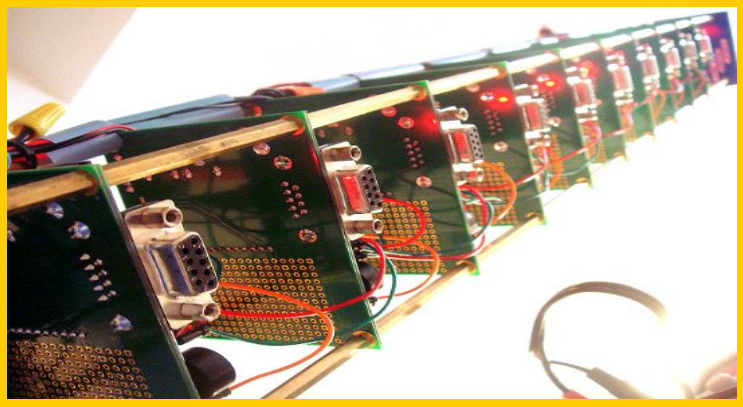
How to Build & Have Fun
with Intelligent Parallax Stamp Supercomputers

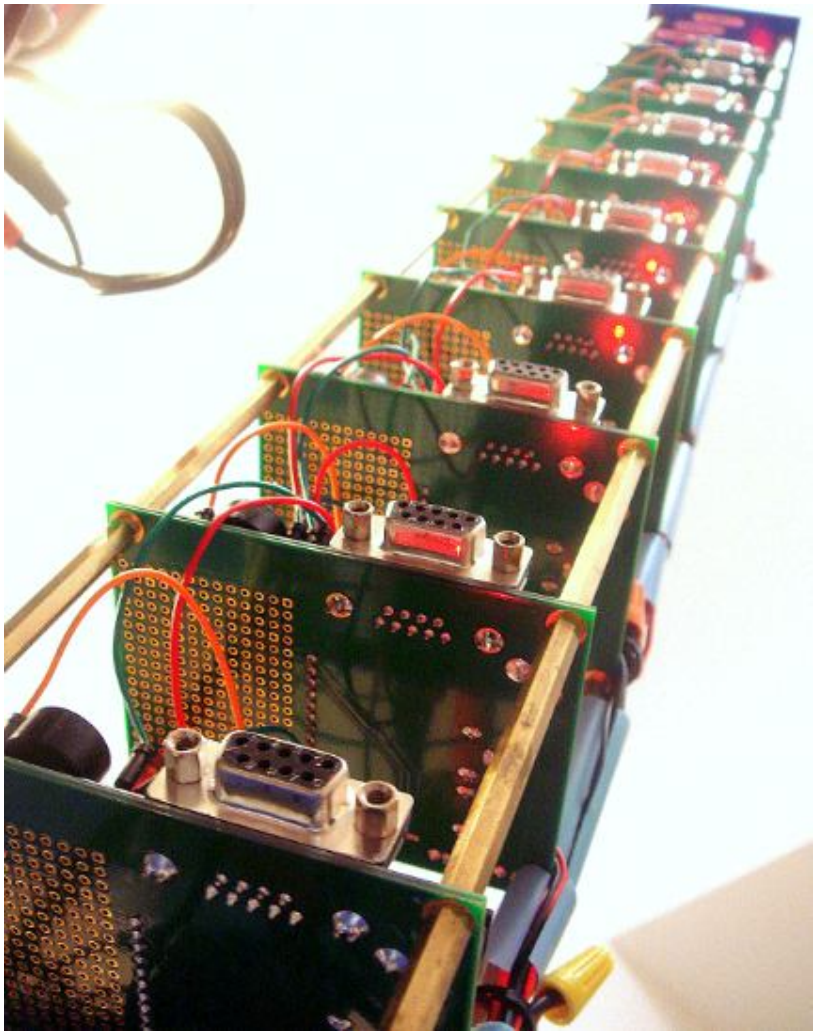
Handbook of Basic Stamp Supercomputing



Supercomputers Included: SEED, TriCore, Minuscule
Supplied with Artificial Intelligence Software
Create Living Basic Stamp Supercomputers

by Dr. Humanoido

CREATIVE PARALLAX PROJECTS



Turn up the spotlight – drum roll, lights, action, camera! The world's first powerful Stamp SEED Supercomputer is born! *SEED'*s features include the evolutionary-revolutionary new Stamp AI Artificial Intelligence software, new *Skyscraper* design, easy construction, extremely low power consumption, new wiring techniques, and very low cost. Of course, it's ten cores over a single Stamp!

This book introduces three new creative Stamp supercomputer projects and numerous applications, tips, references and collections of vital and useful information for all Basic Stamp supercomputer enthusiasts. Included are schematics, build instructions and complete program listings for testing, various methods of communication, and simplified Artificial Intelligence. Here you'll find many idea innovations for "rolling your own."

1) The Stamp Seed Supercomputer SSS, code name Baby, is a ten core collection of BS1 boards. It's power is in the software – the same code that loads to each core. From there, each program becomes a life form and evolves.

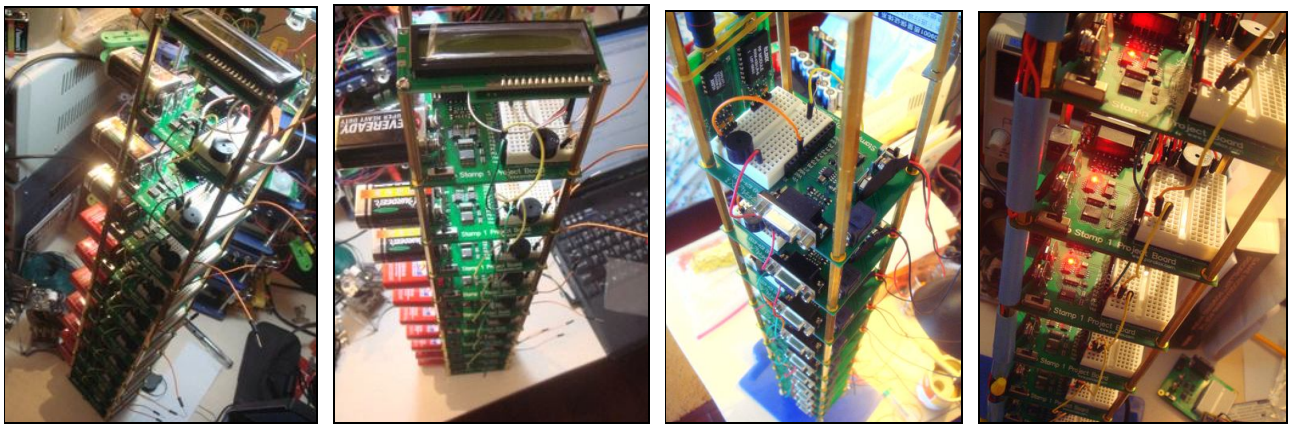
2) The Stamp TriCore Supercomputer is a three core collection of BS1 boards, used to test the SSS AI. It's fully capable of running a special version of the SEED AI software. See posting sources for downloads.

3) The Minuscule Stamp Supercomputer MSS is a smallest Stamp supercomputer in terms of the minimum number of cores. It's purpose is initial testing of the BS1 wiring, design, function, serial communication, and Piezo Language *PLAN* , in a simple easy to use package. There is no SEED software written for the two-core MSS, though it's suggested as an exercise.

Build information about hardware and software is included to replicate and create your own stamp supercomputers. These new generation stamp processing machines are a spinoff of the original Basic Stamp Supercomputer. For more information about the BSS and related machines, consult the resource listing near the end of this book.

Table of Contents

evolutionary new design: THE 1st Basic Stamp Supercomputer with life form programming, self evolving enumerating deterministic, a quantum leap in programming Basic Stamp Supercomputers! Create your own multiple AI life forms and go farther than you ever dreamed possible...



Top Left: The building blocks of life begin with the tower - a Skyscraper made up of brass spacers supporting ten Stamp project boards. Right: blue wire wraps are made up from soda pop straws. Note position of the piezo speakers that overhang the breadboards edge to gain maximum use of wire connection points. This shows test wiring only.

Stamp SEED Supercomputer! All new! This is a ten core, over ten month evolving project, with the 1st Stamp AI software to fit into 256 bytes eeprom - self determinate, evolving, enumerating, dreaming, poetic, noisy, talkative, and downright friendly (consult various software versions). It runs on only one program that self evolves differently in each of the ten computers. It's evolutionary - it's revolutionary!
http://forums.parallax.com/forums/default.aspx?f=21&m=361377&p=1

The Stamp SEED supercomputer is born at a time when the **Parallax Basic Stamp hobby supercomputers** are becoming popular due to the value of a multiple processor core design, a simple way to construct these units with one wire, and the ability to program in a simple but powerful language PBASIC. Connecting together two, ten, or dozens of Basic Stamp Processors creates many advantages by amplifying the power over a single processor. These hand-made cores are the gateway to new inventions, education and computing progress. Code name Baby, this supercomputer includes piezo speakers, batteries, portability, power LEDs, individual board switches, and real estate for wiring, sensors and projects. It uses the famous BS1 stamps as a ten core processing machine, and it includes a Parallax 422Mhz wireless radio frequency communications feature. The project works out details using the Basic Stamp 1 for artificial intelligence (self aware, enumerating, deterministic) parallel clustered processing, serial Rx and Tx, one wire interface, and wireless radio communications for talk with other Basic Stamp Supercomputers or individual radio-equipped Stamps.

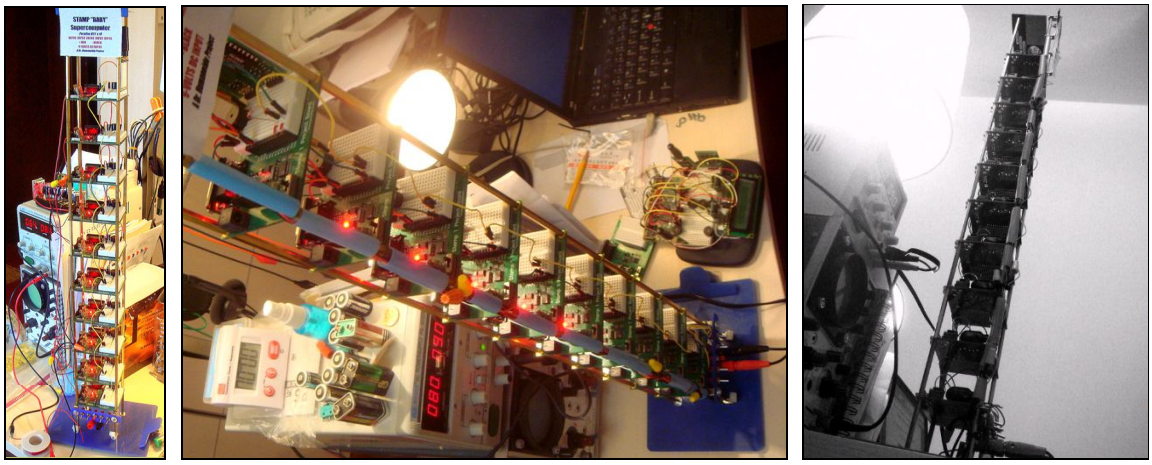
Why Build a Stamp Supercomputer? The objective of creating this project is fun and education: learn more about multiple AI life forms and their interactions, supercomputers with the BS1 (principles of supercomputing, clustering hardware and parallel programming, Artificial Intelligence AI, one evolving program that becomes ten life forms, birthing and self aware code, etc.), and to serve as a platform for some very interesting experiments and demonstration projects. The *Stamp Seed Supercomputer* SSS could be the basis of new technology, experiments in new life forms and artificial intelligence, and an educational classroom or a *Science Fair dream*. The *SEED* can become an Adjunct to more Stamp supercomputers or any single computer. The name conventions, *Stamp SEED Supercomputer* and the *Basic Stamp Supercomputer* are super-stamps, and represent a name convention. They are unique full scale working Basic Stamp supercomputers in context which is a comparison of their single Stamp cores to the full array, and not a comparison to high speed supercomputers outside of the Basic Stamp genre.

About the Stamp BS1 Parallax Company was founded on the Basic Stamp One in the 1990s. Since that time, they have introduced many Basic Stamp 2 models and the Propeller chip. Much of knowledge and information about the BS1 is archived, set aside, and sometimes forgotten. This project reinstates the power of the BS1 and revives the status of its powerful advantages. It is proven there are many gurus on the Parallax Forum who remain masters of this processor. The power and magnitude of the BS1 is often underestimated, even misunderstood. Many BS2 and

Propeller applications can run well with a BS1 platform. It's time to haul out the modern version of the BS1 (see the new Stamp 1 Project Board) and get started with this fascinating project!

List of Features The SEED has a 10-part piezo sound cluster, a control panel with switches, ten power red LEDs, a miniature pushbutton keyboard for input, wireless communications interface (syncs with other computers), a serial interface driven by batteries or AC power supply, a Base Station, and various apps such as LCD for output, and a DS1620 digital thermometer.

- Ten Parallax BS1 Processors using Stamp One Project Board Construction
- New Self Aware AI Software (SEED*) Self Evolving Enumerating Deterministic*
- Dedicated Enumerating Hardware Pins
- Artificial Intelligence Enabled Utilizing *Miniscule AI*
- Bi-directional Wireless Radio Frequency Communications at 433Mhz
- Field Battery Portable or Bench Power Supply Operable
- Ten Cores of Clustered Paralleled Processing, Ten Programs Run Simultaneously
- 80 ports for Circuits, Controls, Sensors & I/O Devices
- New Skyscraper Tower Interlocking Design
- Ten Piezo Speakers, Ten Power Red LEDs
- Ten Solderless Breadboards Real Estate Plot
- Full Instrumentation Control Panel, One Pushbutton, Four Toggle Switches
- Extremely Low Power Requirements
- Software Programs in PBASIC, Hardware Nibble Programmable



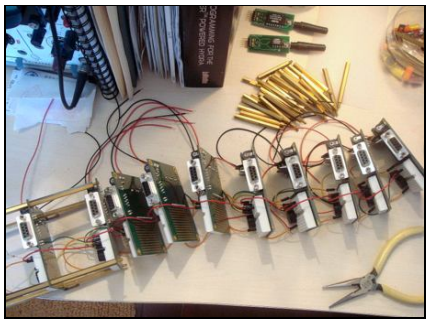
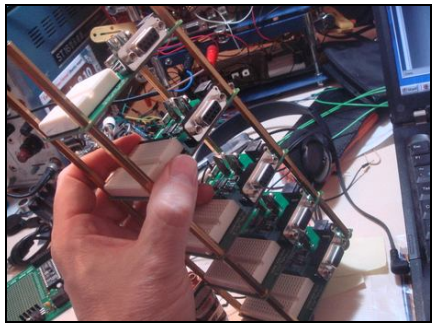
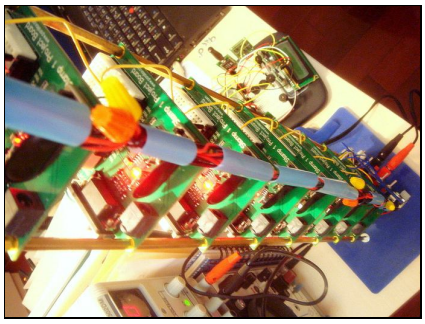
Left- Primed with 10 batteries and ten cores, the new paralleled network array Stamp Baby Super-computer (the development name) is ready for "first power" testing. Boards were checked before clustered wiring with pins P0.

Middle- the Skyscraper design is made up of brass spacers. A top level cage protects two transceiver boards up to the full height of the antennae. The completed Skyscraper has ten levels and a blue crown molding which reads in English and Chinese with the project's construction site location and a logo. As seen on the screen of the bench power supply, all ten boards draw a mere 80 ma at 9-volts. Note the use of ten piezo speakers used for testing and apps.

Right- this 1st version tower was rebuilt for more computer cubical spacing to make wiring considerably easier.

Configuration & Computing Power

Comparing Stamps to Stamps, the novel Baby has a powerful cluster of ten BS1s, each in a BS1 Project Board that amplifies the power of a single stamp 1 by a factor of ten. It creates a super Basic Stamp 1. The port count goes upward from 8 to 80. Speed gains escalate from 4 to 40Mhz. Instruction code capacity in EEPROM grows from 80 to 800. Software is transformed from a single linear program to 10 programs evolving simultaneously. Program execution speed increases from 2,000 IPS to 20,000 IPS. Breadboard real estate goes up ten times. The addition of both software and hardware programming adds great flexibility. In general, the computing power is in the number of ports, new capabilities of life form software, hardware clustering, and parallel processing - not by the speed of conventional supercomputing, though it has ten times the speed of a single hobby processor.

Left & Middle- constructing the Skyscraper Tower is a piece of cake. Just bolt together all these spacers from board to board. Note how smaller spacers were rejected and larger ones substituted in place during construction. A larger space in between boards was needed to reach in and rewire circuits.

Right- this view shows the left front wiring harness which uses free soda pop straws given away by a local grocery store promotion. These make good wire clamps along the side rail on the Skyscraper assembly. In the background is another Stamp project which is being tested as a base station for all Basic Stamp Supercomputers.

Overall Design The design of the Baby Stamp Supercomputer is significantly different from the Basic Stamp Supercomputer BSS.

1. Has unique Artificial Intelligence Programming
2. All ten boards power up at the same time
3. Does not use a single master, each board is an individual, unique with personal freedom
4. The network is a contributory party line with simple rules
5. 433Mhz Wireless Transmitter and Receiver for full communications
6. No programmable LEDs
7. All BS1 construction throughout
8. Serial baud top speed is 2400 bps
9. Can hardware program a 4-bit Nibble through toggle switches
10. The demo is accomplished with sound piezo speakers
11. Simplified wiring, concealed wire harness
12. Faster and easier to build, plug-n-play
13. Lighter in weight, smaller footprint
14. No English or Chinese speech board
15. Speaks Piezo Language with upgrade
16. Ultra low power consumption
17. Ultra low voltage mode
18. New interlocking durable tower design
19. Lowered center of gravity increases stability
20. No board clips necessary, new soda straw design wire clamps
21. The 1st Stamp Supercomputer with a supporting Base Station Bstat
22. Include one program that evolves as a life form in all computers
23. Self deterministic
24. Special pin wiring for Life Form Evolution

Design Similarities to the BSS:

1. Each Board has a Programmable Piezo Speaker
2. Utilizes the One-wire Interface
3. Functions with Clustered Software and Paralleled Programming
4. Handles Massive Numbers of Sensors
5. Each Computer is Easily Rewired
6. Battery or Power Supply Operable
7. Portable
8. Large Array of Solderless Breadboard Real Estate
9. Control Panel Functions

Beginning construction of the Stamp Baby Supercomputer shows a new interlocking Tower design. This uses minimal parts, is simple to construct, and is dirt cheap using low cost brass surplus spacers from the surplus center. Each stamp board supports its adjacent neighbor with the interconnecting skyscraper design.

