc. which is easily programmed using a form of the BASIC programming language. It is called a “Stamp” simply because it is close to the size of an average postage stamp.

Does the BASIC Stamp lose its program when I remove the battery or power supply?

No, your PBASIC code is stored inside a serial EEPROM on-board the BASIC Stamp. EEPROMs provide non-volatile storage; they retain memory even without power. The EEPROM used is the BASIC Stamp is guaranteed to function properly for 40 years and for 10,000,000 write cycles per memory location.

Does the BASIC Stamp come in different versions?

Yes. Currently there are two functional versions of the BASIC Stamp and three physical versions. The BASIC Stamp line consists of the BASIC Stamp I and the BASIC Stamp II. The BASIC Stamp I is available in two package types (physical versions): 1) a through-hole/socketed package (with prototyping area) called the BASIC Stamp Rev. D and 2) a 14-pin SIP, surface-mount package called the BS1-IC. It’s important to note the both versions are functionally equivalent (with the exception of the Reset pin) but are just in physically different packages. The BASIC Stamp II is only available in a 24-pin DIP, surface-mount package called the BS2-IC.

What is the difference between the BASIC Stamp Rev. D (sometimes just called “BASIC Stamp”) and the BASIC Stamp I?

The original version of the BASIC Stamp I was simply called, “BASIC Stamp”. That name was sufficient until the arrival of the new physical package type, as well as the second version, the BASIC Stamp II. At that point, the original version was split into to package types, the Rev. D and the BS1-IC; both of which are considered to be the “BASIC Stamp I”. The “BASIC Stamp II” is known as the BS2-IC. Today, the term “BASIC Stamp” is too generic and really could be applied to any one of the units in the product line. Long-time Parallax customers, however, still frequently refer to the BASIC Stamp I as the BASIC Stamp, thus, in many cases, there is no difference between the term BASIC Stamp and the BASIC Stamp I. To keep things clear, we prefer the names: “BASIC Stamp I”, “BASIC Stamp II”, or more specifically, “Rev. D”, “BS1-IC” and “BS2-IC” and try to use the term “BASIC Stamp” to refer to the entire line of microcontrollers.

How big is the BASIC Stamp?

The BASIC Stamp rev. D measures 2.5” (65 mm) L x 1.52” (39 mm) W x 0.4” (10 mm) H.
The BS1-IC measures 1.4” (35 mm) L x 0.58” (15 mm) W x 0.13” (3 mm) H.
The BS2-IC measures 1.2” (30 mm) L x 0.62” (16 mm) W x 0.35” (9 mm) H.

How do I power the BASIC Stamp?

The BASIC Stamp runs off of 5 to 15 volts DC. Both BASIC Stamp I and II feature an on-board 5-volt regulator which converts the input 6 to 15 volts (on the VIN pin) to the 5 volts that it’s components require. If your power supply is 6 to 15 volts, you should connect it directly to the VIN and GND pins (1 and 2 on the BASIC Stamp rev. D or BS1-IC, and 24 and 23 on the BS2-IC) or to the battery clips on the carrier board. IF you power supply delivers a regulated 5 volts, you should connect it directly to the +5V and GND pins (14 and 2 on the BASIC Stamp rev. D, 5 and 2 on the BS1-IC, and 21 and 23 on the BS2-IC).

How much current does the BASIC Stamp consume?

The BASIC Stamp I (both rev. D and BS1-IC) consumes 2 mA in running mode and 20 uA in sleep mode not including any circuitry on the I/O pins. The BS2-IC consumes 7 mA in running mode and 30 uA in sleep mode not including any circuitry on the I/O pins.

Is the BASIC Stamp sensitive to static electricity?

While many electronic devices, including the BASIC Stamp, can be damaged by static electricity, the BASIC Stamps are generally more impervious to static. We do, however, recommend taking all the usual precautions when handling BASIC Stamps in static prone environments.

Are there production possibilities with the BASIC Stamp?

Yes. We offer the major components of the BASIC Stamp circuit (the interpreter, the EEPROM and the resonator) separately at a discounted price for tight integration into your products. Additionally, the BASIC Stamp I editor software includes a feature to program PBASIC1 code directly into a PIC16C58 microcontroller using the Parallax PIC16Cxx programmer. This creates a very low cost option for using the BASIC Stamp I in a production product.

What items do I need to get started with the BASIC Stamp?

The items you need are: 1) the programming software, 2) the programming cable, 3) the manual, 4) the BASIC Stamp module itself, and optionally 5) the appropriate carrier board. If you are interested in using a particular version of the BASIC Stamp, it is best to purchase either the BASIC Stamp I or BASIC Stamp II Starter Kits. These kits include all five of the items listed above at a lower cost than if you were to purchase the entire components separately. If, however, you intend to use both BASIC Stamp I and II, it is best to purchase the BASIC Stamp Programming Package (which include manual, software and cables for both versions of the BASIC Stamp) and then purchase the BASIC Stamp modules and optionally the carrier boards separately.

How can I make my own BASIC Stamp circuit?

The BASIC Stamp chipset are available in two or three different package types from Parallax, DIP, SOIC and SSOP. Any of these can be used to create your own BASIC Stamp circuit for purposes of deeply imbedding it within an application. Refer to the “General Stamp Schematic” near the front of the BASIC Stamp manual for information on the DIP package type BASIC Stamp I chipset. Please call Parallax Technical Support to receive information and pointers on other package types as the pin configurations of the interpreter chips are different and are not properly shown in the 1995/96 Microchip documentation.

What kind of PIC chips are used in the BASIC Stamp?

The BASIC Stamp I uses a PIC16C56 and the BASIC Stamp II uses a PIC16C57.

What kind of operating environment does the BASIC Stamp require?

The BASIC Stamp I or II modules will work in 0° to 70° C temperatures with up to 70%, non-condensing humidity. While the modules may continue to function outside these ranges, it is not guaranteed or recommended by Parallax, Inc. The parts which make up the BASIC Stamps are available from Parallax, Inc. in industrial (0° to 70° C) and extended (0° to 70° C) ranges. Additionally, it is best to keep the BASIC Stamps away from, or shielded from, any nearby RF interference as this may impact the accuracy of its I/O functions.

What are the main differences between the BASIC Stamp I and II?

The BASIC Stamp I has 8 I/O pins, room for 80 to 100 lines of code, executes approximately 2000 instructions per second and requires a parallel interface for programming. The BASIC Stamp II has 16 I/O pins, 2 dedicated serial port pins, room for 500 to 600 lines of code, executes approximately 4000 instructions per second and requires a serial interface for programming. For a more detailed comparison, refer to the language differences page on our web site: http://www.parallaxinc.com/stamps/langdiff.htm and review the BASIC Stamp I to BASIC Stamp II Conversion document (available on the same web page).

Is the BASIC Stamp II better than the BASIC Stamp I?

Yes and no, depending on your application. The BASIC Stamp II has twice the number of I/O pins, twice the execution speed, 5 times the memory (code) space, a serial port and 5 times the resolution of time sensitive commands. In many cases, the BASIC Stamp II fits applications better than the BASIC Stamp I. In some cases, however, this is not true. For example, you may have a need for just a couple of inputs and outputs with which you need to perform relatively simple, non-speed critical tasks. This is a perfect application for the BASIC Stamp I; the BASIC Stamp II would be overkill if used in this way. A slightly different example would be if you have a variable pulse width signal you need to monitor. The BASIC Stamp I can detect and measure a pulse as little as 10 uS wide and as long as 0.65535 seconds wide. The BASIC Stamp II has 5 times the resolution, so it can measure a pulse as little as 2 uS wide, however, only as long as 0.13107 seconds wide. While the better resolution may be helpful, the smaller maximum pulse width detection ability may prove disadvantageous to your application.

How many I/O pins does the BASIC Stamp have?

The BASIC Stamp I has 8 I/O pins while the BASIC Stamp II has 16 I/O pins plus 2 special serial port pins.

What can I use the I/O pins for?

The BASIC Stamp's I/O pins are perfectly suited for digital input and output with TTL/CMOS level (0 to 5 volt) signals. You can, however, use some special commands and techniques to input and output limited analog signals. For example, the RCTIME and PWM commands can be used to read a variable resistance or output a variable voltage from 0 to 5 volts.

Can the I/O pins be used to control relays, solenoids and other similar devices?

Yes, however, due to the demanding current and voltage requirements of some of these components, driver circuitry will need to be used to properly isolate the I/O pins from harmful effects. For examples of this, refer to BASIC Stamp I Application Note #6 and BASIC Stamp Article #6, "Silicon Steroids for Stamps".

Can I control LEDs with the I/O pins?

Yes. Simply use a 470 ohm resister in series with the LED to limit the current draw through the I/O pin (see "How much current can the I/O pins handle?"). Also, keep in mind that most LEDs require a lot of current in relation to what the BASIC Stamp can provide. If you attach and power 3 or more LEDs at one time from the BASIC Stamp's I/O pins, you are likely to see flaky and unpredictable results caused by voltage sag, I/O pin damage and/or hardware resets. Either driver circuitry or low current LEDs will need to be used if you require such an application.

How much current can the I/O pins handle?

Each I/O pin is capable of sourcing 20 mA and sinking 25 mA. The total across all I/O pins should not exceed 40 mA source or 50 mA sink at any given time, however. One exception to this rule exists with the BS2-IC. If the BS2-IC is powered by an external 5-volt regulator capable of delivering at least 100 mA, the total across each group of 8 I/O pins (0 - 7 and 8 - 15) can reach 40 mA source or 50 mA sink providing a total of 80 mA source or 100 mA sink overall.

What is the input impedance of the BASIC Stamp's I/O pins?

Approximately 1 Meg Ohm.

How fast does the BASIC Stamp execute its PBASIC code?

On average, the BASIC Stamp I executes 2000 lines of code per second and the BASIC Stamp II executes 4000 lines of code per second. This is only an estimate as the true execution speed depends upon many factors including, the particular set of instructions used and the type of arguments provided to the instructions. Some instructions execute at a much faster rate while others, especially those that have to deal with external signal measurement or specific protocol speeds, will execute much slower.