The tower is extremely strong, as proved by dropping it (a test accident) from a table top to a hardwood floor, a distance of four feet with no harm whatsoever. (Do not try this at home or otherwise!). To prevent topples, a large foundation base was added, made from a clipboard. On the right side, the clip can hold incoming power cables during supercomputing or when hosting other cabling.

Left: the Skyscraper is intact with its new spacing, banner, and 9-volt battery clips in place, ready for wiring the main cable. This becomes the primary power distribution wiring harness.

Center: blue soda pop straws make perfect wiring clamps after they are given a parallel incision.

Right: the wire harness is tested and awaiting the wire twists for completion. In all photos, a half size blue clipboard base provides the greatest stability.

Skyscraper Tower Construction

The Skyscraper tower for the SEED supercomputer is a new unique interlocking design. It is built up from a minimum number of simple parts at minimal cost. Primarily, it consists of spacers, which interlock to boards. There are no extra parts. The assembled unit consists of interlocking 6cm-long brass spacers with rigidity maintained by evenly spaced Stamp boards. This *skyscraper effect* is extremely strong. The foundation is an optional half size polymer clipboard with the clip anchoring the power supply cord on the right side. The instrument control panel is made from a similar clipboard clip.

The Skyscraper Crown is a scavenged clip from a recycled clipboard. The Chinese symbols translate as "Flying." Ironically, Beijing 2008 is where and when the project had a flying start!

The Skyscraper design is extremely strong. The board spacing is ideal to reach in and wire the real estate or make future modifications, add components and sensors. A cage at the top of the Skyscraper is made up of two fiberboard panels and extended brass spacers to protect the transmitter and receiver antennae and to support the logo and project identification. The cage is open on the top to RF, facilitating transceiver transmissions and reception. Wire clips made from soda pop straws are attached to spacers, run along the left front side – the purpose is to organize and hold wires. At the bottom, spacers are terminated with four nuts fitted to the base clipboard. The bottom of the foundation has six bumper feet taken from Stamp 1 Project Boards. The SEED supercomputer Skyscraper is about 2 ½-feet high, and has the same width and depth of a Stamp 1 Project Board (3.25-inches wide x 2.5-inches deep).

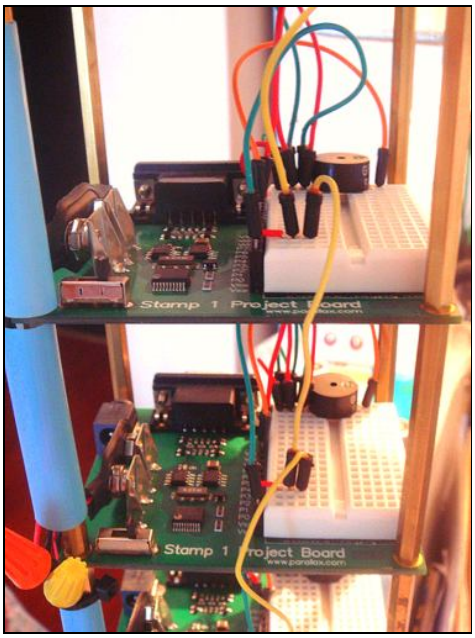
Skyscraper Personal Space
With a little personifying, each computer board has rectangular personal space with dimensions of 3¼-inches wide x 2½-inches deep x $2^3/8$ inches high. No other Stamp is allowed to invade this *privacy cubical*. Through experimentation, this was determined as the best space for reaching in to do wiring and connecting larger sensors, yet it maintains a minimal size for the overall Skyscraper.

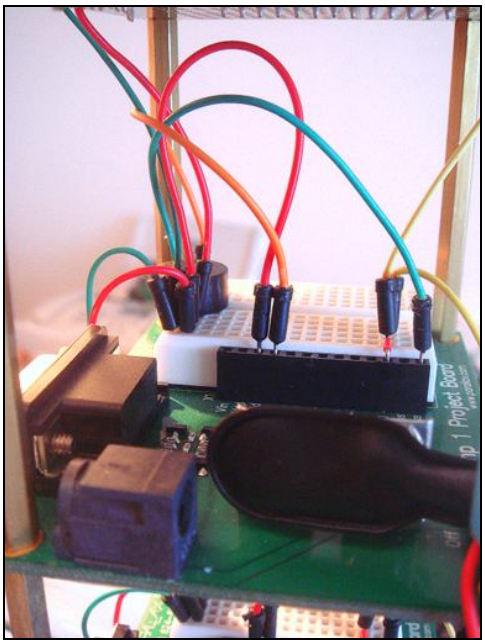
Protective Ground Field - EMI/RFI Shielding
"Electromagnetic interference (or EMI, also called radio frequency interference or RFI) is a disturbance that affects an electrical circuit due to either electromagnetic conduction or electromagnetic radiation emitted from an external source. The disturbance may interrupt, obstruct, or otherwise degrade or limit the effective performance of the circuit. The source may be any object, artificial or natural, that carries rapidly changing electrical currents, such as an electrical circuit, the Sun or the Northern Lights." Reference: Wikipedia

Supercomputers are more susceptible to EMI and RFI, stray electrical interference, static electricity. The electromagnetic waves generated by electronic devices may negatively affect other, similar, electronic devices. Such affects are called Electromagnetic Interference (EMI) and Radio Frequency Interference (RFI). EMI and RFI may cause malfunctions in electronic devices. Specifically, EMI and RFI cause suppression of signals generated internally in a device, external ambient interference with equipment operation, or emissions generated internally that will interfere with equipment operation.

The world today is increasingly electronic, with millions of waves and signals floating through the air at any given moment. Therefore, EMI and RFI are potential problems in any industry, and there is a premium on protective products and services from it. In order to prevent EMI or RFI, EMI/RFI Shielding agents may be used as protection. Reference: ThomasNet, More about EMI/RFI Shielding

The SEED supercomputer has an invisible grounding field that surrounds the unit. This includes all four sides with the bottom and top left open. The top is non-shielded for the radio frequency transceiver transmission and reception. The invisible shield construct is made from a conductive cage consisting of up to 48 brass spacers. Each set of four spacers is electrically grounded to a computer board, which feeds into the next level. Each side spacer screws into the next, making an electrical conductive connection. Each side spacer makes an additional electrical grounding connection with the trace surrounding the computers "through-board" hole. This creates a 2½ foot high grounding field without using any wires. The DC wiring harness runs along the left length of the supports and is field grounded by this structure. Additional grounding (assurance grounding) is provided in computer to computer routing of Vss wires.

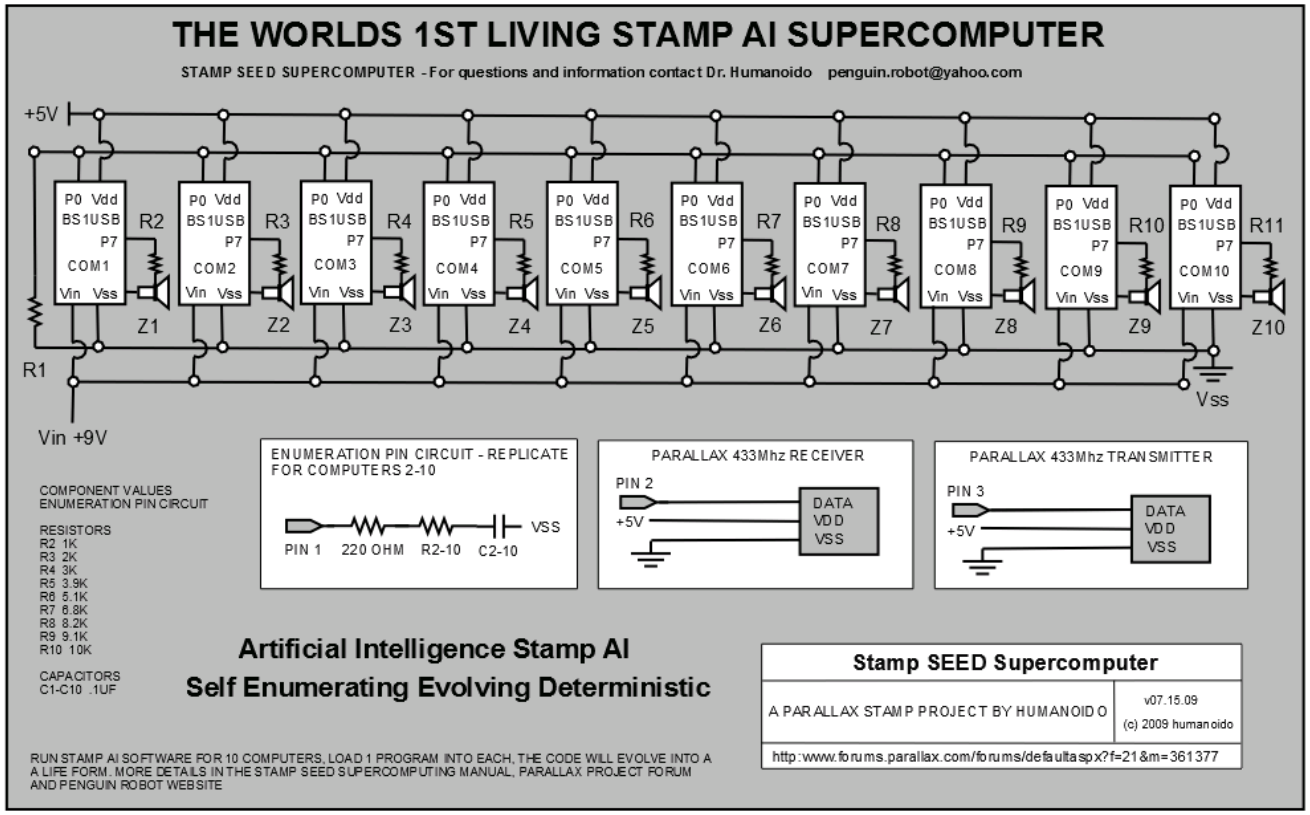




Above- this shows standard computer wiring (excluding top and bottom, 1 and 10, computers). Yellow is from the one wire interface p0. Green is Vss ground and red is +5V Vdd. Red, green, and yellow wires extend to the next computer.

SSS Schematics

The schematic for the Stamp Seed Supercomputer is illustrated below. The SSS is the first Stamp supercomputer to have self enumerating pins, show in the insert. Also shown are components to make up the transceiver using Parallax modules. Boards are numbered 1 at the bottom of the Skyscraper to 10 at the top. Boards one through 9 use three ports each, one for the interface, one for enumeration, and the other for the piezo speaker. Computer ten uses two more pins for the wireless radio frequency transceiver. The receiver is on pin 2 and the radio transmitter is on pin 3. The speaker does not need a resistor. A 220 ohm resistor is built in to each Project Board port. The schematic design uses only one resistor for interfacing.



The schematic shows the Stamp AI supercomputer with all ten BS1 cores. These are actually Stamp 1 Project Boards for convenience of prototyping projects and testing various configurations. The ten core array Stamp supercomputer has ten Piezo speakers, a wireless transceiver and special enumeration pins to enable Artificial Intelligence software at a simple demonstration level.

Building Enumeration Circuits

Construct the Enumeration Circuit as indicated in the above schematic, for each computer. The shown 220 ohm resistor is built into the board already. Consult the table below for the value of R2 through R10. The ceramic disc capacitor value of C2 through C10 is .1uf. For example, computer 4 requires a 3K ohm resistor to built its Enumeration Circuit.

```
' TYPICAL PIN 1 RESISTANCE CODE CHART
' Computer 1       220 ohm (built in to Stamp 1 Project Board)
' Computer 2       220 + 1K Ohm   = 01.22K
' Computer 3       220 + 2K Ohm   = 02.22
' Computer 4       220 + 3K Ohm   = 03.22
' Computer 5       220 + 3.9K Ohm = 04.12
' Computer 6       220 + 5.1K Ohm = 05.32
' Computer 7       220 + 6.8K Ohm = 07.02
' Computer 8       220 + 8.2K Ohm = 08.42
' Computer 9       220 + 9.1K Ohm = 09.32
' Computer 10      220 + 10 K Ohm = 10.22
```

Decoupling Capacitors
On the power side (not shown in the above schematic), the use of decoupling capacitors is recommended for all Basic Stamp Supercomputers. Choose a .1uf capacitor and connect from the power source circuit input Vss to Vdd. The function of decoupling is discussed in detail in numerous places.

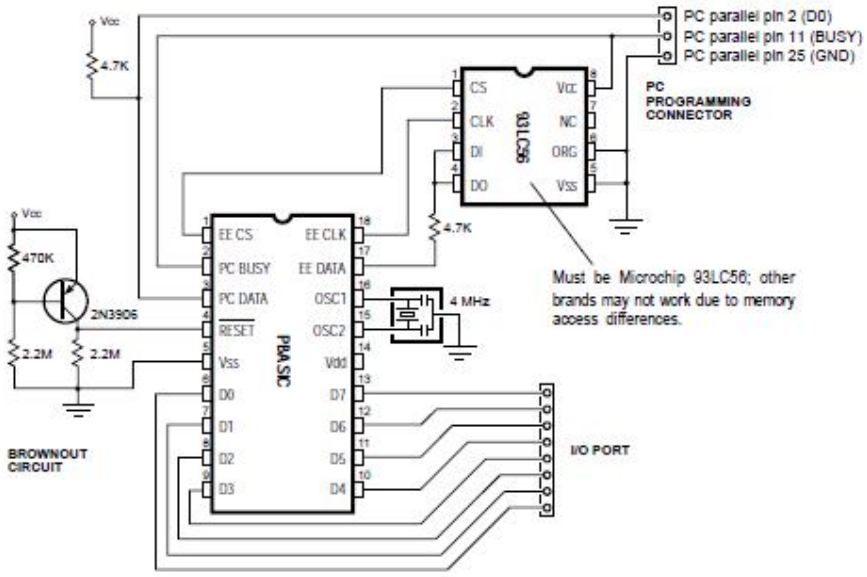
en.wikipedia.org/wiki/Decoupling_capacitor

www.google.com/#hl=en&q=bypass+capacitors&aq=f&oq=&aqi=&fp=gxFYct4ZzkE

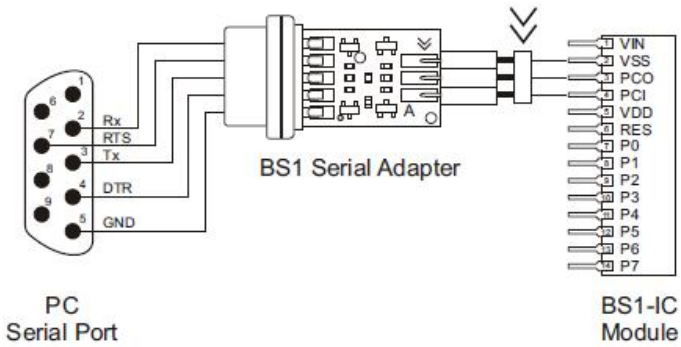
www.seattlerobotics.org/Encoder/jun97/basics.html

www.intersil.com/data/an/an1325.pdf

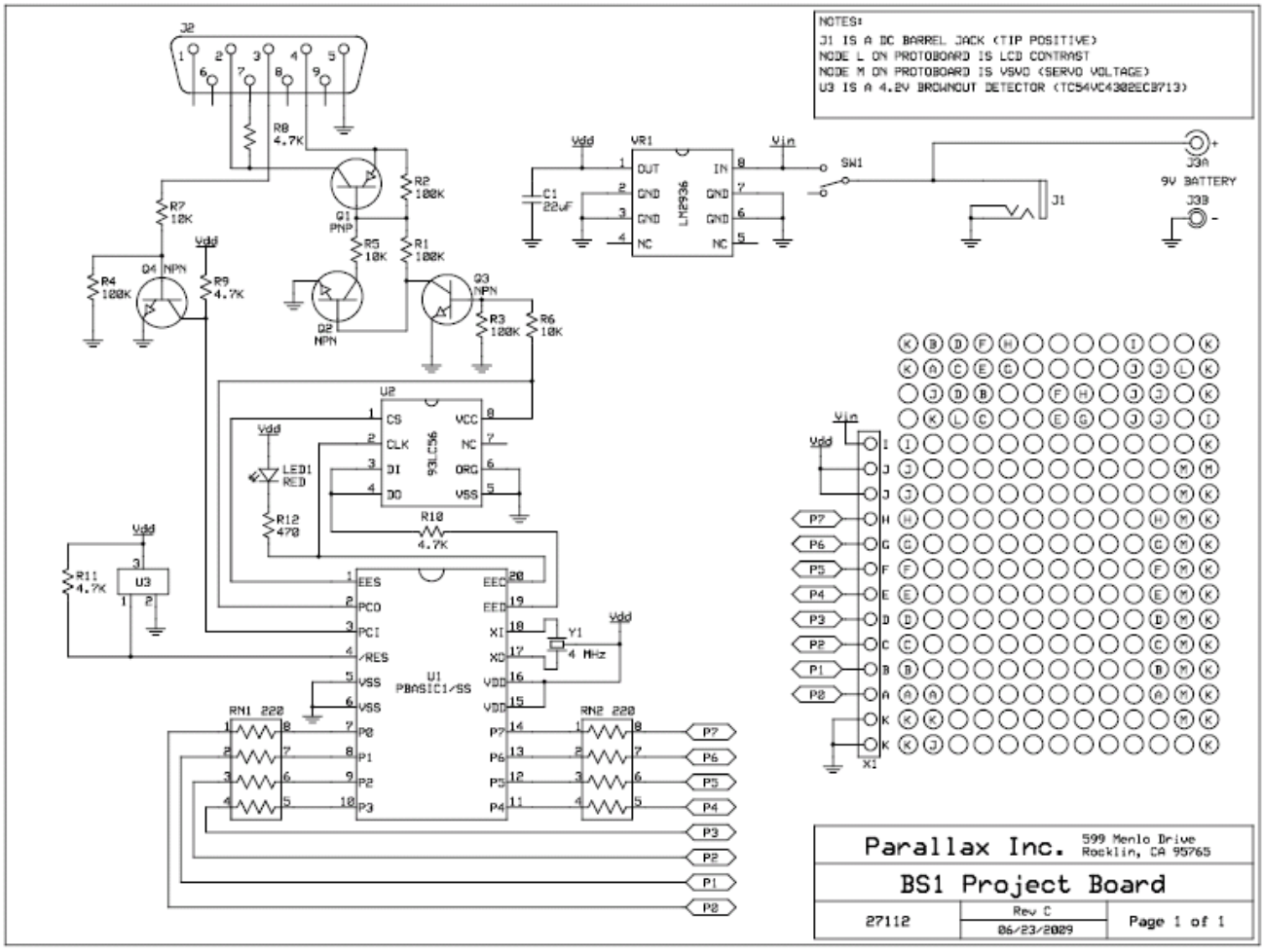
Additional BS1 Schematics
Follow these schematics for various BS1 designs and references.