What is the VIN pin used for?

The VIN (Voltage Input) pin is used to power the BASIC Stamp from a 6 to 15 volt source. The VIN pin is the positive connection while the VSS pin is the negative, or ground, connection. When powered from the VIN pin the BASIC Stamp regulates the voltage down and outputs +5 volts on the VDD pin.

How does the VDD pin work?

The VDD pin (+5V) outputs 5 volts when the BASIC Stamp is powered by a 6 to 15 volt source using the VIN and VSS (ground) pins. The VDD pin can be used to power other circuitry provided that the overall current consumption is within the capabilities of the BASIC Stamp's regulator.

How does the reset pin (RES) work?

The reset pin is internally controlled by the BASIC Stamp's brown-out detector. It is normally high (+5V), which allows the BASIC Stamp to run its program normally, and is pulled low when the power supply voltage drops below 4 volts, which safely puts the BASIC Stamp to sleep. This pin can be monitored to detect when a reset condition occurs, or, you can pull the line to ground to force a reset. After the reset pin is allowed to raise to +5 volts, the BASIC Stamp wakes up and starts executing its program from the first line of code. Do not drive this pin high, it should be left electrically disconnected (floating) when you want the BASIC Stamp to run normally.

On the BASIC Stamp II, how does the ATN (Attention) pin work?

The ATN pin is used strictly for programming the BASIC Stamp II and should not be connected to anything other than the DTR pin of the programming connector. The ATN pin is an input and has an inverse relationship with the reset (RES) pin; when the ATN pin is left unconnected or is driven low it causes the RES pin to go high (allowing the BASIC Stamp II to function properly) and when the ATN pin is driven high it causes the RES pin to go low (performing a hardware reset on the BASIC Stamp II). The BASIC Stamp II editor software toggles this pin from low to high and then low again at the start of the programming process to perform a hardware reset on the BASIC Stamp II and begin the bi-directional programming communication. It is important to be aware of a particular artifact of the DTR pin on a standard serial port: upon port selection by standard software (terminal packages, development tools, etc.), the DTR pin is driven high by default. If the same connections for programming the BASIC Stamp II are used to communicate with it via a serial port, the default state of the DTR pin will cause the BASIC Stamp II to be held in permanent reset and its PBASIC program will not execute. This problem can be fixed in a number of ways: 1) disconnect the DTR to ATN pin connection when trying to communicate serially to the BASIC Stamp II, or 2) explicitly set the DTR pin to low after selecting the port if the terminal package or development tool allows this, or 3) modify the programming connection to include a 0.1 uF capacitor in series with the DTR to ATN pin connection. The last two options will allow you to use the same connection for both programming and serial communications.

How much current can the BASIC Stamp's on-board regulator provide?

The 5-volt regulator built into the BASIC Stamp can supply 50 mA current if powered by a 12 volt source. The BASIC Stamp I consumes 2 mA in running mode and the BASIC Stamp II consumes 7 mA. This leaves 48 mA and 43 mA respectively for use with other circuitry via the VDD pin and the I/O pins.

Can the BASIC Stamp generate a sine wave?

Yes. Both the BASIC Stamp I and II can generate sine waves. The BASIC Stamp I uses the SOUND command to generate from 94.8 Hz to 10,550 Hz. The BASIC Stamp II uses the FREQOUT command to generate from 0 Hz to 32,767 Hz. Note that the output from the SOUND and FREQOUT commands appear to be static when viewed on an oscilloscope. This is due to the nature of the underlying algorithm which generates the frequency, however, when averaged out over time, the result is a sine wave. You may achieve a cleaner signal by attaching the I/O pin to an appropriate filter circuit.

Can the BASIC Stamp generate a sine wave on two pins at once?

No. The BASIC Stamp is a single-task device and thus will execute only one SOUND or FREQOUT command at a time.

Can the BASIC Stamp generate more than one sine wave on a pin at the same time?

Yes, the BASIC Stamp II's FREQOUT command can generate two frequencies at the same time, on the same pin. This can be used to create harmonics which greatly improve the sound output.

Can the BASIC Stamp generate square waves?

Yes, however, it is limited. No PBASIC function exists to generate an accurately timed square wave, however, it can be done manually as long as fast timing is not an issue. This can be done with multiple lines of code which simply set a pin high or low and pause for the appropriate duration in-between transitions. The timing will vary somewhat depending upon your loop execution speeds.

Does the BASIC Stamp include a real time clock function?

No. If you need to keep track of time, especially dates and time, it is best to interface a real time clock to the BASIC Stamp.. Many are available from various manufacturers. The real time clocks with a serial interface are usually the best choice as they require fewer I/O pins from the BASIC Stamp to function properly.

Does the BASIC Stamp have a built-in timer?

Yes, however, it is used for timing functions within the BASIC Stamp interpreter and there is no direct access to it through any PBASIC commands. Refer to BASIC Stamp I Application Note #20, “An Accurate Timebase” for one possible solution to timing problems.

Does the BASIC Stamp support interrupts?

No, the interpreter chip used in the BASIC Stamp does not support interrupts. In many cases, a fast polling routine may be used to accomplish the same effect, however, depending on the number and size of the tasks involved in some applications, this will not be fast enough and the BASIC Stamp may not be a plausible solution.

Does the BASIC Stamp II have the same bsave/bsload feature available in the BASIC Stamp I?

No. It is intended to provide this feature in the future. Currently the BASIC Stamp II editor will accept the BSAVE directive but will not generate any object code.

How do I program the BASIC Stamp?

You can write your PBASIC program using the BASIC Stamp I or II editor on a standard IBM compatible PC or a Macintosh which is running SoftPC or SoftWindows 2.0. After you write the code for your application, you simply connect the BASIC Stamp to the computer's parallel port (BASIC Stamp I) or serial port (BASIC Stamp II), provide power to the BASIC Stamp and press ALT-R within the appropriate BASIC Stamp editor to download your program into the BASIC Stamp's EEPROM. As soon as the program has been downloaded successfully, it begins executing its new program from the first line of code.

How do I connect the BASIC Stamp to my computer for programming?

The BASIC Stamp I (rev. D or BS1-IC) requires a three-wire connection to any available parallel port on your PC. The BASIC Stamp II (BS2-IC) requires a standard, straight-through, serial cable connection to a 9-pin or 25-pin serial port on your PC. Only 6 wires are used for programming the BASIC Stamp II, 4 of which are absolutely required. You can not use a null-modem cable to program the BASIC Stamp II. Refer to the schematics on our web site for the details of the cable connections, http://www.parallaxinc.com/stamps/stmpsche.htm.

How does my program get stored in the BASIC Stamp?

Your PBASIC code is tokenized (compressed) and stored within the Stamp's EEPROM memory. This memory is non-volatile, meaning it retains it's program even without power. Every time the BASIC Stamp receives power, it starts running its PBASIC code starting with the first executable line. This EEPROM can be rewritten immediately, without any lengthy erase cycle or procedure, by simply pressing ALT-R in the BASIC Stamp editor again. Each location in the EEPROM is guaranteed for 10,000,000 write cycles before it wears out. The code is tokenized before downloading, source code elements like comments and constant and variable names are not stored in the BASIC Stamp, thus you may feel free to use as many comments and descriptive symbol names in your code as you like without worrying about increasing your code size.

How do I erase the BASIC Stamp's program space?

An erase cycle is performed at the beginning of every programming process, thus, by pressing ALT-R in the BASIC Stamp editor the EEPROM is erased and then reprogrammed with the current PBASIC program in the editor.

How do I reprogram the BASIC Stamp?

Simply re-connect it to the computer, run the BASIC Stamp editor and press ALT-R.

How big of a program can I store in the BASIC Stamp?

The BASIC Stamp I has 256 bytes of program storage; enough for 80 to 100 lines of PBASIC1 code. The BASIC Stamp II has 2048 bytes of program storage; enough for 500 to 600 lines of PBASIC2 code.

Can I expand the program memory?

No, the program memory is not expandable as the interpreter chip expects the memory to be a specific, fixed size.

Can I expand the data memory?

Yes, you may interface EEPROMs, or other memory devices, the BASIC Stamp's I/O pins to gain more data storage area. You will have to include the appropriate code within your PBASIC program to interface to the particular device you choose.

How difficult is it to program the BASIC Stamp?

When compared to other programmable microcontrollers, the BASIC Stamp I and II is perhaps the easiest to use because of its simple, but powerful, language structure and straight forward method of downloading and debugging. If you have some experience programming in BASIC, C or Pascal, you should find the learning process with the BASIC Stamp to be a simple one. If, however, you have never had any programming experience, you may have to spend some extra time studying the examples and application notes before you feel comfortable with the BASIC Stamp.

Can I program the BASIC Stamp in Visual BASIC or QBASIC?

No. The BASIC Stamp is a microcontroller not a miniature PC. It does not have a graphical user interface, a hard drive or a lot of RAM. The BASIC Stamp must only be programmed with PBASIC which has been specifically designed to exploit all of the BASIC Stamp's capabilities.

What is PBASIC?

PBASIC (Parallax BASIC) is a hybrid form of the BASIC programming language in which many people are familiar with. Currently there are two versions of PBASIC, PBASIC1 for the BASIC Stamp I and PBASIC2 for the BASIC Stamp II. Each version is specifically tailored to take advantage of the inherent features of the hardware it runs on. PBASIC is called a hybrid because, while it contains some simplified forms of the normal BASIC control constructs, it also has special commands to efficiently control the I/O pins.

Can I imbed assembly language routines in my PBASIC code?

No. The BASIC Stamp is a PBASIC interpreter only and cannot except assembly language routines.

How much space does each PBASIC command take?

Each command takes a variable amount of space within the BASIC Stamp's EEPROM. It varies due to complexities of the command itself and the types of arguments you supply each command. Generally, each command takes 2 to 4 bytes of memory space, however, commands like SERIN, SEROUT, LOOKUP and LOOKDOWN, which accept a variable length list of arguments, may take tens of bytes or more.