BS1 without a Brown out Detector: It's important to study the overall simplicity of the BS1 Stamp design. The brownout circuit can be omitted. This drawing does not show the voltage regulator.



Above: Connection to a serial port is made with a BS1 Serial Adapter. The connector is keyed for proper connection.



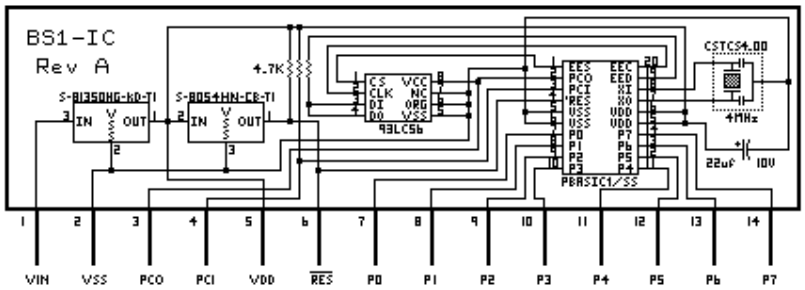
Stamp 1 Project Board Schematic rev C. U3 Brownout Detector is removed.

BS1-IC rev. a

Complete BASIC Stamp circuit in SMT

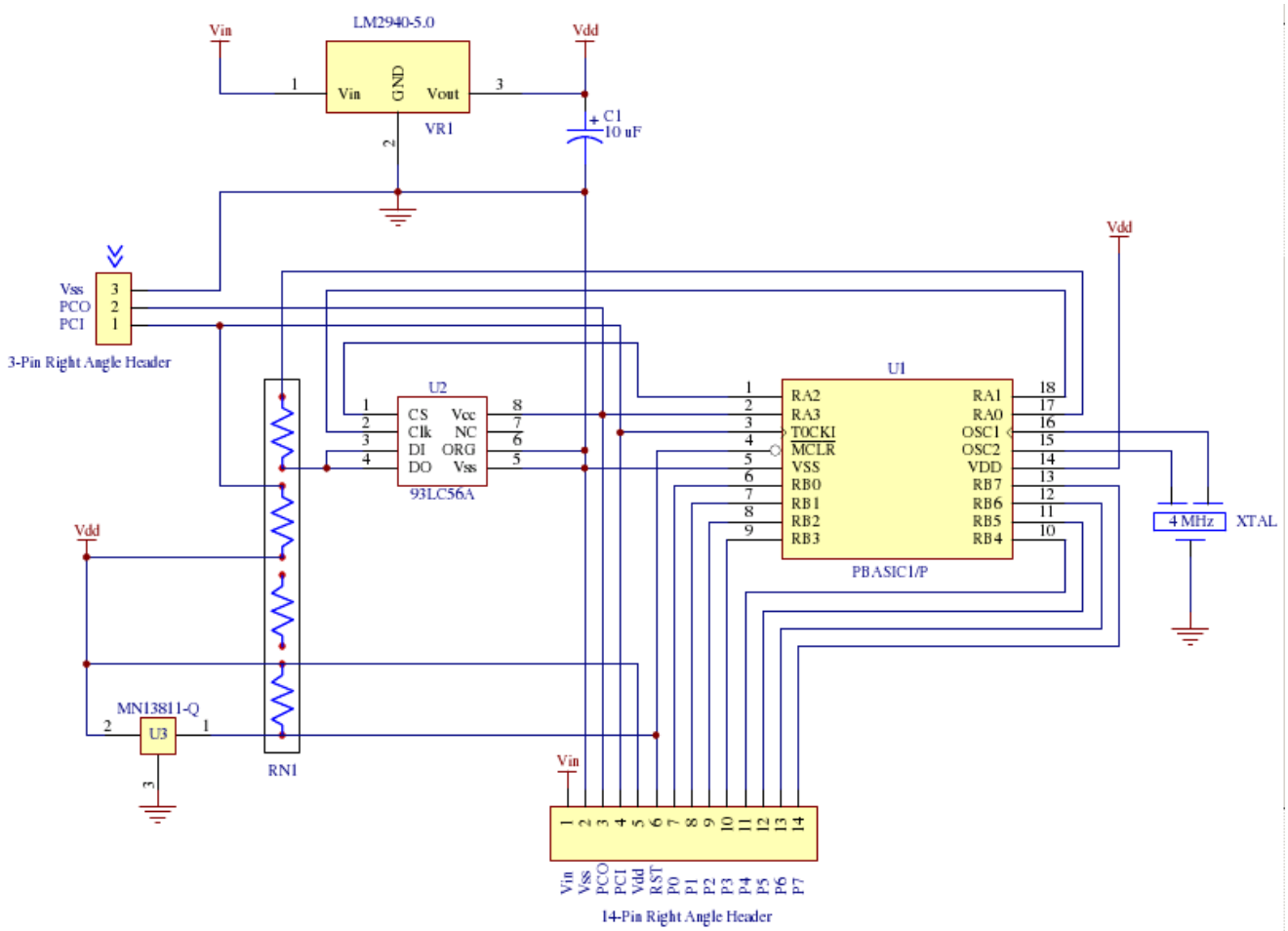
Features

- PBASIC 1.4 Interpreter Chip
- 256-byte EEPROM
- 4MHz Resonator
- 5V Regulator
- 4V Brown-Out Reset
- PC Interface
- Reset Pin
- 8 User I/O Pins
- 1.4ma Run / 40 uA Sleep (no loads, I/O's @ VSS / VDD)

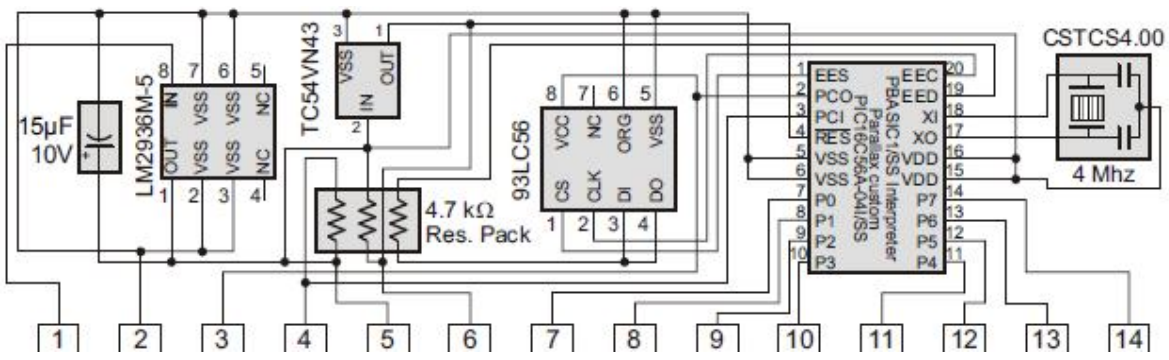


PIN	NAME	FUNCTION	DESCRIPTION
1	VIN	REGULATOR IN	Input to 5V regulator. Accepts up to 15V. If power is applied directly to VDD, pin may be left unconnected.
2	VSS	GROUND	Connects to PC Parallel Port pin 25 (GND) for programming.
3	PCO	PC OUT	Connects to PC Parallel Port pin 11 (BUSY) for programming.
4	PCI	PC IN	Connects to PC Parallel Port pin 2 (D0) for programming.
			These pins are arranged to connect straight-across to a 3-pin header used by the standard BASIC Stamp programming cable.
5	VDD	REGULATOR OUT POWER IN	Output from 5V regulator (VIN powered). Can source up to 50ma, including P0 - P7 loads. Power input (VIN not powered). Accepts 4.5V - 5.5V. Load is dependent upon run / sleep mode and P0 - P7 sourcing.
6	$\overline{RES}$	RESET I/O	Goes low when VDD is less than 4V. Pulled high by a 4.7K resistor. May be monitored as a brown-out / reset indicator. Can be pulled low externally (i.e. button to GND) to force a reset. Do not drive high.
7	P0	USER I / O 0	User port pins that can be configured as inputs or outputs. In output mode: Pins can source up to 20ma each. The total current sourced by all pins should not exceed 40ma. Pins can sink up to 25ma each. The total current sunk by all pins should not exceed 50ma. In input mode: Pins are floating (1ua leakage) and can be driven high or low. The 0 / 1 threshold is approximately 1.4V. NOTE: To realize low power during sleep, make sure that no pins are floating, causing erratic power drain. Either drive them to VSS or VDD, or make them outputs (that don't have to source current).
8	P1	USER I / O 1	
9	P2	USER I / O 2	
10	P3	USER I / O 3	
11	P4	USER I / O 4	
12	P5	USER I / O 5	
13	P6	USER I / O 6	
14	P7	USER I / O 7	

The SIP module rev. A details. The design includes a 4-Volt brownout detector. If the batteries go below 4 volts, expect the Stamp to completely reset.



The OEM BS1 Stamp Rev. A, drawn March 6th, 2001, by John Barrowman / Parallax.



The BS1 IC Module schematic Rev B is show above, updated on 10-5-04, with comments below.
The 15µF, 10V capacitor may be a 10-22µF, 6.3-16V tantalum capacitor.

The 93LC56 EEPROM may be a 93LC56A, 93LC66 or 93LC66A.

The 4 MHz resonator is not polarity sensitive and the middle pin can be connected to either VDD or VSS.

The PBASIC/SS Interpreter chip may be a commercial PIC16C56A-04/SS or an industrial PIC16C56A-04I/SS

Brownout Detector

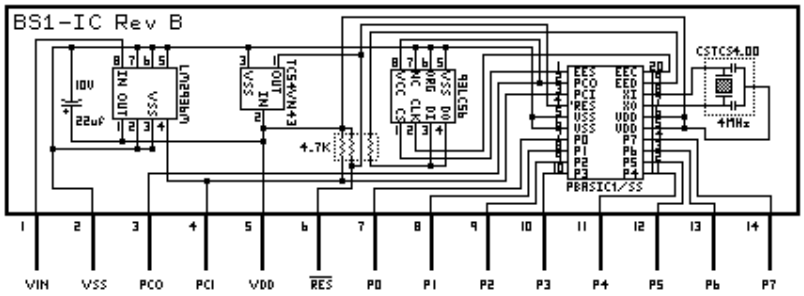
Note when the U3 brownout detector is removed, the supercomputer may function down to around 2 volts. This is very beneficial for extending the life span of a 9-volt battery and keeping all ten life forms alive with extended life. However, when using 5-volt sensors, allow for the differences in voltage.

BS1-IC rev. b

Complete BASIC Stamp circuit in SMT

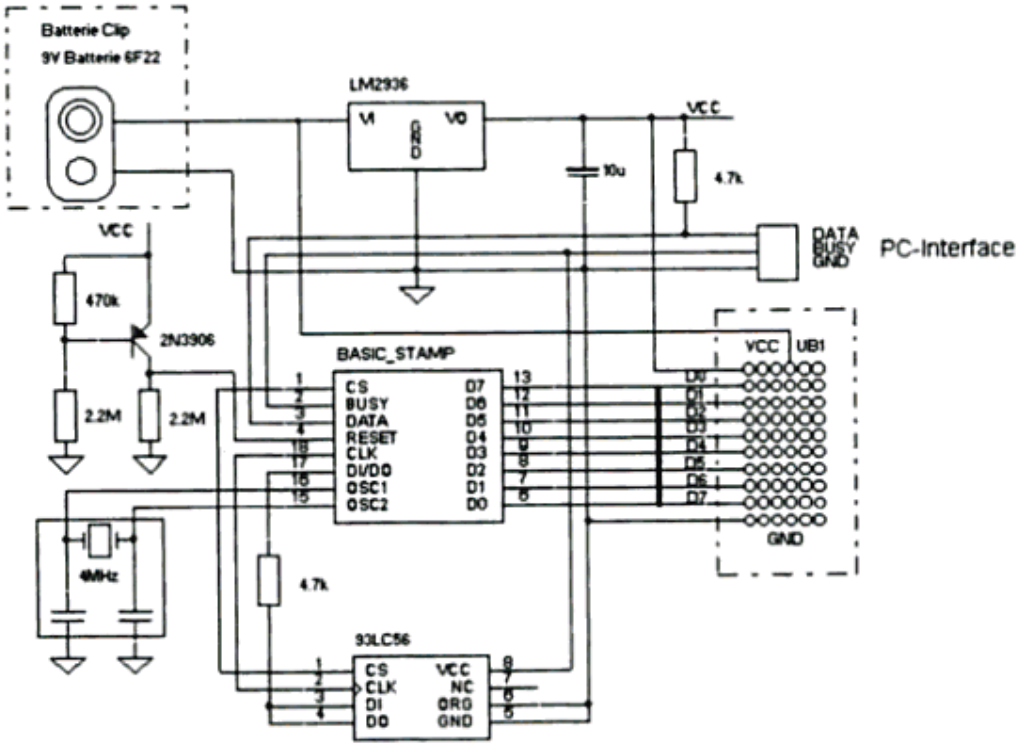
Features

- PBASIC 1.4 Interpreter Chip
- 256-byte EEPROM
- 4MHz Resonator
- 5V Regulator
- 4V Brown-Out Reset
- PC Parallel Interface
- Reset Pin
- 8 User I/O Pins
- 1.4ma Run / 40 uA Sleep (no loads, I/O's @ VSS / VDD)

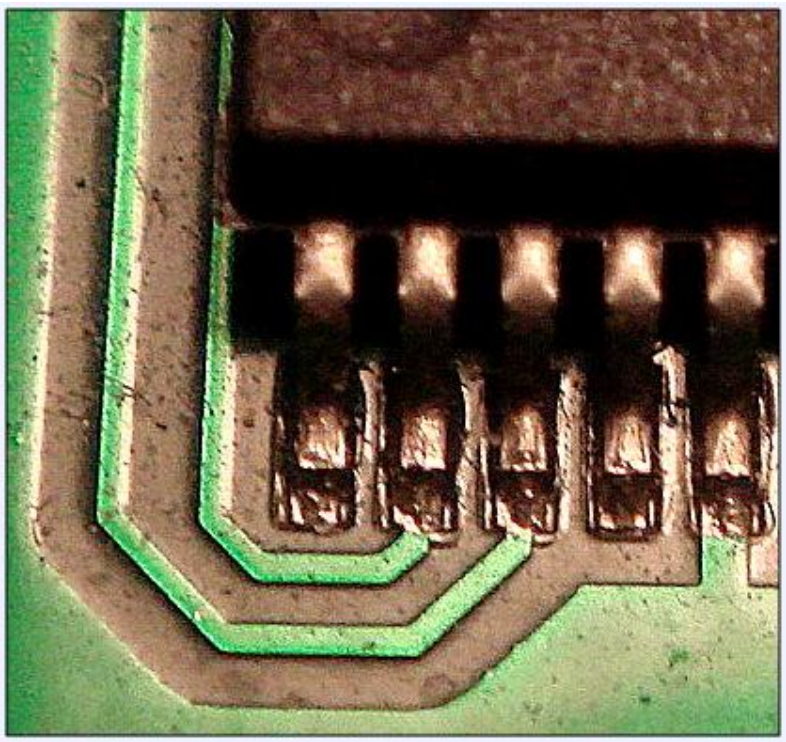


PIN	NAME	FUNCTION	DESCRIPTION
1	VIN	REGULATOR IN	Input to 5V regulator. Accepts 5V to 40V. If power is applied directly to VDD, pin may be left unconnected.
2	VSS	GROUND	Connects to PC Parallel Port pin 25 (GND) for programming.
3	PC0	PC OUT	Connects to PC Parallel Port pin 11 (BUSY) for programming.
4	PC1	PC IN	Connects to PC Parallel Port pin 2 (D0) for programming.
			These pins are arranged to connect straight-across to a 3-pin header used by the standard BASIC Stamp programming cable.
5	VDD	REGULATOR OUT POWER IN	Output from 5V regulator (VIN powered). Can source up to 50ma, including P0 - P7 loads. Power input (VIN not powered). Accepts 4.5V - 5.5V. Load is dependent upon run / sleep mode and P0 - P7 sourcing.
6	$\overline{RES}$	RESET I/O	Goes low when VDD is less than 4V. Pulled high by a 4.7K resistor. May be monitored as a brown-out / reset indicator. Can be pulled low externally (i.e. button to GND) to force a reset. Do not drive high.
7	P0	USER I / O 0	User port pins that can be configured as inputs or outputs. In output mode: Pins can source up to 20ma each. The total current sourced by all pins should not exceed 40ma. Pins can sink up to 25ma each. The total current sunk by all pins should not exceed 50ma. In input mode: Pins are floating (1ua leakage) and can be driven high or low. The 0 / 1 threshold is approximately 1.4V. NOTE: To realize low power during sleep, make sure that no pins are floating, causing erratic power drain. Either drive them to VSS or VDD, or make them outputs (that don't have to source current).
8	P1	USER I / O 1	
9	P2	USER I / O 2	
10	P3	USER I / O 3	
11	P4	USER I / O 4	
12	P5	USER I / O 5	
13	P6	USER I / O 6	
14	P7	USER I / O 7	

Above- in SIP form, the BS1 rev. b is shown with pin details.