How do I know how much space my PBASIC program consumes?

On the BASIC Stamp I, enter the following code at the start of your PBASIC1 program:

```
READ 255,B0
DEBUG #B0
```

Upon running the program, a number will display in the debug window of the editor. Use the following equation to determine how many bytes are used by your PBASIC1 code: 255 - # - 6; where # is the number displayed on the debug window. Note, the "- 6" in the equation results from the fact that the above two lines of code take 6 bytes of program space, thus without those two lines, your program takes 6 fewer bytes of space.

On the BASIC Stamp II, from within the Stamp II editor and with your PBASIC2 program in the screen, press ALT-M to view the memory map screens. The first screen shows your RAM space, the second screen shows an expanded view of the program (EEPROM) space and the third screen shows a detailed view of the program space. Press the spacebar to alternate between the three screens.

Why doesn't the Memory Map feature show me data I have already stored in the BASIC Stamp II's EEPROM?

The memory map feature (ALT-M) shows only what the memory space will look like immediately after you program the Stamp II. The editor determines this by examining your PBASIC source code within the editor, not the PBASIC code in the Stamp. It does not read the BASIC Stamp's EEPROM. It will only display data in the EEPROM that you've explicitly defined with DATA commands; no data which is operated upon by READ and WRITE commands can be shown since those are run-time, rather than compile-time, functions.

How do I debug my PBASIC programs?

You may use two features of the BASIC Stamp editor to debug your PBASIC program: 1) Syntax checking, and 2) the DEBUG command. Syntax checking alerts you to any syntactical errors and is automatically performed on your code the moment you try to download to the BASIC Stamp. Any syntax errors will cause the download process to abort and the editor will display an error message with a highlight cursor pointing to the source of the error in your code. Logical errors are sometimes more difficult to find but may be more easily discovered by using the DEBUG command. Debug operates similar to the PRINT command in the BASIC language and can be used to print the current status of specific variables within your PBASIC program as it is executed within the BASIC stamp. If you PBASIC code includes a DEBUG command, the editor opens up a special window at the end of the download process to display the results for you.

How do I print out my PBASIC programs?

The BASIC Stamp editors do not have a print option built-in, however, the source file created by the editor is simply a DOS text file. This means it can be printed with the standard PRINT or TYPE commands as in:

```
PRINT source.bas

    -or-

TYPE  source.bas > LPT1
```

where source.bas is the name of your PBASIC program.

Can I read-out the PBASIC program which is already stored in the BASIC Stamp?

No. For security reasons this feature was not made available. It is possible to read the tokenized form of the PBASIC code out of the BASIC Stamp's EEPROM, but there is no easy way to "reassemble" it into usable source code.

Since the BASIC Stamp II uses the serial port for programming can I use a simple terminal program to download my PBASIC code?

No. The download process requires a very fast, bi-directional communication between the BASIC Stamp and PC to program and verify the contents of the BASIC Stamp's memory.

Will a program and circuitry designed for the BASIC Stamp I work with the BASIC Stamp II?

Yes, however, some changes, usually in the PBASIC code only, will have to be made for it to run properly or to take advantage of some of the new features of the BASIC Stamp II. For help in performing these modifications or for insight into how the PBASIC1 language differs from PBASIC2, download the "BASIC Stamp I to BASIC Stamp II Conversions" document from our web site: http://www.parallaxinc.com.

Can I program the BASIC Stamp II with the BASIC Stamp I editor (stamp.exe)?

No. The BASIC Stamp II requires the BASIC Stamp II editor (stamp2.exe) due to the enhancements in the PBASIC2 language and the serial, rather than parallel, programming interface. Similarly, you cannot program the BASIC Stamp I with the BASIC Stamp II editor (stamp2.exe).

How do I make my PBASIC program start over or continue running forever?

All BASIC programs will simply end if one or more of their executable paths don't lead them to a continuous loop. In the simplest case, a program can be made to continue endlessly by adding a label (a unique name followed by a colon ':') to the start of code and adding a GOTO statement at the end of code directing logical execution back to that label. Without this, the BASIC Stamp will run the program only once and then will remain unresponsive until the power is cycled or a reset condition is created.

How do I set an I/O pin to input or output mode?

There are many ways to set the I/O pin directions. The most readable and intuitive method is to use the INPUT and OUTPUT commands. For example:

```
INPUT 5
OUTPUT 6
```

will set I/O pin 5 to an input and I/O pin 6 to an output. By default, upon reset or startup, all I/O pins are inputs, thus only the second line of code above is really necessary.

Another method for setting the pin directions is to use the predefined variable DIRS or any one of its derivatives. The following code is functionally equivalent to the first example:

```
DIR5 = 0
DIR6 = 1
```

DIR0 through DIR15 (or DIR7 on the BASIC Stamp I) are bit variables which control the direction of the corresponding I/O pin. DIRS is a 16-bit variable (or 8-bit variable on the BASIC Stamp I) in which each bit represents the direction of the corresponding I/O pin. A '1' bit means output and a '0' bit means input. The advantage of this method of declaring pin directions is that multiple pin directions can be defined at once:

```
DIRS = %00101110
```

The above line of code defines I/O pins 7, 6, 4 and 0 as inputs and 5, 3, 2 and 1 as outputs. Review the BASIC Stamp manual for more examples of this.

How do I make an I/O pin output a high or a low?

As with setting I/O pin directions, there are many ways to set the I/O pin states. The most readable and intuitive method is to use the HIGH and LOW commands. For example:

```
HIGH 5
LOW 6
```

will first set I/O pin 5 to output mode and then set it to a high (+5V) state. The next line will first set I/O pin 6 to output mode and then set it to a low (0V) state. Notice that these commands automatically set the pin direction for you; it’s not necessary to manually set the pins to output when using the HIGH and LOW commands. Keep in mind that if a pin is set to input, its logical state, high or low, remains the same internally within the BASIC Stamp but will no longer affect the voltage on the I/O pin.

Another method for setting the pin state is to use the predefined variable OUTS (or PINS in the BASIC Stamp I) or any one of its derivatives. The following code is functionally equivalent to the first example:

```
DIR5 = 1
DIR6 = 1
OUT5 = 1
OUT6 = 0
```

Notice here we had to explicitly define the I/O pins as outputs first. If we had left the pins as inputs, the OUT5 and OUT6 lines would have changed the state of the pins internally, but those changes would not appear externally from the BASIC Stamp. OUT0 through OUT15 (or PIN0 through PIN7 on the BASIC Stamp I) are bit variables which control the logical state of the corresponding I/O pin. OUTS is a 16-bit variable (PINS is an 8-bit variable on the BASIC Stamp I) in which each bit represents the state of the corresponding I/O pin. A ‘1’ bit means high and a ‘0’ bit means low. The advantage of this method of declaring pin states is that multiple pin states can be defined at once:

```
DIRS = %00101110
OUTS = %00001010
```

The above line of code defines I/O pins 7, 6, 4 and 0 as inputs and 5, 3, 2 and 1 as outputs. And sets I/O pins 7, 6, 5, 4, 2 and 0 to logical 0’s and I/O pins 3 and 1 to logical 1’s. Review the BASIC Stamp manual for more examples of this.

How do I input or output data in parallel on the I/O pins?

Both BASIC Stamps have predefined variables which refer to the I/O pins. On the BASIC Stamp I, the PINS variable refers to all eight I/O pins. Thus code such as:

```
SYMBOL  Snapshot = B0
Snapshot = PINS
```

would copy the current state of all I/O pins (an 8-bit byte) into the variable Snapshot. Similarly:

```
DIRS  = %11111111           'Set all eight pins as outputs
PINS = 150
```

would set the eight output pins to the binary form of 150, or %10010110.

On the BASIC Stamp II, the PINS variable was replaced with the INS and OUTS variables. Both INS and OUTS refer to all sixteen pins, while their derivatives, INL, INH, OUTL and OUTH refer to the low and high bytes (groups of 8 pins). For example, the following code will input 8-bits in parallel from the lower 8 pins:

```
Snapshot     VAR     BYTE
Snapshot = INL
```

The following code will output 8-bits in parallel from the upper 8 pins:

```
DIRS = %1111111100000000         'Set upper 8 pins to outputs
OUTH = 255
```

Why do the BASIC Stamp's output pins toggle briefly when I use the SLEEP, NAP or END commands or allow my program to end naturally?

Inside the BASIC Stamp's interpreter chip is a watchdog timer whose main purpose is to reset the interpreter chip if, for some reason, it should stop functioning properly. The SLEEP and NAP commands also utilize the watchdog timer to periodically (every 2.3 seconds to be exact) "wake-up" the BASIC Stamp from its low-power mode. Upon reset the I/O pins are set to inputs for approximately 18 mS before returning to their previous directions and states. If you have an output pin set to a logical 1 state (+5V) and you use the SLEEP command, every 2.3 seconds during sleep mode that I/O pin will switch to an input for 18 mS causing a momentary signal loss. This "power glitch" is easily viewable with an LED and a 470 ohm resister tied to an I/O pin and switched on just before entering sleep mode. In many cases this problem can be remedied by tying a pull-up or pull-down resistor to the I/O pin in question to provide a constant source of power should the I/O pin change directions. Allowing a PBASIC program to end naturally, or using the END command, will exhibit the same "power glitch" behavior as the interpreter chip enters a low-power state. The BASIC Stamp II includes the STOP command which acts just like END, except it does not enter the low-power mode and the output pin remain driven.

The LET command is not available on the BASIC Stamp II. Does this mean the BASIC Stamp II cannot perform mathematical operations?

No. The LET command on the BASIC Stamp I was truly optional; i.e.: you could leave it out of the variable assignment statement or equation and it would work normally. To save some space in the interpreter code of the BASIC Stamp II, the LET command was removed completely, however, the mathematical ability was not removed and, in fact, it is greatly improved on the BASIC Stamp II.

Can the BASIC Stamp store data, such as temperature readings, for review or use at a later time?