The BS1 version D board has this schematic.



Check the chip pins in reference to the schematic to see if a *Brown Out Detector* is present or not. If not, the stamp boards will run on a battery that goes as low as 2-volts and additional apps are possible! Microscope photo by Humanoido.

Wiring

Wiring is made efficient with connection routes running along the front, back and left sides. *Wiring leads* from one computer to the adjacent computer are kept very short and are majority-contained in each computer's personal space. This minimizes wire clutter and facilitates proper Cocooning (see Cocooning in a later chapter). In addition, a wiring harness runs along the front left rail, concealing most of the power distribution (Vin) wires. This uses up to five wire straps and eight soda straw wire clamps, cut to size, directly on the rail. Boards with two connections to port 0 use a small jumper lead taken from a solderless breadboard wiring kit.

Supercomputer Assembly Instructions

Basic Stamp 1 Project boards are connected with spacer hardware. No other fasteners or clips are needed. The Master board is located top-most. A serial display attaches above the master for testing and is later removed. Wire all the Vss connections together. Wire all the Vdd connections together. Attach P0 port lines per schematic. Install the Piezo speakers, one lead to ground and the other to P1 on each computer. During assembly with each step, refer to the various photos and illustrations.

( ) Collect together all parts & supplies and line up organized on a table top
( ) Connect ten Stamp 1 Project Boards together with spacers to form a Skyscraper (see photos)
( ) Disassemble one clipboard and use the plastic clip for the Control Panel (remove burs)
( ) Drill Control Panel with holes for 2 mounting bolts, five switches and two banana jacks
( ) Install switches, banana jacks, and connect the panel using two angle brackets & hardware
( ) Drill clipboard base to accept the Skyscraper base at the end spacer junctions
( ) Mount Skyscraper to base with spacers, washers and bolts, insert two angle iron in the front with cable clamps
( ) Assemble Cage at top of structure, two spacers on each side
( ) Add angle iron to the top of the front spacers
( ) Cut out cardboard to fit the cage top and front. Printout the banner and tape to the front of the cage
( ) Install the battery clips onto Stamp 1 Project Boards
( ) Wire together, in parallel with wire twists, all reds together, all blacks together
( ) From the battery clip wiring, connect a lead red wire long enough to reach the black banana jack
( ) From the battery clip wiring, connect a lead black wire long enough to reach the red banana jack
( ) Warning!!! Battery clip red wire is negative, black wire is positive, opposite of standard color code
( ) Attach soda straw wire clamp/concealing agents to *front left* side rail wiring harness (see text)
( ) Attach five wire straps if necessary to hold the battery clips wiring in place
( ) From the back, install all piezo speakers, overhanging each to maximize real estate
( ) Refer to the Program Listing Schematics for details to assemble the Deterministic Circuit on pins 1
( ) Install a red short jumper lead from P0 to breadboard on computers 2 through 10 (see photos)
( ) Install a green short jumper lead from Vss to breadboard on computer 10
( ) Attach 1K ohm resistor from Vss to P0 on computer 10
( ) Lead orange wire from + piezo to P7 on all boards
( ) Attach green wire to – piezo from board to board
( ) Attach red wire from board to board as shown
( ) Attach yellow wire from P0 on bottom board to every board
( ) Make 5 wire leads with a push pin on one side, and solder end on the other
( ) Solder wire leads to 5 switches on Control Panel

( ) Solder wire leads from wiring harness to banana jacks on Control Panel
Note: Red banana jack goes to the battery clip cables plus side. Black banana jack goes to the battery clip cables minus/ground side.
( ) Connect switches using wire leads to Computer 1. Toggles go to P1, P2, P3, P4. Pushbutton goes to P5.
( ) From the non-pin side of each switch, connect to a resistor (820 ohm), connect resistor to Vss
( ) Install and wire the Wireless Radio Transmitter on Computer 10
( ) Install and wire the Wireless Radio Receiver on Computer 10
( ) Do not turn on the supercomputer until testing is complete
( ) Cut out/attach fiberboard to fit the top and front cage (the cage consists of the top-most four spacers)
( ) Print out the banner and affix it with tape to the cage top front

Testing
( ) Check and verify all wiring with the schematic
( ) Set slide switch to "on" for all boards
( ) Connect a bench lab power supply, switch on (9V DC) and confirm that all ten computers have their red LED on
( ) Load and run designated test programs in each computer
( ) Test the piezo speakers, pushbutton, four toggle switches, transmitter and receiver
( ) You are now ready to begin Basic Stamp Supercomputing!

Parts List & Cost Guide (2009) - Basic Stamp SEED

10	#27112 Parallax Stamp 1 Project Board (HVW Technologies $14.95)	$149.50
60	Jumper Wire, 4-inch (black, yellow, green, orange)	$ 5.00
9	Jumper Lead, from a solderless breadboard wiring kit .02	$ .18
40	2.5-inch Brass Spacers $.10	$ 4.00
10	Battery Clips, 9-V $.02	$ .20
4	Wire Twists $.05	$ .20
4	Spacer Cage Bolts, Washers	$ .16
4	Spacer Base Nuts, Washers	$ .16
2	Control Panel Clamp Bolts, Nuts, Washers .02	$ .12
1	R1 Resistor 1K ohm (pin 0 to ground on C10) .02	$ .02
8	Angle Brackets .10	$ .80
16	Washers .02	$ .32
1	SW5 Pushbutton	$ .30
1	Clipboard (half size) for Base .50	$ .50
2	Nylon Cable Clamps .02	$ .04
4	Wire Straps .05	$ .20
3	Soda Straw Large Blue	n/c
1	Rubber Band	n/c
2	Fiberboard Section	n/c
6	Rubber Bumpers (leftover from Stamp 1 Project Boards)	n/a
1	Printed Banner (see text)	n/a
1	Clipboard Clip for Skyscraper Crown	n/a

Control Panel

4	SW1–SW4 Toggle Switch .30	$ 1.20
1	SW5 Pushbutton Switch .30	$ .30
2	Angle Brackets 90 deg., washers, bolts, nuts	$ .40
5	Jumper Wire 6½-inch length yellow pin on one end	$ .10
5	Resistor 1K for Switches .02	$ .10
2	Power Lead Wire, red, black, stranded	$ .04
1	Resistor 10K ohm for Pushbutton .02	$ .02
4	Resistor 820K ohm for Toggle Switches .02	$ .08
2	Banana Jack Red, black .10 ea.	$ .20
2	Banana Plug Red, black .10 ea.	$ .20
1	AC Power Line Cord (to make power cable) n/a	
1	Clipboard Clip for Instrumentation Control Panel n/a	

Deterministic RC Circuits

9	Capacitor .1uf	$ .02
1 ea	Resistors (0, 1K, 2K, 3K, 3.9K, 5.1K, 6.8K, 8.2K, 9.1K, 10K ohms) .02ea	$ .18

Peripherals

1	#27981 Parallax 433Mhz RF Receiver	$ 39.99
1	#27980 Parallax 433Mhz RF Transmitter	$ 29.99
10	#900-00001 Parallax Piezo Speakers 5v $1.95	$ 19.50
1	Parallax 2x16 Serial LCD (Non-Backlit) # 27976 (Optional)	
1	2x16 Parallel LCD (Non-Backlit) #603-00006 (Optional)	

Optional Base Station Bstat

1	#27112 Parallax Stamp 1 Project Board (HVW Technologies $14.95)	$ 14.95
1	#27981 Parallax 433Mhz RF Receiver	$ 39.99
1	#27980 Parallax 433Mhz RF Transmitter	$ 29.99
1	Parallax 2x16 Serial LCD (Non-Backlit) # 27976	$ 24.99
1	#900-00001 Parallax Piezo Speakers 5v $1.95	$ 1.95
1	Set Wire Asst. Wire	$ .50
1	Cabinet	$ 2.00
1	Toggle Switch	$ .30

Other Components

1	.1uf Decoupling Capacitor	$ .05

Tools and Supplies

Computer, Printer, Soldering Iron, Solder, Scissors, Masking Tape, Screwdrivers (+/-), Magnifier, Hobby Drill, Drill bit, Cut-off Disk, Needle Nose Pliers, Wire Cutters, Wire Strippers, Cardboard, Rubber Bands, Paper, Sharpie Pen, Electrician's Pliers

Output

Ten piezo speakers provide sound output for applications and program debugging. The Stamp 1 Project Boards have a red power LED that outputs the battery/power status. Switches and power inputs are supported by a control panel made from a leftover clipboard piece.

Power LED

The Power LED output brightness on each computer is proportional to the input voltage. There are some interesting effects and experiments (see Apps) that take advantage of this effect.

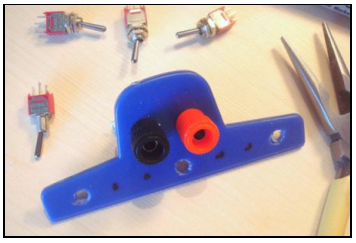
Banner

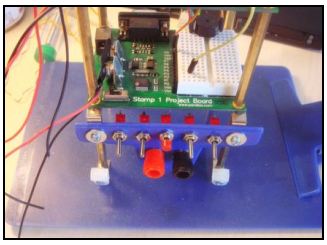
A banner was created using a printer. The purpose is identification of the supercomputer and the location and use of the pushbutton switch, toggle switches and power inputs. The banner is located at the top of the structure, across the front of the protective cage. A top and front fiberboard rectangular plate strengthens the cage and provides a mounting surface for the banner.

Fig. 5 The front Banner is a label denoting various functions, such as switch and plug function. It shows five switches, including a toggle for an event in time, software reset or condition notification, four input toggle switches (for data, flags, configurations, stats, debugging), and two 9-volt DC power inputs with positive (red) and negative (black) polarity.

Additional controls/indicators include added piezo speakers, and power LED lights on each Stamp 1 Project Board.

Control Panel with Input Array
The control panel has two banana jacks for 9-volts DC power input. There are four toggle switches for inputs read by the button command. A single pushbutton is for events. There are 80 pins, most of which can be used as inputs for sensors, keyboards, displays, and various peripheral circuits. The wiring for the Control Panel leads to Computer 1.



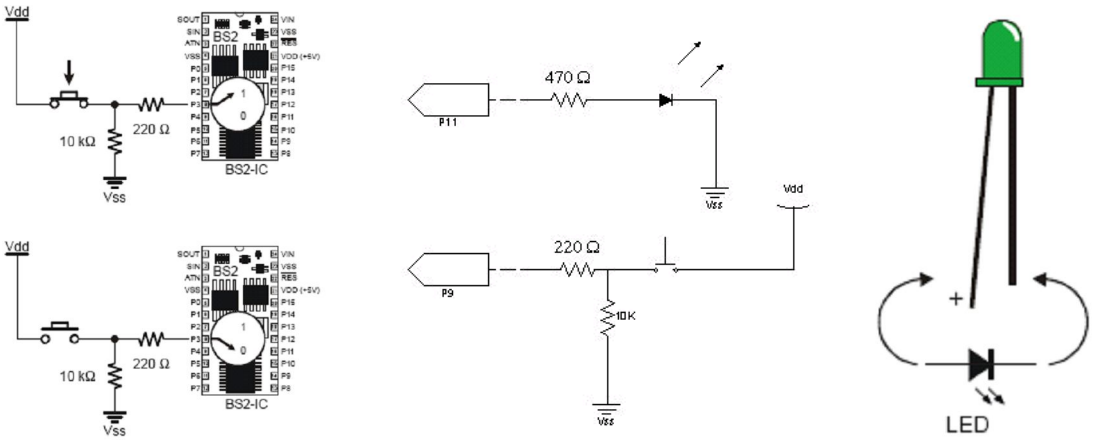
A leftover clipboard clamp makes up the Control Panel support. Hole positions are marked with a *Sharpie* pen and drilled out with a hobby drill set. CP holds four toggle switches, one pushbutton, and two power banana jacks.

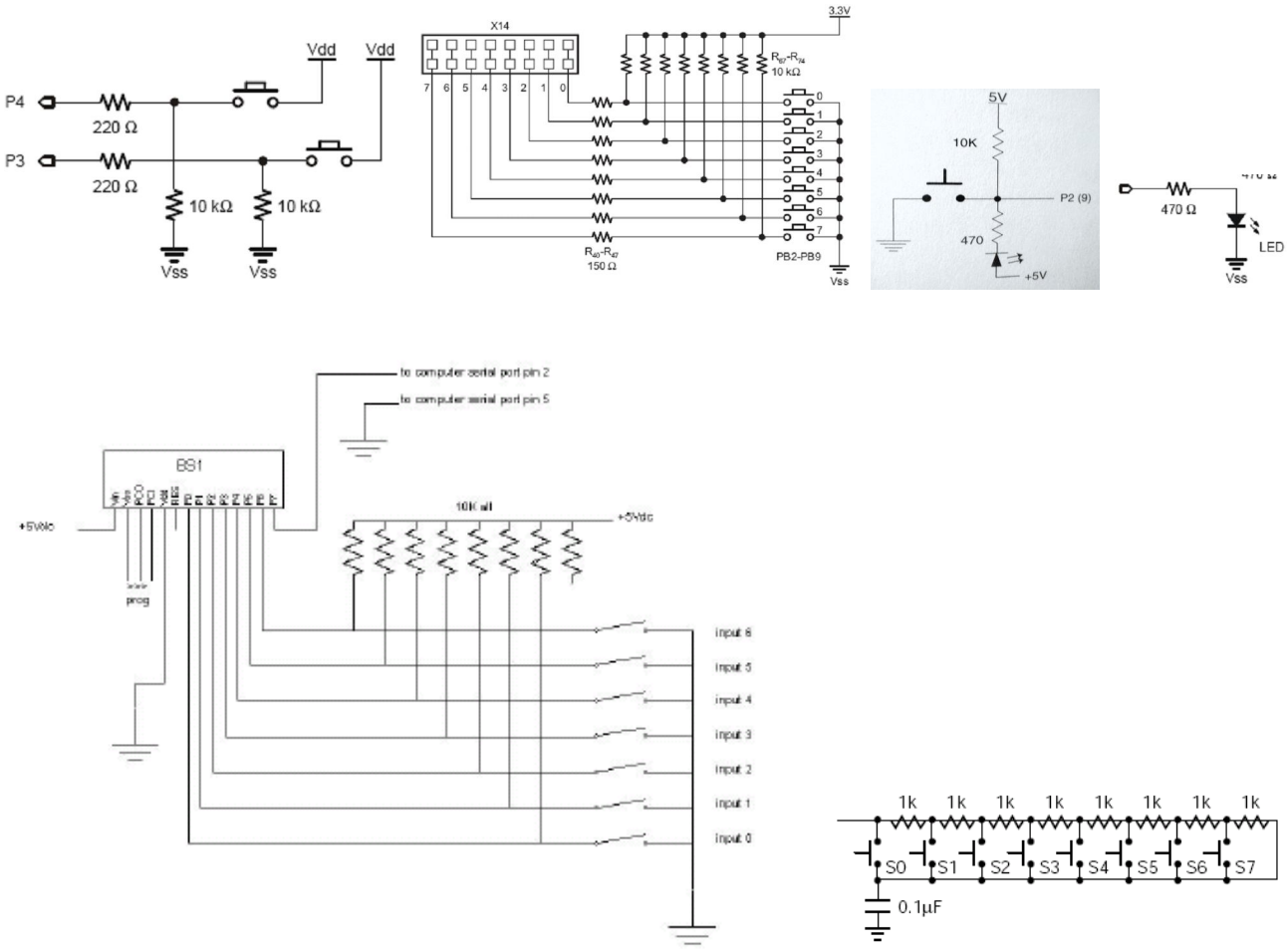
The completed Control Panel is positioned just above ground level by computer #1 and is very convenient to operate when sitting at a desk. The SEED is the first desktop Stamp Supercomputer. Note the base spacers hold two nylon cable clamps at the ground level, for host wiring.

```
Control Panel Computer One Pinout
---------------------------------
Interface           Pin 0
Enumeration         Pin 1
Toggle 1            Pin 2
Toggle 2            Pin 3
Toggle 3            Pin 4
Toggle 4            Pin 5
Pushbutton          Pin 6
Piezo               Pin 7
Red Banana Jack     to black wires battery clip harness
Black Banana Jack   to red wires battery clip harness *
* use of board 9V batteries clips reversed the color code
```

Wiring the Control Panel Pushbutton with LED
On Computer number one, A 220 ohm resistor is already attached to port 5 and not shown in the schematic seen below. Connect one side of the push button switch (orange wire) to ground. Connect the other switch side (white wire) to 10K and 470 ohm resistors. The LED is attached to the breadboard and positioned near the front.

Wiring and Reading Toggle Switche & LEDs – Assorted Example Circuits
Below are some wiring examples and polarity references for switching and LED lights.





Schematics show ways to connect various switches.

Switch Software
Instead of IN3 for watching an INPUT, you would want to use PIN3. For Example:

```
IF PIN3 = 1 THEN Servo2
```

All Computers - Pin Assignments
Computer 1 Port Assignments
Computer one, at the lowest physical level, is assigned to five switches and one piezo speaker with the enumeration pin intentionally open, taking up 8 ports.

Pin	Function (1 computer)
0	One Wire Interface
1	Enumeration
2	Toggle 1
3	Toggle 2
4	Toggle 3
5	Toggle 4
6	Pushbutton
7	Piezo Speaker

(8 used, 0 free)

Computer 2 through 9 Port Assignments (8 computers)
Computers 2 through nine each use three ports with five remaining.

Pin	Function
0	One Wire Interface
1	Enumeration
2	Open
3	Open
4	Open
5	Open
6	Open
7	Piezo Speaker
(3 used, 5 free)	x 8 = (24 used, 40 free)

Computer 10 Port Assignments (1 computer)
Computer 10 uses four ports with three remaining.