Yes. The READ and WRITE commands allow the BASIC Stamp to manipulate memory locations in its EEPROM. Any space in the built-in EEPROM which is not occupied by PBASIC code may be used for data storage and is just as non-volatile as the PBASIC code itself. You may also attach external EEPROMs of various sizes to gain an even greater data storage area. The READ and WRITE commands will not function on external EEPROMs so you must write the appropriate code to interface with the device.

How can I store a word value into the internal EEPROM?

The internal EEPROM of the BASIC Stamp (used for both program and data storage) is organized and accessed in units of bytes only. Since a word value is actually two bytes it requires two consecutive memory locations to store it in EEPROM. You must break the word value into its lower and upper byte parts and use two WRITE commands to store the value, or two READ commands to retrieve it.

The following code demonstrates this for the BASIC Stamp I:

```
W0 = 1250
WRITE 0, B0
WRITE 1, B1
```

The following code demonstrates this for the BASIC Stamp II:

```
Temp VAR WORD

Temp = 1250
WRITE 0, Temp.LOWBYTE
WRITE 1, Temp.HIGHBYTE
```

How does the fpin (flow control pin) work in the SERIN and SEROUT commands on the BASIC Stamp II?

The optional fpin argument in the SERIN and SEROUT commands allows the use of an I/O pin as a hardware flow control line between two BASIC Stamps IIs at baud rates of up to 19.2 kB. The flow control pin is always controlled by the receiving device (SERIN) and monitored by the sending device (SEROUT). The logical state of this pin and its meaning depend on the data mode, true or inverted, as specified within the baudmode parameter. If true data mode is selected, the receiving BASIC Stamp II holds the fpin low to indicate it is ready to receive or high to indicate it is not ready to receive. If inverted data mode is selected, the receiving BASIC Stamp II holds the fpin high to indicate it is ready to receive or low to indicate it is not ready to receive.

Assuming true data mode: the fpin is set to output high the moment the SERIN command is first encountered and remains high until the SERIN command is complete at which point the fpin is set to low and remains there until the SERIN command is executed again. The transmitting BASIC Stamp II's SEROUT command monitors this line and halts transmission whenever a low on the fpin is detected.

How are arithmetic expressions evaluated within the BASIC Stamp?

All expressions are evaluated using 16-bit, integer arithmetic. Even if byte and bit variables are used in expressions they are first expanded to 16-bits and then the expression is evaluated.

In the BASIC Stamp I, mathematical expressions are evaluated strictly from left to right. No order-of-precedence is utilized, parentheses are not allowed and expressions must be on a line by themselves; i.e.: they can not be used as an argument for a command. For example, the expression: W0 = 10 - 2 * 3 results in the value 24 being stored into W0 (not the value 4 as you might think). To evaluate this expression properly, specify it as: W0 = 0 - 2 * 3 + 10.

In the BASIC Stamp II, mathematical expressions are evaluated from left to right, as in the BASIC Stamp I. No order-of-precedence is utilized, however, unlike the BASIC Stamp I, parentheses are allowed and expressions may be used as arguments within commands. For example, the expression: Answer = 10 - 2 * 3 results in the value 24 being stored into Answer, however: Answer = 10 - (2 * 3) results in the value 4.

Does the BASIC Stamp handle signed numbers and arithmetic?

Yes. The BASIC Stamp uses twos-compliment notation for signed numbers. This means that the expression: 0 - 10 + 5 will result in -5 if viewed as a signed number, however, most instructions see the number as a positive value, in this case 65531 (the twos-compliment value for -5). All mathematical operators, except division, will work properly with signed numbers in the BASIC Stamp and signed numbers can be formatted for output properly using the SDEC, SHEX and SBIN formatters within DEBUG and SEROUT statements on the BASIC Stamp II. Be careful how you use signed numbers elsewhere. For example, if the value -5 is stored in a variable called Temp, and you use the following statement:

```
IF Temp < 0 THEN Loop
```

it will evaluate to false and will not branch to Loop because -5 is actually 65531 in twos-compliment form and thus 65531 is not less than 0.

How can I define an alias to an I/O pin or another variable?

In the BASIC Stamp I you could specify the following:

```
SYMBOL Counter      =  B0
SYMBOL Index        =  B0
SYMBOL LED          =  PIN0
```

to designate the symbol Index as an alias to the Counter variable and the symbol LED as an alias to I/O pin 0. Since Counter and Index use the same register, B0, anytime Counter is changed, Index will change in the same way. The symbol LED will always contain either a 1 or a 0 depending on the logical state of I/O pin 0.

In the BASIC Stamp II you could specify the following:

```
Counter     VAR   BYTE
Index       VAR   Counter
LED         VAR   IN0
```

to designate the symbol Index as an alias to the Counter variable and the symbol LED as an alias to I/O pin 0. Since Index uses the same memory space as Counter, anytime Counter is changed, Index will change in the same way. The symbol LED will always contain either a 1 or a 0 depending on the logical state of I/O pin 0.

How do I reference a specific bit within a byte or word variable?

On the BASIC Stamp I there is only one general purpose word register, W0, and two general purpose byte registers, B0 and B1, which are bit addressable. The predefined symbols Bit0 through Bit15 refer to the corresponding bits within W0 as well as B0 and B1 since those two byte variables correspond to the lower and upper bytes of W0 respectively. Thus Bit0 is the lowest bit of both W0 and B0 while Bit8 is the lowest bit of B1 (and the 9th bit of W0).

On the BASIC Stamp II all the variables are bit addressable, as well as nibble addressable. You can reference a specific part of a variable by specifying the variable name followed by a dot '.' and then the name of the portion you're interested in. For example, bit 2 of a word variable called Temp can be referenced with the notation: Temp.BIT2. Additionally, the second nibble of Temp can be referenced with: Temp.NIB1. The valid modifiers are: BIT0 through BIT15, LOWBIT, HIGHBIT, NIB0 through NIB3, LOWNIB, HIGHNIB, BYTE0 through BYTE3, LOWBYTE and HIGHBYTE.

How do I define a string variable?

A string variable is simply an array of bytes. Only the BASIC Stamp II allows the explicit definition of arrays. If, for example, you want to define a string variable called Text to hold 6 characters, use the following code:

```
Text VAR    BYTE(6)
```

The elements of an array can be accessed by specifying the array name followed by an open parenthesis, an index value and a close parenthesis. The first element is always index 0. For example, the second element, or character in this case, can be referenced with: Text(1) and the last element with: Text(5). Byte arrays are useful for receiving a string of characters via the SERIN command using the STR modifier. The BASIC Stamp II allows you to define arrays of bits, nibbles and words as well.

How do I define aliases to specific bytes within a word array on the BASIC Stamp II?

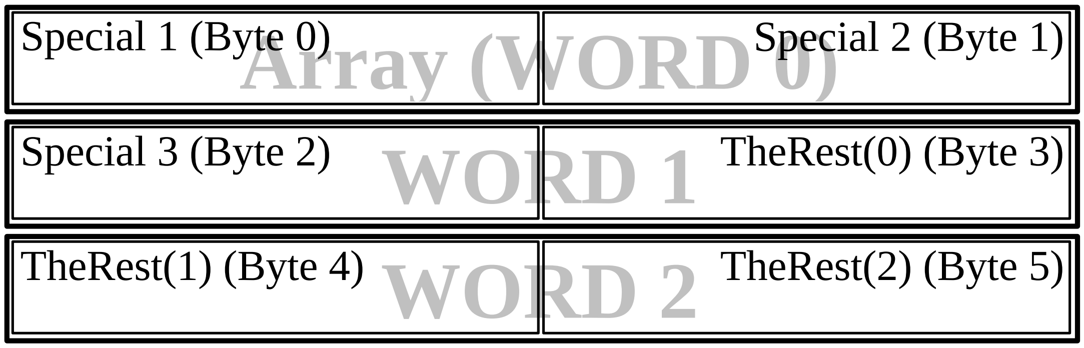
The BASIC Stamp II uses a very relaxed and flexible indexing scheme when it comes to arrays of RAM space. When trying to define aliases in this way it is best to view the entire RAM space as one array of word, byte, nibble and bit registers. The BASIC Stamp II organizes variables in the RAM space from largest entities (word declarations) to smallest entities (bit declarations), regardless of the order you define them, to be most efficient. The relaxed nature of indexing allows a user to cross array boundaries at will, or by accident, quite easily. For example, the following code defines two distinct byte variables:

```
Result        VAR   BYTE
Temp          VAR   BYTE
```

Normally you would reference the Result and Temp variables by name, as in: Result = 10. Even though this variable is not an array, you can still access it as such. The code: Result(0) = 10 is equivalent to the previous example. The most interesting, and powerful, feature is that you can index beyond the boundaries of this byte variable, as in Result(1) = 20. This statement actually modifies the byte of RAM immediately following the Result variable. The next byte happens to have been defined as Temp, in this case, thus the variable Temp will now equal 20. By carefully defining variables we can take advantage of this flexible feature to define byte sized aliases to specific locations within a word array. Suppose we have the need for an array of three words in which the first three bytes have a special meaning; we'd like to be able to easily manipulate those bytes as well as access the three distinct word elements.

```
Array         VAR WORD
Special1      VAR Array.LOWBYTE
Special2      VAR Array.HIGHBYTE
Special3      VAR BYTE
TheRest       VAR BYTE(3)
```

Remember: the BASIC Stamp II will always organize RAM space from biggest elements to smallest. In the example above we defined a single word variable called array (this becomes assigned to the very first word in RAM space, word 0). The next two definitions create aliases to the first and second bytes of Array (no additional RAM space is allocated here since we're pointing at previously allocated space, byte 0 and byte 1). Next we define a single byte variable (this becomes assigned to the first byte of RAM immediately following the Array variable, byte 2). Finally, we define an array of three bytes (which are assigned to the next three bytes of RAM immediately following the previous byte declaration, byte 3, byte 4 and byte 5). Visually, our RAM space now appears like this:



Now, Array(0), Array(1) and Array(2) refer to the three words of our array and Special1, Special2 and Special3 refer to the first three bytes within our three byte array.