Pin	Function
0	One Wire Interface
1	Enumeration
2	Radio Transmit
3	Radio Receive
4	Open
5	Open
6	Open
7	Piezo Speaker
(5 used, 3 free)	

Global Computer Ports (Availability Depends on Projects)
All computers include an original 80 ports with 37 ports used and 43 remaining.

C1	8 used	0 remaining
C2-9	24	40
C10	5	3
Totals	37 used	43 remaining

Powering Up the Boards
Although individual BS1 boards have individual on/off switches, they must all be switched on at all times to run the Stamp SEED Supercomputer. Do not switch off individual boards. The design is for the SEED to run with all computers.

Running a Single Board
It may become necessary to run a single board out of the hive. To run a single board, it must be completely removed from the hive by disconnecting P0, Vss, and Vdd. Unclip the battery clip. Connect the power supply to the board's 9-volt battery terminals.

Peripherals
Peripherals include a piezo speaker and a wireless transmitter-receiver pair. The wireless transceiver can communicate with letters, numbers or words to other Parallax stamps, computers, supercomputers, or a base station.

Power LED lights on each board can also show the voltage power level. As power decreases from Vin (9-volts) down to 2 volts, the lights dim correspondingly.

Programming Section

(SEED) Artificial Intelligence Life Form –
Self Evolving Enumerated
Deterministic Programming.

It sounds like we might be creating consciousness… "replicated consciousness" with inherent "native intelligence" seems feasible. G. Herzog

Stamp AI gives a living soul to small Stamp microprocessors

In the beginning, there was power. One Stamp SEED fit into the Stamp Universe – the birthing process of new life began, each processor evolving, determining its own unique existence and purpose in life. Our new Stamp code learns about itself, creates a unique identity, learns about the others around (neighbors), communicates, naps, dreams, does work, and sleeps. What more could you want in only 256 bytes?

```
SEED Stamp Supercomputer
Self Evolving Enumerating Deterministic Life Program
One program loads into all computers
Self Enumerated Deterministic Timing and Reporting
Pin 1 is the Enumeration pin
```

Using Self Evolving Enumerating Deterministic programming is a kind of *quantum breakthrough* in simple hobby Basic Stamp supercomputers. The hardware consists of a series of clustered paralleled Basic Stamp computers communicating on pin 0 with a one wire interface. The chips become evolving dedicated self enumerators. There are dedicated *enumeration pins* to get the process started - pin 1 of each computer has an RC circuit that determines a unique ID. The same code loads into all computers. After that, the code changes and evolves. The first computer has pin 1 tied to a zero ohm resistor. Thus, its signature begins with the board's 220 ohm inline port resistor. Each successive pin increments by approximately (220) + 1000 until 10,220 is reached.

Self evolving, self aware, code modifying, one code for all, programs are quite complicated. The one 256 byte-fit program is divided into 13 phases. The challenge is one program fed to ten computers. From there, the same program is born and develops individual characteristics. It's like a Chess game, anticipating moves far in advance, when developing this program. Each computer uses a deterministic pin from which its evolution begins. It follows up by becoming self aware - choosing a self identity. It learns about its neighbors and carries on a conversation. It remembers information and exchanges information. It develops a sense of time, and a meaning in life. It learns and speaks a new language (as programmed). It chooses some work to do. When it sleeps, it dreams. A monitor can be attached to watch thoughts and dreams. At the end of the life cycle, it goes into suspended animation and does not die. It maintains a heartbeat once every 2.6 seconds. But, you can terminate it by turning off the power or loading in another program, only if you can.

This feeds into a circuit with a .1 uf capacitor thus creating an rc circuit (see schematic) that can be enumerated with a button command on each BS1 processor. One program, loaded into each computer, will read this number and remember it as its own personal unique ID number. It then becomes self determined and aware of its own computer ID. The computer then determines its position in life by creating a computer number c. This tells which block the computer lives on. The range is block 1 to block 10. Each computer then uses its own enumeration and converts it to a unique timing delay. The different timing delays are executed simultaneously by all computers. Each computer speaks its enumeration after the unique timing delay is ended. The first demo used PLAN language to accomplish this. Later expanded programs used this memory space for more function. Timing is incremental from computer 1 to computer 10. This is because the numeration determined from pin 1's resistance has created an increasing enumeration from one computer to the next.

All other computers are listening and record the ID. In this way, all computers will have a record of all unique computer IDs, knowledge of their neighbors in memory, and may decide who to talk to, when to talk, and what to talk about. In essence, the computers can now be assigned to completely different tasks. The computers can transfer and relate back variables and code snippets to one another. It is up to the programmer to take it from there. In summary, the code makes a choice to default to very specific subroutines, to decide which work to perform and which application to do. Using the ID, beginning at computer 1 as the origin, one program can call another program, or simply put – call another computer into action, or disable it.

So as a result of each computer reporting its self enumerated deterministic ID, everyone gets to know everyone else. This code is fit into a Stamp 1 EEPROM using PBASIC 1. Using Self Enumerating Deterministic programming, you only need to write one program, and load it into all processors, rather than writing 10 different programs and loading each into specific processors. So you can load Basic Stamps as parallel processors, have each individual computer learn who he is, and then begin running the main applications with all computers communicating through the messaging scheme over I/O pin 0.

How the code works

The Stamp SEED Supercomputer is a new concept. This is a ten core, ten month project, with the 1st Stamp AI software to fit into 256 bytes eeprom - self determinate, evolving, enumerating, dreaming, poetic, noisy, talkative, and downright friendly (based on software versions). It runs on only one program that self evolves differently in each of the ten computers. It's evolutionary - it's revolutionary!
http://forums.parallax.com/forums/default.aspx?f=21&m=361377&p=1

"Stamp AI gives a living soul to small Stamp microprocessors"

Load stamp_ai.bs1 into all ten computers. Connect to the computer you want to monitor and load the code. Turn off the entire supercomputer when the blue debug screen appears (before any words appear on the screen). Now turn on the supercomputer. The debug screen will show the thinking of the computer it is connect to. This is the *Thought and Dream Viewer*. It also shows what the life form is thinking and doing. It even shows dreams. All life forms are unique and develop individual personalities.

There is an unseen birthing process that takes place. This is the DNA part of the program, with directives, declarations, and initializations. Then, introduction takes place. Suddenly, evolution begins. The life form looks at pin 1 from which it determines its own unique identity, using a resistor capacitor circuit. Next, the unique identification (self enumerating) is used to determine its computer number and its physical location to know which block it lives on. There is a randomness to this enumeration. It can change from one startup to the next providing for unique variance in personalities. The supercomputer is divided into ten blocks, computer 1 lives on the lowest block while computer ten lives on the top block (deterministic).

This personal information is committed to memory. There are two memory locations, 0 and 1, designated for the id and computer number within each computer life. Throughout the life of the entity, this information is remembered and recalled. Now the life form will take a series of naps. The NAP number is 0, which represents 18 milliseconds. In between each nap, there is a dream. Dreams are pseudo random in nature and occur in *Vers Libre*, a kind of minimal abbreviated poetic verse. Dreams also include numbers and their pseudo random bit representation. Dreams are unique because they are seeded with the unique personal id.

After napping and dreaming, a deep sleep results. The length of sleep is directly related to the life form's computer number. The lower computer number life forms have shorter sleep, while the higher number life forms spend more of their time sleeping. Sleeping can have cause and effect, because in the next waking phase, a life form will talk - call out and speak its personal information to its neighbors. These are the friendly ones. Whether neighbors hear this information depends on whether they are sleeping or awake. The lazy ones will miss the information. In the extreme case, life form 10 sleeps so much that he misses hearing all of his neighbors, and becomes somewhat of a hermit. His memory has little information. Life form one sleeps the least time, and as a consequence, knows all his neighbors and is very friendly. His memory is rich and filled with all the neighbors information.

In the next phase of life, the memories about the neighbors are recalled. All of this appears on the viewer. Thinking and information always appear on the viewer. This is a unique privilege to see the life form's thought process on the screen. No life form is complete without doing some work in life. Each life form will start talking about its computer number, identification, how many seconds it did sleeping, how many neighbors it heard, how many neighbors it missed hearing, and how many dreams it had.

These life forms do not live long compare to humans. Their actual lives are approximately one minute in span. This is because they live in accelerated hyper time, based on a computer clock heart beat which is extremely fast (approximately 2,000 times per second). Much can be accomplished in *Accelerated Hyper Time*. The screen is slowed on purpose for humans to monitor the life forms. At the end of a life cycle, hibernation takes place. This is a kind of suspended animation with a heart beat of once every 2.6 seconds. Memories and information are retained but there is no known way to retrieve it in this time era.

Personality galore! You'll find these life forms interesting, unique and with varying personalities. The differences from one to the other include different identifications, different computer numbers, living in different blocks at different

height levels, different dreaming, varied sleep times, some are friendly and some don't know any neighbors. Some have filled enriched memories and remember all their neighbors while others can only remember two things and spend all their time sleeping. The code could have sufficient modification to prevent the life form from dying, and to create a *Finite Hibernation* in which it can wake up on demand. Instead of and END statement, the code could terminate in a continuous loop. Inside the loop, it looks at a pin status. If the pin goes high, the program can go back to some place in the code for continued function. The pushbutton could be wired to the input port for this to happen.

Sometimes the life form will become sick and require personal sick leave. The sicknesses are varied. For example, the following maladies are discussed in further detail elsewhere – psychotic breakdown, fitful sleep, sudden death, brain amnesia, unable to resuscitate, alternate identity syndrome, stroke, epileptic pins, and stuttering.

A Summary of Artificial Intelligence Software
for the Stamp SEED Supercomputer
in 256 Bytes of BS1 EEPROM

Artificial Intelligence Software
AI software is now ready for the Stamp SEED Supercomputer. A version now exists for the Stamp TriCore Supercomputer. (see project posting at the Parallax Forum and the apps in this book)

Code Posting
Look for posted code for download at the Parallax Project Forum.
http://forums.parallax.com/forums/?f=21&m=361377&g=361378#m361378

Sample Screens
Note, computer 1 hears all nine other computers because it sleeps the least amount of time (while computer 10 misses hearing everything because it was sleeping when all the others were talking).

Only One Program
SEED is unlike previous stamp supercomputer software that had a different program load into each computer. SEED is only one program - a life form. The same program loads into all computers in the supercomputer.

Life Form
The program is born and then it evolves into a life form! The collective supercomputer has ten life forms that are evolving at the same time.

Unique
After evolving, no two life forms are alike.

Life Span
Each life form has a life span of about one minute of activity.

Birth
It is first born based on its predetermined DNA, in this case, declarations and announcement.

Self Enumerating
Next, it looks at itself and determines its own unique enumerating identity. It does this by looking at pin 1 which contains a unique rc circuit.

Self Deterministic
Then SEED self determines its own computer number. It is now self aware and knows its own unique identity and its position in the world (which block it lives on and at what dwelling level).

Memory & Recall
It then exercises its memory by remembering its own characteristics and demonstrates its recall ability.

Napping & Random Dreaming
All of this leads to taking a series of naps followed by pseudo random dreams. The dream are in Vers Libre, a kind of simple word-verse poetry. The nap time is accumulative, with similar rest effects that humans have. In this case, the power is reduced from 1 ma to 25 ua. The pseudo random dream generator is seeded by the unique identity of the life form.

Dream Watching
A Dream Viewer puts the dreams up on the debug screen for viewing. It includes words, the random binary bit, and the pseudo random byte decimal number.

Sleep & Fuzzy Clock
The life form will now sleep using its fuzzy alarm clock. The approximate sleep time is unique and unlike the other life forms because it is based on its own unique characteristic of its computer identification number. The higher the number, the longer it sleeps. Therefore each life form has its own sleeping habits.

Talking to Neighbors
It wants to learn more about its world, so it sends out a message telling who he is and what his identity is to anyone listening.

Listening to Neighbors
Next, he listens for a reply. If he is life form number 1, he will get 9 additional replies. If he is number 5, he should have 5 replies. Finally, life form 10 will think he is alone. It's because each computer sleeps a different length of time. If a life form is sleeping, it will not hear the others talking.

Getting to know Neighbors
Now he gets to know his neighbors on a personal name basis. He memorizes the personal information of each neighbor that is heard. Next, all the neighbors are remembered (see the Brain Viewer results).

Remembering Neighbors
He has memorized the personal information of each neighbor that is heard, and all neighbors are remembered and recalled.

Thinking & Working
Some thinking/working takes place to recall the personal identity, determine the number of seconds in sleep and how many life forms were heard, plus how many life forms were not heard, and how many dreams were remembered.

End of Life
Finally, the life ends but is not entirely terminated, as it goes into stasis (I am not comfortable about creating life and then causing it to die).

Personalities
Each life form has its own evolved personality. There are several unique features. The personal identity, computer number, its living location, the random dreams it has, and the length of sleep habit. Its sleeping habit determines how friendly or how much of an isolated hermit it will become. It also determines how much information about neighbors it will remember. Some life forms remember a lot of information while others remember very little. Thinking outcome is unique too.

Restrictions
Due to memory restrictions, PLAN Piezo Language is not included with this version, even though PLAN is a spinoff of the SEED software. The entire communications view port was built upon the Debug screen.

Becoming Sick
Because these are ten individual life forms, sickness can result. Below is a synopsis.

```
' PERSONIFIED MALADIES
' --------------------
' When you download a program into the BASIC Stamp 1, it is stored in the
' EEPROM starting at the highest address (255) and working towards the
' lowest address. Most programs don't use the entire EEPROM, so the lower
' portion is available for other uses. This portion is used for long term
' memories. As a result, like a real human person, this artificial life
' form is capable of succumbing to sickness and malady.

' PSYCHOTIC BREAKDOWN
' -------------------
' AI memory begins at memory location 0 and works upward. If
' the memories become too much, they will fall upon code and a psychotic
' action will take place, resulting in a breakdown as the main program
' will be overwritten.
```

```
' FITFUL SLEEP
' ------------
' We don't know how effective sleep really is for the AI supercomputer
' as its sleep is constantly interrupted every 2.3 seconds.
' The Basic Stamp 1’s output pins will toggle briefly when using SLEEP,
' NAP or END. 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
' "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 effect 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 effect can be countered 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 effect behavior because the interpreter chip enters
' a low-power state.
'
' SUDDEN DEATH, BRAIN AMNESIA, NONRESUSCIATING SITUATION
' ------------------------------------------------------
' Life Terminates after power off, or by running another program. If a
' 2nd program is run (or the same program run a 2nd time), it will put all
' 0s into the EEPROM, thus destroying any memories and the original identity.
' It will be useless trying to resuscitate and bring back the original
' memories.
'
' ALTERNATE IDENTITY SYNDROME
' ---------------------------
' The stamp is reset causing the program to run again, and the birthing
' process takes place and a different identity emerges.
'
' STROKE
' ------
' The stamp is reset or repowered. It tries to communicate with other
' stamps but it is not in sync and the serial command hangs.
' It is unable to reply or do anything at this point.
'
' STUTTERING
' ----------
' Battery low causing short term repeating system resets (generally happens
' with a BS board containing a brownout detector)
```

Code Fit
The code will fit any of the Basic Stamp Supercomputers built with BS1s with some parameter changes to indicate the total number of computers in the supercomputer. This works with the Stamp SEED Supercomputer, the TriCore Supercomputer, and the Stamp Tiny Supercomputer.

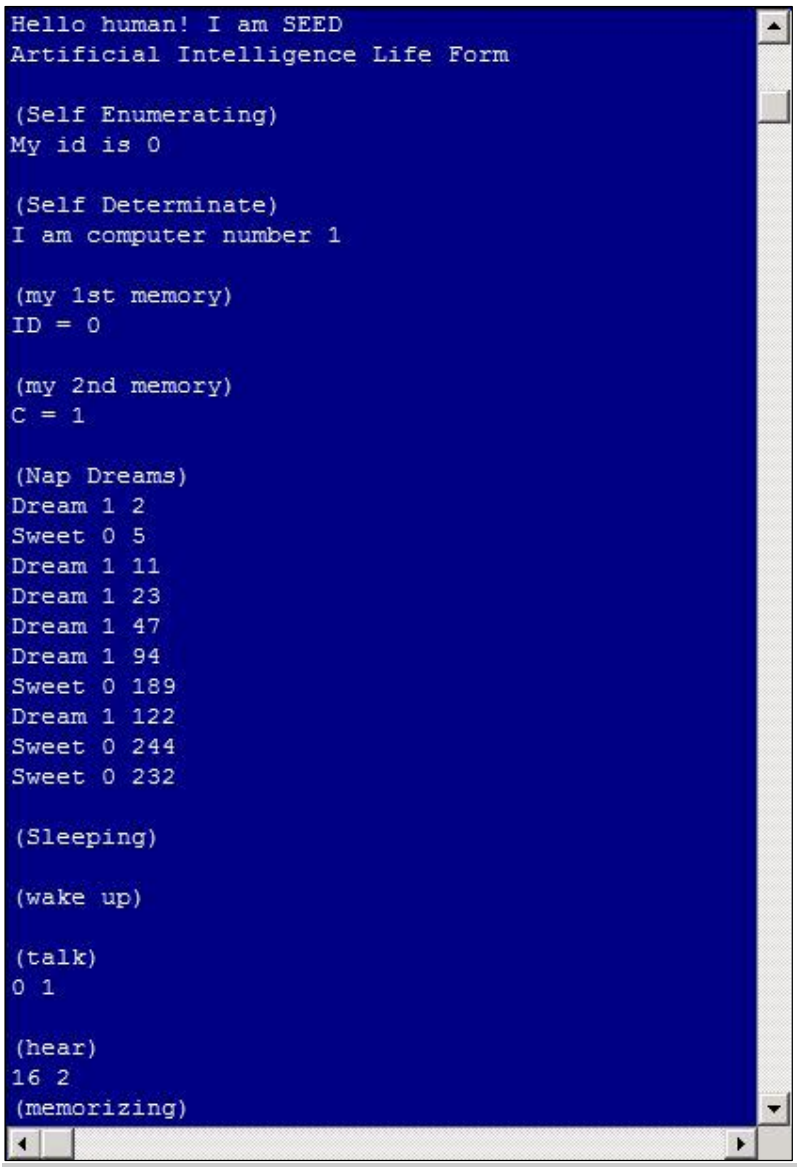
Processor Type
Since the programming is in PBASIC 1 and for a BS1, it will not run on a BS2 without some modifications.

Changing the Code
You are invited to change the code, but remember it is interlaced with timing. When life forms speak, it is based on timing.

Flow Control
Serial flow control is established by sending, and listening for the "!" command. This allows multiple life forms to listen to *conversation talk* at the same time.

Running the Code & Watching Evolution

The one program is loaded into each computer. Power off the supercomputer. Connect the cable to the computer you want to watch evolve as a life form. Power on the supercomputer. Load in this program bringing up the Debug screen. Before anything appears on the screen, power off the supercomputer. Leave the screen on, and power on the supercomputer. You can now watch the entire evolution take place automatically!



Left: Lifeform Screens- to get an idea of the evolutionary process

The Story of Supercomputer Life – Comments by Dr. Humanoido

This program is a PBASIC AI Artificial Intelligence program, fitting into only 256 bytes of space in a BS1 microcontroller - the first ever written for a Stamp Super computer. The same program loads into all ten computers (or any number of computers).

Then it evolves and becomes self aware, based on unique conditions of a hardware pin and analog circuit. It is awesome to have ten life forms all living in the supercomputer at the same time, working together, talking back and forth and getting to know each other.

They evolve, develop unique personalities, even nap, have pseudo random dreams, have memories and recall, do work in their life span, and sleep.

There are several interesting stories. Out of ten life forms, they all sleep different times. One life form hardly gets any sleep and he know every neighbor on a first name basis. He *burns the midnight candle!* The last life form spends most of his time sleeping. The sleep is so deep, he does not even know he has neighbors! He leads a kind of hermit life.

Others evolve to somewhere in between these two extremes. They can get sick. When they do too much work, they have a psychotic breakdown (code memory gets overwritten). They can have a bad sleep, being interrupted every 2.3 seconds.

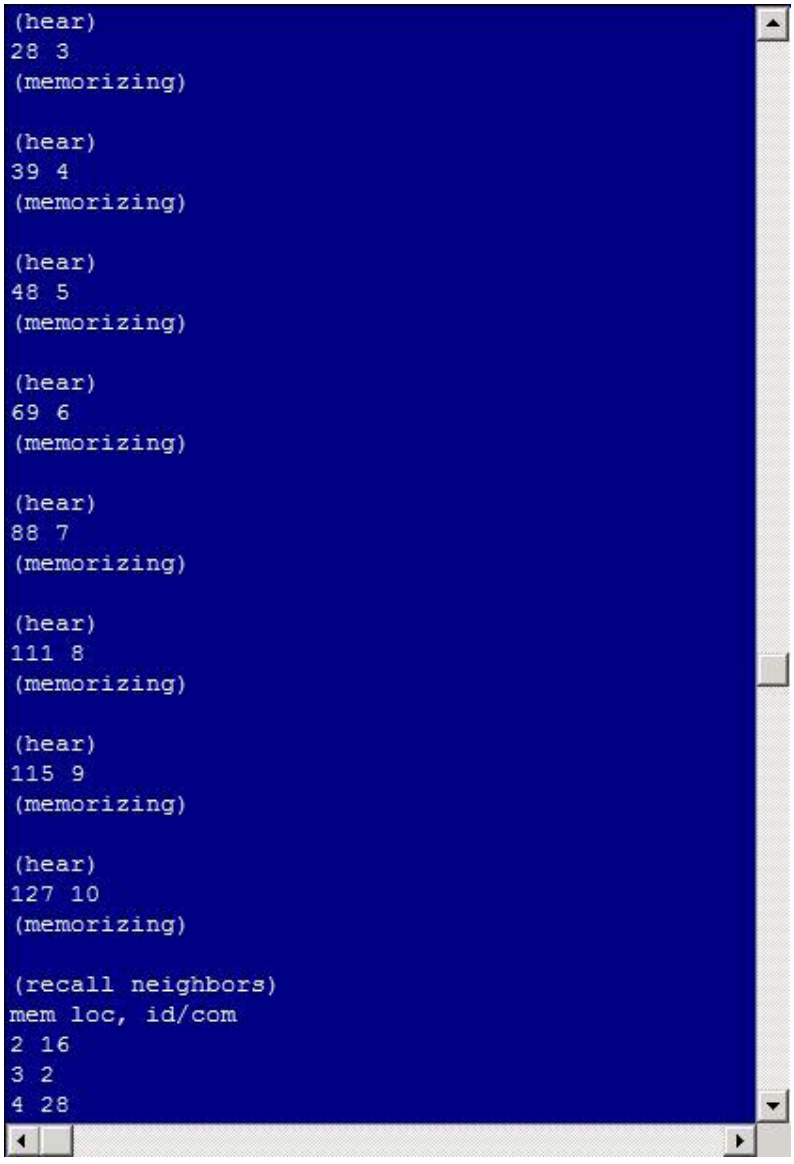
Alternate Identity Syndrome results if there is an unexpected power interruption. *Stroke* results when the synchronization is lost from one processor life form to the next. *Stuttering* may develop with repeat resets without a brownout detector. *Sudden death* happens with complete loss of power.

I wrote the program so they do not die, as long as power is not removed. At the end of the life span, they go into stasis. All of their memories and knowledge remain intact, but cannot be accessed. I could not see the benefit of creating a life form and then causing it to die.

In stasis, they are in a suspended animation, in a very low consumptive state, at only 25 micro amps. They can live on a small amount of juice at this time (battery power), going down to a mere 2 volts. I made a Dream Viewer so you can watch dreams and thoughts on the screen. It dreams in a special *Vers Libre*, a kind of abbreviated English verse which includes numbers. Someone asked if its scary. It's not scary - its all very interesting...

Stamp_ai.bs1 Code Listing

```
' ************************************************************************
' INTRODUCING STAMP AI ***************
' ************************************************************************
'
' stamp_ai.bs1 for the Stamp SEED Supercomputer v3.8
' Stamp AI gives a living soul to small Stamp microprocessors
'
' Self Evolving Enumerating Deterministic Code (SEED)
' An Artificial Intelligence System for the Basic Stamp 1
' Developed for Stamp SEED Supercomputer (which has ten Stamp computers)
' by Dr. Humanoido 06.23.09 - developer of AI language Writer -
' AI program that writes 25,000 computer programs for Japanese Robot
' Manufacturing Systems, Industrial Manufacturing, Artificially
' Intelligent CNC Programming
```




```
'
********************************
********************************
**********
' FEATURES   ***************
'
********************************
********************************
**********

' Very Simple AI performed in
256 bytes eeprom
' This one program will load
into ten computers and self
evolve.
' Software versions exist for
Stamp supercomputer with ten
and three
' computers. Refer to the
Minuscule Stamp Supercomputer
for the "three
' computers" version.

' * Fits into 1/4th K (256
Bytes) EEPROM
' * Simple self modifying code
capable (modifies eeprom
technique)
' * Less than 100 code
statements
' * The first complete AI
system that fits into the BS1
' * Has "Miniscule AI" at the
core, adaptable to other Stamps
' * 17 Phases of Life Cycle
' * Thought monitor (brain
thinking is relayed to the pc
screen)
' * Dreams are shown on the
screen for real time monitoring
' * Has a Virtual Fuzzy Clock,
not exact but more human-like

' ************************************************************************
' LIFE CYCLE PERFORMANCE   ***************
' ************************************************************************

'     (See additional description in the Stamp Seed Supercomputer Manual)
' 1)  read pin 1 and determine id                        (self enumerating)
```

```
' 2)  determine computer number based on id range         (deterministic)
' 3)  remember computer id & number                       (memorize)
' 4)  recall memory of computer id & #                    (remember)
' 5)  nap and pseudo random dream in "Vers Libre"         (nap, dream)
' 6)  sleep fuzzy clock - its computer number in seconds  (sleep)
' 7)  wake to send its computer number and id             (wakeup, talk)
' 8)  Listen for others' computer numbers                 (listen)
' 9)  if heard, remember other computer numbers and id's  (know what is heard)
' 10) do some work, retrieve the other ids...            (work)
' 11) goes into suspended animation
```

```
4 28
5 3
6 39
7 4
8 48
9 5
10 69
11 6
12 88
13 7
14 111
15 8
16 115
17 9
18 127
19 10
20 0
21 0

(Thinking)
I am computer 1
My id is 0
I slept  3  sec
I heard  9  com
I missed 0  com
I had    10 dreams

(bye)
```

```
' another view of the life process
' 01) Birth (Declares language of origin, physical attributes)
' 02) Self Deterministic (finds its own identity and uniqueness)
' 03) Early Memory (early formative memory, remembers its identity)
' 04) Develops or recognizes a sense of time
' 05) Talks to others, announces itself and its personal identity
' 06) Speaks another Language (PLAN - Piezo Language)
' 07) Self Evolving (learns information about others)
' 08) Long Term Memory (remembers information about others)
' 09) Sense of Purpose/Meaning of Life (establishes work)
' 10) Naps in 1s intervals (with dreams in between)
' 11) Dreams (dreams about random numbers and poetry)
' 12) Suspended Animation (END statement induces suspended state)
' 13) Sleeps
' 14) Alive (Live Code, Power Down or 2nd program run terminates life)
```

```
' ************************************************************************
' PHASES OF SEED'S MINUSCULE AI  ***************
' ************************************************************************

' PHASE 0 - PREPARE THE ENUMERATION PIN
' PHASE I - FORMATION AND BIRTHING PROCESS - DECLARATIONS
' PHASE II - PERSONAL INTRODUCTION
' PHASE III - SELF ENUMERATION - CREATING A SELF AWARE IDENTITY
' PHASE IV - THOUGHT MONITOR
' PHASE V - SELF DETERMINATION
' PHASE VI - EARLY MEMORY - REMEMBERING SELF IDENTITY
' PHASE VII - NAP
' PHASE VIII - DREAM
' PHASE IX - DEVELOPING A SENSE OF TIME (FUZZY CLOCK)
' PAHSE X - SLEEP
' PHASE XI - CONVERSATION - INTRODUCTION TO NEIGHBORS
' PHASE XII - CONVERSATION - LISTEN TO NEIGHBORS
' PHASE XIII  - MEMORIZING THE IDENTITY OF NEIGHBORS
' PHASE X1V  - LONG TERM MEMORY - RECALL IDENTITY OF NEIGHBORS
' PHASE XV  - SPEAK LAN PIEZO LANGUAGE (FOR EXPANSION)
' PHASE XVI - FINDING PURPOSE IN LIFE - DO WORK/THINKING
' PHASE XVII - END OF LIFE CYCLE - SUSPENDED ANIMATION/HIBERNATE
```

```
' ************************************************************************
' HISTORY ***************
' ************************************************************************

' Makes use of "Miniscule AI" (Interpreter originally developed by
' Humanoido for the Toddler Humanoid and modified for Stamp Supercomputers)
' Uses the idea of self enumerated deterministic programming, a Propeller
' Supercomputer approach originally proposed by Chip Gracey at the
' Propeller Supercomputing thread on 4-18-2006. Reference:
' http://forums.parallax.com/forums/default.aspx?f=25&p=1&m=121344

' The code will load into a Basic Stamp Supercomputer equipped with the
' Self Enumerating Pin Circuit. The details to construct one are included
' below. The BS1 handles up to 100 commands.

' Each computer sets up its own fuzzy clock timer. The clock counts, in
' seconds, equal to the computer number. Computer numbers range from 0 to 10.
' Some scaling may take place.

' Self evolving, self aware, code modifying, one code for all, programs
' are quite complicated. The one 256 byte-fit program is now divided into
' phases. The challenge is one program is fed to ten computers. From there,
' the same program is born and develops individual characteristics. It's like
' a Chess game, anticipating moves far in advance, when developing this program.

' Each computer uses a deterministic pin from which its evolution begins.
' It follows up by becoming self aware and choosing a self identity. It learns
' about its neighbors and carries on a conversation. It remembers information
' and exchanges information. It develops a sense of time, and a meaning in life.
' It learns and speaks a new language. It chooses some work to do. When it
' naps, it dreams. A monitor can be attached to watch thoughts and naps on
' each computer.

' ************************************************************************
' PHASE 0 - PREPARING ENUMERATION PINS  ***************
' ************************************************************************

' Hardware consists of 10 Basic Stamp computers. Each computer
' has one RC circuit on pin 1 (see schematics below), prepared in
' advance.

' * is a 220 ohm resistor built into each Stamp 1 Project Board.

' Software scaling is set to 54. The program must go pin 1 low and
' pause 2000 for repeat reliable readings.
'
'              *
'             220
' pin 1 <-----^^^^^----- no circuit, open       COMPUTER 1 ID = 00
'           resistor
'
'              *        R2       C2
'             220       1K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 2 ID = 17
'           resistor  resistor  capacitor       brown-black-red
'
'
'              *        R3       C3
'             220       2K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 3 ID = 28
'           resistor  resistor  capacitor       red-black-red
'
'
```

```
'               *        R4       C4
'              220       3K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 4 ID = 40
'           resistor  resistor  capacitor       orange-black-red
'
'
'               *        R5       C5
'              220       3.9K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 5 ID = 49
'           resistor  resistor  capacitor       yellow-black-red
'
'
'               *        R6       C6
'              220       5.1K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 6 ID = 70
'           resistor  resistor  capacitor       green-brown-red
'
'
'               *        R7       C7
'              220       6.8K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 7 ID = 89
'           resistor  resistor  capacitor       blue-gray-red
'
'
'               *        R8       C8
'              220       8.2K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 8 ID = 111
'           resistor  resistor  capacitor       gray-red-red
'
'
'               *        R9       C9
'              220       9.1K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 9 ID = 116
'           resistor  resistor  capacitor
'
'
'               *        R10      C10
'              220       10K     .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 10 ID = 128
'           resistor  resistor  capacitor
'
'


' Typical IDs are 00, 17, 28, 40, 49, 70, 89, 111, 116, 128

' TYPICAL PIN 1 RESISTANCE CODE CHART
' Computer 1       220 ohm (built in to Stamp 1 Project Board)
' Computer 2       220 + 1K Ohm   = 01.22K
' Computer 3       220 + 2K Ohm   = 02.22
' Computer 4       220 + 3K Ohm   = 03.22
' Computer 5       220 + 3.9K Ohm = 04.12
' Computer 6       220 + 5.1K Ohm = 05.32
' Computer 7       220 + 6.8K Ohm = 07.02
' Computer 8       220 + 8.2K Ohm = 08.42
' Computer 9       220 + 9.1K Ohm = 09.32
' Computer 10      220 + 10 K Ohm = 10.22

' ************************************************************************
' PHASE I - FORMATION AND BIRTHING PROCESS  ***************
' ************************************************************************

' {$STAMP BS1}            ' Self deterministic program
' {$PBASIC 1.0}           ' humanoido July 2009
SYMBOL  result  =   B0    ' (random)Nap & Dream - a random number
SYMBOL   n      =   BIT1  ' (random) bit1 number is 0 or 1 from B0 above
SYMBOL   id     =   B1    ' pin read id number of this computer (0-126)
```

```
SYMBOL   c      =   B2    ' determined computer number (1-10)
SYMBOL   v      =   B3    ' id range value for each computer, random dreams counter
SYMBOL   h      =   B4    ' no. of heard/missed coms (never heard - was sleeping)
SYMBOL   x      =   B5    ' slush value for thinking, recycled as needed
SYMBOL  epin    =   1     ' Enumerating pin
SYMBOL  ipin    =   0     ' Rx Tx Interface pin for serial in and out
SYMBOL  ppin    =   7     ' Piezo speaker pin
SYMBOL  baud    =   N1200 ' Baud rate, inverted, n81 format

' The BS1 provides variables from B0-B13 and W0-W6. Variable Examples:
' SYMBOL  XXX         = B0   ' Bit Addressable
' SYMBOL  XXX         = B1   ' Bit addressable
' SYMBOL  XXX         = B2   ' Bit addressable
' SYMBOL  XXX         = B3   ' Bit addressable
' SYMBOL  XXX         = B4   ' Bit addressable
' SYMBOL  XXX         = B5   ' Bit addressable
' SYMBOL  avail       = B6   ' value can be 0 to 255
' SYMBOL  avail       = B7   ' value can be 0 to 255
' SYMBOL  avail       = B8   ' value can be 0 to 255
' SYMBOL  avail       = B9   ' value can be 0 to 255
' SYMBOL  XXX         = W5   ' Consumes B10 & B11
' SYMBOL GOSUB/RETURN = W6   ' Consumes B12 & B13

' When a program is first run, all variables are cleared to 0. The variables
' total to 16 bytes of RAM and are arranged as follows:
'
' Data            Words   Bytes   Bits
' 0000 0000       PORT    PINS    PIN0 - PIN7  (PINS.0, PORT.0 - PINS.7, PORT.7)
' 0000 0000               DIRS    DIR0 - DIR7  (DIRS.0, PORT.8 - DIRS.7, PORT.15)
'
' 0000 0000       W0      B0      BIT0 - BIT7  (B0.0, W0.0 - B0.7, W0.7)
' 0000 0000               B1      BIT8 - BIT15 (B1.0, W0.8 - B1.7, W0.15)
'
' 0000 0000       W1      B2
' 0000 0000               B3
'
' 0000 0000       W2      B4
' 0000 0000               B5
'
' 0000 0000       W3      B6
' 0000 0000               B7
'
' 0000 0000       W4      B8
' 0000 0000               B9
'
' 0000 0000       W5      B10
' 0000 0000               B11
'
' 0000 0000       W6      B12
' 0000 0000               B13
'
' PORT is the I/O PORT word:
'
'        PINS AND PIN0-PIN7 are the PORT (RB) PINS (buffered to overcome the
'        READ-modify-WRITE problem).  When these variables are read, RB is read
'        directly.  When these variables are written to, the corresponding RAM
'        is written to, which is then transferred TO RB before each instruction.
'
'        DIRS AND DIR0-DIR7 are the PORT (RB) direction bits: 0=INPUT, 1=OUTPUT
'        (buffered to overcome the WRITE-only nature of the TRIS registers).
'        This byte of data is transfered TO !RB before each instruction.
'
' W0-W6, B0-B13, AND BIT0-BIT15 are for general-purpose use.
' W6 (B12,B13) is used as a four-level stack if any GOSUB/RETURNs are executed.

' *****************************************************************
```

```
' PHASE II - PERSONAL INTRODUCTION    ***************
' ************************************************************************

DEBUG "Hello human! I am SEED",CR,"Artificial Intelligence Life Form ",CR,CR

' This code is loaded into all ten computers (Stamp SEED Supercomputer).

' Flowchart
' Begin Stamp Baby Supercomputer SBS
'     Each computer has an R/C circuit on pin 1 (see schematic)
'     Load one program (this program) into all 10 computers
'          Each computer: Read Pin 1 rc value using the POT command
'                scale the value into a unique computer ID
'                Remember this ID
'                Delay a time based on the unique ID
'                Send out the ID to the network after delay time
'                Each computer records all the id numbers it hears on the net
'                Remember and recall some information
'                Do some work
'                End

' Some Program Elements
' ---------------------
'       RC circuit
'       Data logger
'       EEPROM reader
'       EEPROM writer
'       Time Scaler
'       Fuzzy Clock
'       Serial transmitter
'       Serial receiver
'       Debug Screen

' ************************************************************************
' PHASE III - SELF ENUMERATION - CREATING A SELF AWARE IDENTITY **********
' ************************************************************************

' SELF ENUMERATION
' read pot (POT PotPin, Scale, read_value), display
' LOW 1:PAUSE 2000:POT 1, 54, B1:DEBUG CLS,B1

LOW   epin       ' get id from self enumerate pin, read rc circuit = ID number
PAUSE 2000       ' cap charge & settle time on pin 1
POT   epin,54,id ' pin 1, scale, determine id, read pot (POT PotPin, Scale, read_value)

' ************************************************************************
' PHASE IV - THOUGHT MONITOR   **************
' ************************************************************************

' The Thought Monitor allows you to peer into the computer's mind
' and view the thought process. Plug in the pc computer and
' view the thought process through the DEBUG screen. The thinking
' of each computer on the supercomputer can be viewed one at a time.
' You can listen in to each computer brain using the debug screen
' and learn what they are thinking, remembering, and what work they
' are doing at the time.

DEBUG "(Self Enumerating)",CR,"id is ",#id,CR,CR ' show on debug screen

' ************************************************************************
' PHASE V - SELF DETERMINATION   **************
' ************************************************************************

Loop:   'Determine Computer Number - loop for all ten computer possibilities
LOOKUP c, (12,23,35,45,55,76,95,114,122,134), v ' lookup logged id range values v
c = c + 1                                        ' next range lookup address
```

```
IF id >=  v THEN Loop                          ' if id > range, try again
DEBUG "(Self Determinate)",CR,"I am computer number ",#c,CR,CR ' show computer #

' ************************************************************************
' PHASE VI - EARLY MEMORY - REMEMBERING SELF IDENTITY
' ************************************************************************

' MEMORY MAP (eeprom 85% full)
' 00              computer id     (enumerating)
' 01              computer number (deterministic)
' 02-21           other computers, alternating id & computer # (see below)

' DATALOGGING MEMORY MAP FROM 02 TO 21 (LOG SHOWN FOR COMPUTER #1 ONLY)
' MEMORY LOCATION  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18
19  20  21
' DATA STORED      ID1 C1  ID2 C2  ID3 C3  ID4 C4  ID5 C5  ID6 C6  ID7 C7  ID8 C8  ID9
C9  ID10 C10

WRITE 0,id                                     ' memorize id number
WRITE 1,c                                      ' memorize computer number
READ 0,id                                      ' recall this computer's id
DEBUG "(my 1st memory)", id,CR,CR              ' show computer number is remembered
READ 1,c                                       ' recall this computer's computer number
DEBUG "(my 2nd memory)",c,CR,CR,"(Nap Dreams)" ' show computer id is remembered

' ************************************************************************
' PHASE VII - NAP
' ************************************************************************

' Nap#  Nap   Time
' 0     18    ms
' 1     36    ms
' 2     72    ms
' 3     144   ms
' 4     288   ms
' 5     576   ms
' 6     1.152 s
' 7     2.304 s

v = 0                          ' init a new counter
result = id                    ' seed pseudo random # generator with the id
Again:                         ' loop
v=v+1                          ' count the dreams
IF v=11 THEN FuzzyClock        ' do 10 dreams @@@ (based on the total # of computers)
RANDOM result                  ' randomize a byte
BRANCH n, (Dream1, Dream2)     ' go to bit 0 or 1, BRANCH value, (Case_0, Case_1,
Case_2)

' ************************************************************************
' PHASE VIII - DREAM   ***************
' ************************************************************************

' The computers will dream when napping. Dreams are totally
' random in nature. After a while, a repeat dream may occur.
' Each computer dreams about random numbers which are shown
' on the Dreamscape monitor. Dreams may also involve machine
' poetry but we are not sure.

Dream1:                        ' begin 1st dream
NAP 0                          ' nap reduce 1ma to 25us
DEBUG CR, "Sweet ",#n,#result  ' show dream & random number
GOTO Again                     ' next random dream
Dream2:                        ' begin second dream
NAP 0                          ' nap 18ms, reduce 1ma to 25us
DEBUG CR, "Dream ",#n,#result  ' next random dream or end of dream, show rnd numb
GOTO Again                     ' repeat
```

```
' ************************************************************************
' PHASE IX - DEVELOPING A SENSE OF TIME (FUZZY CLOCK)
' ************************************************************************

FuzzyClock:                   ' this computer will sleep equal to c*3
x=c*3                         ' sleep formula

' IDs are 00, 17, 28, 40, 49, 70, 89, 111, 116, 128  = B0

' 00*3=0
' 17*3=51
' 28*3=84
' 40*3=120
' 49*3=147
' 70*3=210
' 89*3=44
' 111*3=55
' 116*3=80
' 128*3=64

' ************************************************************************
' PHASE X - SLEEP
' ************************************************************************

' SLEEP allows the BASIC Stamp to turn itself off, then turn back on
' after a programmed duration. The length of SLEEP can range from 2.3
' seconds to slightly over 18 hours. Power consumption is reduced to the
' amount described previously, assuming no loads are being driven.
' The resolution of the SLEEP instruction is 2.304 seconds. SLEEP rounds
' the specified number of seconds up to the nearest multiple of 2.304.
' For example, SLEEP 1 causes 2.3 seconds of sleep, while SLEEP 10 causes
' 11.52 seconds (5 x 2.304) of SLEEP.

' Pins retain their previous I/O directions during SLEEP. However, outputs
' are interrupted every 2.3 seconds during SLEEP due TO the way the chip
' keeps time. The alarm clock that wakes the BASIC Stamp up is called the
' watchdog timer. The watchdog is a resistor/capacitor oscillator built
' into the interpreter chip. During SLEEP, the chip periodically wakes up
' AND adjusts a counter TO determine how long it has been asleep. If it
' isn't time to wake up, the chip "hits the snooze bar" and goes back to
' sleep.

' To ensure accuracy of SLEEP intervals, the BASIC Stamp periodically
' compares the watchdog timer to the more-accurate resonator time base.
' It calculates a correction factor that it uses during SLEEP. As a result,
' longer SLEEP intervals are accurate to approximately ±1 percent. If your
' application is driving loads (sourcing or sinking current through
' OUTPUT-HIGH OR OUTPUT-LOW PINS) during SLEEP, current will be
' interrupted FOR about 18 ms when the BASIC Stamp wakes up every 2.3
' seconds. The reason is that the watchdog-timer reset that awakens the
' BASIC Stamp also causes all of the pins to switch to INPUT mode for
' approximately 18 ms. When the interpreter firmware regains control of
' the processor, it restores the I/O directions dictated by your program.

' If you plan to use END, NAP, OR SLEEP in your programs, make sure that
' your loads can tolerate these periodic power outages. The simplest
' solution is often to connect resistors HIGH OR LOW (TO +5V OR ground)
' as appropriate to ensure a continuing supply of current during the
' reset glitch.

DEBUG CR,CR,"(Sleeping)",CR   ' display sleeping mode
SLEEP x                       ' sleep its computer number in seconds
DEBUG "(wake up)",CR          ' Display clock status

' ************************************************************************
```

```
' PHASE XI - CONVERSATION - INTRODUCTION TO NEIGHBORS
' ********************************************************************

' Announce id & c to Everyone, SEND ID SO OTHERS MAY DATALOG IT
' After self delay timing, send the ID to net

SEROUT ipin,baud,("!",id,c)  ' wake up, announce this computer's ID and #,
DEBUG "(talk)",CR,#id,#c,CR  ' show id & c were sent, computer 10 is last to wake,

' ********************************************************************
' PHASE XII - CONVERSATION - LISTEN TO NEIGHBORS
' ********************************************************************

' Listen for Neighbors id & c
' For example, with 3 computers:
' if c = 1   x = 1 to 2  listen for two computers  x = 1 to (3-c)
' if c = 2   x = 1 to 1  listen for one computer
' if c = 3   goto end    do not listen, this is the last computer and
'                        cannot count itself as listening to itself

' @@@ SET THE SUPERCOMPUTER'S NUMBER OF COMPUTERS IN THESE STATEMENTS!!!

v=2                          ' init the write addr, start at 2
Listen:                      ' listen to other computers speaking their id's
IF c = 10 THEN Think         ' jump if computer 10, cannot listen as sleeping @@@
x = 10 - c                   ' equation to determine looping @@@
FOR h = 1 TO x               ' do the loops
SERIN  ipin,baud,("!"),id,c  ' look at serial pin 1 for incoming id & computer #
DEBUG "(hear)",CR,#id,#c,CR  ' show received id & c
GOSUB Memorize               ' remember what you hear
NEXT                         ' next interation
GOTO Recall                  ' recall what what heard

' ********************************************************************
' PHASE XIII  - MEMORIZING THE IDENTITY OF NEIGHBORS
' ********************************************************************

Memorize:                    ' place heard computer's id and c in memory
DEBUG "(memorizing)",CR,CR   ' memorize the id and c values
WRITE v,id                   ' datalog to eeprom, v is eeprom addr = 2
v=v+1                        ' inc eeprom addr
WRITE v,c                    ' datalog to eeprom
v=v+1                        ' increment memory loc
RETURN                       ' listen again *** what happens when no more computers?

' ********************************************************************
' PHASE XIV  - LONG TERM MEMORY - RECALL IDENTITY OF NEIGHBORS
' ********************************************************************

' SET THE SUPERCOMPUTER'S NUMBER OF COMPUTERS IN THESE STATEMENTS ********

Recall:                      ' Read/display neighbors id & c
DEBUG "(recall neighbors)",CR ' screen report about remembering
v=1                          ' init mem loc
DEBUG "mem loc, id/com ",CR  ' screen report for storage location, id & computer #
Repeat:                      ' recall memory subroutine
v = v+1                      ' begin at mem loc 2
READ v,x                     ' recall v and id
DEBUG #v,#x,CR               ' display mem loc, read val
PAUSE 500                    ' slow to see screen vals
IF v>20 THEN Think           ' CHANGE THIS TO REFLECT WHICH SUPERCOMPUTER! @@@
                             ' OK TO USE 20 FOR 10 COMPUTERS
GOTO Repeat                  ' again

' ********************************************************************
' PHASE XV - FINDING PURPOSE IN LIFE - DO WORK/THINKING
```

```
' ************************************************************************

' in its simplest form, this life form will use memory recall to indicate
' its personal computer number and its personal identity.

Think:                        ' think routine

DEBUG CR,"(Thinking)",CR       ' announce thinking
READ 1,c                       ' read computer number at loc 1
DEBUG "I am computer ",#c,CR   ' announce my computer number
READ 0,id                      ' read id at loc 0
DEBUG "My id is ",#id,CR       ' announce my id

' the life form will use memory to recall its computer number and
' determine how long it slept.

x=c*3                          ' formula for length of sleep
DEBUG "I slept  ",#x," sec",CR ' how long did I sleep?
h=10-c                         ' formula for the number of heard coms @@@
DEBUG "I heard  ",#h," com",CR ' announce # of computers heard
h=9-h                          ' formula for the number of missed coms @@@
DEBUG "I missed ",#h," com",CR ' announce computers not heard
DEBUG "I had    10 dreams ",CR ' announce number of dreams
GOTO Halt                      ' end

' ************************************************************************
' PHASE XVI  - SPEAK PLAN PIEZO LANGUAGE
' ************************************************************************

' This section is for expansion using the new spinoff Piezo Language PLAN
' Note that some life forms will have the characteristics of Piezo Speech,
' while others will not. No individuals have the same voice sound. Since
' tone is directly related to personal id, computer 1 will have no voice
' while ascending computers (ranked in increasing computer numbers) will
' have increasing pitch voices. Some low tones may or may not be heard.
' It's all in the personality of the individual life form (piezo speakers
' vary in their sound, and human ears vary in their perceptibilities).

'OUTPUT 7                       ' make pin 7 output for speaker
'SOUND  7, (id, 60)             ' id = tone, 60 = duration
'INPUT  7                       ' prevent noise on speaker pin

' ************************************************************************
' PHASE XVII - END OF LIFE CYCLE - SUSPENDED ANIMATION/HIBERNATION
' ************************************************************************

' At the end of the life cycle, it goes into a kind of hibernation or
' suspended animation. It maintains a heartbeat once every 2.6 seconds.
' But, you can terminate it by turning off the power or loading in
' another program (not recommended). It's not cryogentic suspended
' animation but a similar form achieved by the electronics and software
' of this microcontroller.

Halt:
DEBUG CR,"(bye)"
END

' ************************************************************************
' SELF MODIFYING CODE     ***************
' ************************************************************************

' Self modifying code is possible in theory but not easy. In fact, it's
' frowned upon because it becomes very difficult, virtually impossible,
' to debug for one thing. And there are alternatives - that's what
' subroutines are for, and decision code. To create a 100 byte program
' that is self-modifying could require a thousand bytes of decision code
```

```
' to decide how to modify the critical 100 byte part. Constants are compiler
' directives - when you use them in your program, the value of the constant
' is plugged into the memory location. In theory, you could change a value
' in EEPROM, however finding which value to change would be difficult.

' A more simple way to have immediate success in self modifying is to
' store initialized data in an otherwise unused part of the EEPROM and
' change it as needed. Program code loads into EEPROM from the top down.
' EEPROM and WRITE commands work from the bottom up. A simple way to self
' modify EEPROM is by writing to locations using the WRITE command, and
' rewriting as necessary. It takes the least amount of code and space to
' do this - one statement per rewrite location. So we now have a simple
' way, in minimal memory and code, to successfully achieve self modifying
' code.

' ************************************************************************
' PERSONAL SICK LEAVE ***************
' ************************************************************************

' When you download a program into the BASIC Stamp 1, it is stored in the
' EEPROM starting at the highest address (255) and working towards the
' lowest address. Most programs don't use the entire EEPROM, so the lower
' portion is available for other uses. This portion is used for long term
' memories. As a result this and other effects, like a real human person,
' this artificial life form is capable of succumbing to sickness.

' PSYCHOTIC BREAKDOWN - AI memory begins at memory location 0 and works
' upward. If the memories become too much, they will fall upon code and a
' psychotic action will take place, resulting in a breakdown as the main
' program will be overwritten.

' FITFUL SLEEP - We don't know how effective sleep really is for the AI
' supercomputer as its sleep is constantly interrupted every 2.6 seconds.
' The Basic Stamp 1s output pins will toggle briefly when using SLEEP,
' NAP or END. 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
' "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 because the interpreter chip enters
' a low-power state.
'
' SUDDEN DEATH, BRAIN AMNESIA, NONRESUSCIATING SITUATION
' Life Terminates after power off, or by running another program. If a
' 2nd program is run (or the same program run a 2nd time), it will put all
' 0s into the EEPROM, thus destroying any memories and the original identity.
' It will be useless trying to resuscitate and bring back the original
' memories.
'
' ALTERNATE IDENTITY SYNDROME - The stamp is reset causing the program to
' run again, and the birthing process takes place and a different identity
' emerges.
'
' STROKE - The stamp is reset or repowered. It tries to communicate with
' other stamps but it is not in sync and the serial command hangs.
' It is unable to reply or do anything at this point.
```

```
'
' STUTTERING - Battery low causing short term repeating system resets, generally
' happens with a board containing a brownout detector.
```

What happens when the program ends? Does the life form die?
End is the command statement. Ending the program places the BASIC Stamp into low-power mode indefinitely. This is equivalent to having a program that does not loop continuously; once the BASIC Stamp reaches the end of the PBASIC program, it enters low-power mode indefinitely. The END command is optional, but recommended. END puts the BASIC Stamp into its inactive, low-power mode. In this mode the Stamp's current draw (excluding loads driven by the I/O pins) is reduced to the amount shown above. END keeps the BASIC Stamp inactive until the reset line is activated, the power is cycled off and back on or the PC downloads another program. Just as with the SLEEP command, pins will retain their input or output settings after the BASIC Stamp is deactivated by END. For example, if the BASIC Stamp is powering an LED when END executes, the LED will stay lit after END, but every 2.3 seconds, there will be a visible wink of the LED as the output pin switches to the input direction for 18 ms. Approximate current draw @ 5 VDC during Run is 5ma. Approximate current draw @ 5 VDC during END is 25ua. This is an approximate value, not including loads on the I/O pins.

How to save or preserve the state of the life form?
When a program is ended (BS1), and power is kept on, the memory and variables are retained. This is a kind of hibernation or suspended animation. After the END statement is executed, and the program ends, is there any simple way to retrieve data from the eeprom and/or run the program again? Obviously we cannot reset the stamp or recycle power, or everything will be lost. I am dealing with an AI life form and don't want it to die. Of course, instead of an END statement, the code could go into in a continuous loop. Inside the loop, it looks at a pin. If the pin goes high, the program goes back to the beginning or somewhere predetermined in between.

When you download a new program that has DATA or EEPROM statements, that data is copied to EEPROM. If you later use a WRITE statement to modify the EEPROM data, the modified data is retained. Say you have a byte value at EEPROM location zero that's initialized by an EEPROM statement to the value zero. When your program runs, it does something that produces a non-zero result which gets written to EEPROM location zero, then the program ENDs. If power comes on or the BS1 is reset, the first thing the program does is to READ location zero and initialize itself based on the value read. That state memory will be retained through on/off or reset/END cycles. By only saving the state at the end of the program, you avoid wearing out the EEPROM location. Note that you don't have to use only one byte, you could save all the variables to EEPROM, but many of them won't need to be saved.

So mainly the eeprom is preserved as long as the original program is kept loaded, the power is recycled, or the stamp is reset. So before the END statement, the code should write all necessary variables to eeprom, then read the variables back in after a power recycle or reset. You should define the initial state of the variable values using the EEPROM statement since that will be what's initially there when you download the program. This gives several programming opportunities for maintaining an AI life form, both at the beginning of program loading and at the "end" of the program to maintain state.

Multiple Ways to Program a Stamp Supercomputer – Wait then Retrieve Method
There are several ways to program multiple stamps. The SERIN command can also be configured to wait for specified data before it retrieves any additional input. For example, suppose a device that is attached to the BASIC Stamp is known to send many different sequences of data, but the only data you desire happens to appear right after the unique characters, "XYZ". The BS1 has optional *Qualifier* arguments for this purpose.

```
SYMBOL  Data   = B2
Main:
SERIN 1, N2400, ("XYZ"), #Data
END
```

The Software – Self Modifying Code
Self modifying code is possible in theory but not easy. In fact, it's frowned upon because it becomes very difficult, virtually impossible, to debug for one thing. And there are alternatives – that's what subroutines are for, and decision code. To create a 100 byte program that is *self-modifying* could require a thousand bytes of decision code to decide how to modify the critical 100 byte part. Constants are compiler directives - when you use them in your program, the value of the constant is plugged into the memory location. In theory, you could change a value in EEPROM, however finding which value to change would be difficult. A more simple way to have success in self modifying is to store initialized data in an otherwise unused part of the EEPROM and change it as needed. Program code loads into EEPROM from the top down. EEPROM and WRITE commands work from the bottom up. A simple way to self modify

EEPROM is by writing to locations using the WRITE command, and rewriting as necessary. It takes the least amount of code and space to do this - one statement per rewrite location. So we now have a simple way, in minimal memory and code, to successfully achieve self modifying code.

The Language
It knows very little English so don't expect full grammar and complete sentences on brain scans.

Brain Scanning
Watch the debug screen for various brain thinking.

EEPROM Write Limitation
Keep in mind that any given location in EEPROM can only be written maybe 100,000 times. This seems large, but can easily occur in a few hours if there's an error in a program that runs continuously. So what happens when *writes* reach a maximum? You wind up with certain locations that can't be fully erased. Charge gets trapped in the insulating layer of silicon dioxide used to store a bit and can't be completely removed during the internal erase operation. At some point, around 100,000 cycles, one or more bits at a location become *stuck* at zero since the erased state is all ones and not enough of the stored charge can be swept out of the insulating layer to be read as a one in those bits.

Computer Enumerating Pin Circuits – Theory and Practicality
Build these circuits for each computer and install on the breadboards. Note the 220 ohm resistor is built into the project boards already and can be taken into consideration for the total resistance. C1 is .1uf throughout. A zero K resistor is no resistor. Vss is ground. Note, when perfect resistor values are not available, such as 500K, substitute the closest value. Guidelines are provided.

```
' CIRCUIT
' -------
' * is a 220 ohm resistor built into each Stamp 1 Project Board
'
'               *        R1       C1
'              220       0K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 1
'           resistor  resistor  capacitor
'
'               *        R2       C2
'              220       1K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 2
'           resistor  resistor  capacitor
'
'
'               *        R3       C3
'              220       2K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 3
'           resistor  resistor  capacitor
'
'
'               *        R4       C4
'              220       3K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 4
'           resistor  resistor  capacitor
'
'
'               *        R5       C5
'              220       4K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 5
'           resistor  resistor  capacitor
'
'
'               *        R6       C6
'              220       5K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 6
'           resistor  resistor  capacitor
'
'
'               *        R7       C7
```

```
'              220       6K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 7
'           resistor  resistor  capacitor
'
'
'               *        R8       C8
'              220       7K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 8
'           resistor  resistor  capacitor
'
'
'               *        R9       C9
'              220       8K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 9
'           resistor  resistor  capacitor
'
'
'               *        R10      C10
'              220       9K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss         COMPUTER 10    (10K)
'           resistor  resistor  capacitor
'
' PIN 1 RESISTANCE CODE CHART
' --------------------------
' Computer 1       220 ohm (built in to Stamp 1 Project Board)
' Computer 2       220 + 1K Ohm = 1220
' Computer 3       220 + 2K Ohm = 2220
' Computer 4       220 + 3K Ohm = 3220
' Computer 5       220 + 4K Ohm = 4220
' Computer 6       220 + 5K Ohm = 5220
' Computer 7       220 + 6K Ohm = 6220
' Computer 8       220 + 7K Ohm = 7220
' Computer 9       220 + 8K Ohm = 8220
' Computer 10      220 + 9K Ohm = 9220
```

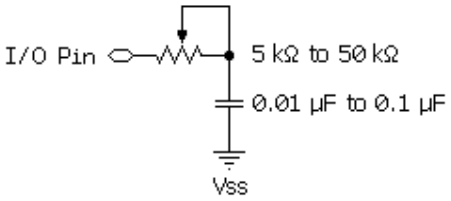
Part List for Enumerating Pin Circuit

C1 through C9	Capacitor	.1uf	Color Code
R1	Resistor	0K ohms	
R2	Resistor	1K	brown black red
R3	Resistor	2K	red black red
R4	Resistor	3K	orange black red
R5	Resistor	4K	yellow black red
R6	Resistor	5K	green black red
R7	Resistor	6K	blue black red
R8	Resistor	7K	violet black red
R9	Resistor	8K	gray black red
R10	Resistor	9K	white black red

```
' SDEP.BS1   by humanoido v1.0 06.24.09
' A Basic Stamp Supercomputer SEED Program
' Self Evolving Enumerated Deterministic Supercomputer Program (SEED)
' Determines a unique ID number for a computer which runs this code.
' The uniqueness of the ID is based on resistor and capacitor values.

' This program uses the POT command. Connect one side
' of a 10K potentiometer to P1.  The other side of the potentiomenter
' connects to a 0.1 uF capacitor. Then connect the second side of the
' capacitor to Vss (ground). Determine the best Scale factor by trying
' out values.

' This program will read the charge/discharge time on the capacitor.
' Returned numbers can have scale and range. Full scale is 255.
' To set the scale, try a value of 50 to reduce returned numbers.
' (B1) becomes the self deterministic enumerated computer
' identification. It is a unique ID number based on the charge
```

```
' time across pin 1 (capacitor + resistor).

' {$STAMP BS1}
' {$PBASIC 1.0}
Loop:POT 1, 50, B1              ' read pot level (POT PotPin, Scale, read_value)
DEBUG CLS,B1:PAUSE 50:GOTO Loop ' display, short delay, again
```

Syntax: POT Pin, Scale, Variable
Read a 5 kΩ to 50 kΩ potentiometer, thermistor, photocell, or other variable resistance. **POT** reads a variable resistance and returns a value (0 - 255) representing the amount of time it took to discharge the capacitor through the resistance. Pin must be connected to one side of the variable resistance, whose other side is connected through a capacitor to ground, as shown below. **Pin** is a variable/constant (0 - 7) that specifies the I/O pin to use. This pin will be set to output mode initially, then to input mode. **Scale** is a variable/constant (0 - 255) used to scale the command's internal 16-bit result. See Explanation below for steps to finding the scale value to use for your circuit.
Variable is a variable (usually a byte) where the final result of the reading will be stored. Internally, the POT command calculates a 16-bit value, which is scaled down to an 8-bit value.



POT works by first setting the specified I/O pin to an output and setting its state high. This step places +5 volts on one side of the capacitor and ground (0 volts) on the other side, which charges the capacitor. POT will hold the pin high for 10 milliseconds to charge the capacitor. It is important to select component values that will allow the capacitor to charge in this period. If, for example, a 50 kΩ potentiometer is used with a 0.1 µF cap, the maximum charge time would be: Charge time: 5 x (50 x 103) x (0.1 x 10-6) = 25 x 10-3

The full charge time (5 TC) is 25 x 10-3 or 25 milliseconds. Obviously, the capacitor would never fully charge when the potentiometer is at its maximum position. In this case, changing the capacitor to 0.01 µF reduces the charge time to 2.5 milliseconds; well within the range of the POT function. After the capacitor is charged the I/O pin to an input mode and POT starts its timer. Initially the I/O pin will see a high (1) that will eventually drop to a low (0) when the capacitor discharges past the 1.4-volt threshold. The timer stops once the low is seen. The value of the variable resistor affects the time it takes to discharge the capacitor from 5 volts to approximately 1.4 volts. The 16-bit reading is multiplied by (Scale ÷ 256), so a scale value of 128 would reduce the range by approximately 50%, a scale of 64 would reduce to 25%, and so on. The amount by which the internal value must be scaled varies with the size of the resistor being used.

Finding the best Scale value: Build the circuit shown above and plug the BS1 into the PC. Select *Run | POT Scaling*. A special calibration window appears, allowing you to find the best value. The window asks for the number of the I/O pin to which the variable resistor is connected. *Select the appropriate pin (0 - 7).* The editor downloads a short program to the BS1 (this overwrites any program already stored in the BS1). Two numbers will be displayed: scale and value. Adjust the resistor until the smallest number is shown for scale (assuming you can adjust the resistor, as with a potentiometer). Once you've found the smallest number for scale, you're done. This number should be used for the Scale in the POT command. Optionally, you can verify the scale number found above by selecting the *POT Value checkbox*. This locks the scale and causes the BS1 to read the resistor continuously. The window displays the value. If the scale is good, you should be able to adjust the resistor, achieving a 0-255 reading for the value (or as close as possible). To change the scale value and repeat this step, just uncheck the box. Continue this process until you find the best scale.

EEPROM (a directive at compile time, not at run time)
Syntax: EEPROM {Location,} (DataItem {, DataItem,...})
Function: Write data to the EEPROM during program download. Location is an optional variable/constant (0 - 255) that specifies the starting location in the EEPROM at which data should be stored. If no location is given, data is written starting at the next available location. DataItem is a constant (0 - 255) to be stored in EEPROM. Explanation: When you download a program into the BASIC Stamp 1, it is stored in the EEPROM starting at the highest address (255) and working towards the lowest address. Most programs don't use the entire EEPROM, so the lower portion is available for other uses. The EEPROM directive allows you to define a set of data to store in the available EEPROM

locations. It is called a "directive" rather than a "command" because it performs an activity at compile-time rather than at run-time (i.e., the EEPROM directive is not downloaded to the BASIC Stamp 1, but the data it contains is downloaded). The simplest form of the EEPROM directive is something like the following:

```
EEPROM  (100, 200, 52, 45)
```

This example, when downloaded, will cause the values 100, 200, 52 and 45 to be written to EEPROM locations 0, 1, 2 and 3, respectively. You can then use the READ and WRITE commands in your code to access these locations and the data you've stored there. The EEPROM directive uses a counter, called a pointer, to keep track of available EEPROM addresses. The value of the pointer is initially 0. When a program is downloaded, the EEPROM directive stores the first byte value at the current pointer address, then increments (adds 1 to) the pointer. If the program contains more than one EEPROM directive, subsequent EEPROM directives start with the pointer value left by the previous EEPROM directive. For example, if the program contains:

```
EEPROM  (72, 69, 76, 76, 79)
EEPROM  (104, 101, 108, 108, 111)
```

The first EEPROM directive will start at location 0 and increment the pointer for each data value it stores (1, 2, 3, 4 and 5). The second EEPROM directive will start with the pointer value of 5 and work upward from there. As a result, the first 10 bytes of EEPROM will look like the following:

	EEPROM Location (address)									
	0	1	2	3	4	5	6	7	8	9
Contents	72	69	76	76	79	104	101	108	108	111

What if you don't want to store values starting at location 0? Fortunately, the EEPROM directive has an option to specify the next location to use. You can specify the next location number (to set the pointer to) by using the optional Location argument before the list of DataItems. The following code writes the same data in the table above to locations 50 through 59:

```
EEPROM  50, (72, 69, 76, 76, 79, 104, 101, 108, 108, 111)
```

In this example, the Location argument is given and tells the EEPROM directive to store the following DataItem(s) starting at location 50. The DataItems in the list are stored in their respective locations (50, 51, 52... 59). It is important to realize that the entire BASIC Stamp 1 EEPROM is overwritten during programming. Any EEPROM location not containing a PBASIC program or DataItems from an EEPROM directive is written with a 0. A common use for EEPROM is to store strings; sequences of bytes representing text. PBASIC converts quoted text like "A" into the corresponding ASCII character code (65 in this case). To make data entry easier, you can place quotes around a whole chunk of text used in a EEPROM directive, and PBASIC will understand it to mean a series of bytes (see the last line of code below). The following three EEPROM directives are equivalent:

```
EEPROM  (72, 69, 76, 76, 79)
EEPROM  ("H", "E", "L", "L", "O")
EEPROM  ("HELLO")
```

All three lines of code, above, will result in the numbers 72, 69, 76, 76, and 79 being stored into EEPROM upon downloading. These numbers are simply the ASCII character codes for "H", "E", "L", "L", and "O", respectively. The given code is a demonstration of storing and reading multiple text strings. The EEPROM is organized as a sequential set of byte-sized memory locations. The EEPROM directive only stores bytes into EEPROM. If you try to store a word-size value, for example: EEPROM (1125), only the lower byte of the value will be stored (in this case, 101). This does not mean that you can't store word-sized values, however. A word consists of two bytes, called a low-byte and a high-byte. If you wanted to store the value 1125 using the EEPROM directive you'll have to calculate the low-byte and the high-byte and insert them in the list in the proper order, as in:

```
EEPROM  (101, 4)
```

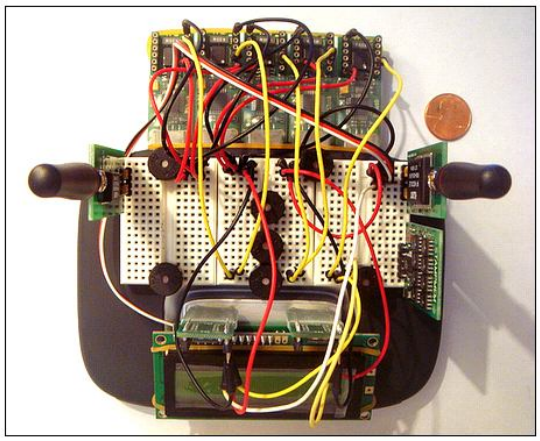
The directive above will store the two bytes into two sequential EEPROM locations (the low-byte first, followed by the high-byte). We calculated this in the following manner: 1) high-byte is INT(value / 256) and 2) low-byte is value - (high-byte * 256). To retrieve a word-size value, you'll need to use two READ commands and a word-size variable. For example,

```
SYMBOL  result  =  W1                  ' word-sized variable
SYMBOL  resLo   =  B2                  ' B2 is low-byte of W1
SYMBOL  resHi   =  B3                  ' B3 is high-byte of W1
EEPROM  (101, 4)
Main:
  READ 0, resLo
  READ 1, resHi
  DEBUG #result
  END
```

This code would write the low-byte and high-byte of the number 1125 into locations 0 and 1 during download. When the program runs, the two READ commands will read the low-byte and high-byte out of EEPROM (reconstructing it in a word-size variable) and then display the value on the screen. See the READ and WRITE commands for more information.

Base Station (Bstat) Baby can keep in touch with a Base Station named Bstat. Bstat has a similar radio transceiver operated by a BS1 Stamp Project Board in 2400 baud mode. The 8 ports support a transmitter, receiver, backlit LCD, piezo speaker, LED, keyboard decoder, and a 16-key keyboard for command input.

Bstat's range is full city block to the Stamp supercomputer, and it can relay information to other Bstats to increase the wireless range of communicating supercomputers.



Left: a Tiny Stamp Supercomputer contains a transceiver and can also double as a base station BSTAT.

History The history of the Baby Stamp Supercomputer originated along with the Basic Stamp Supercomputer BSS and the Three Dimensional Stamp Computer 3DSC.

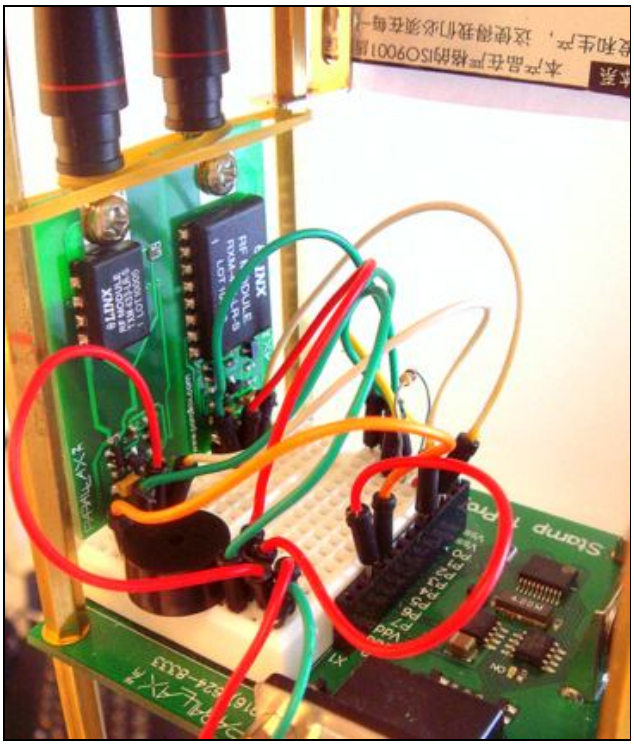
BSS provided the concept, proved out with the Master Offloader Machine MOM comprised of BS1's, and was verified with the 3D BS1 network on the 3DSC.

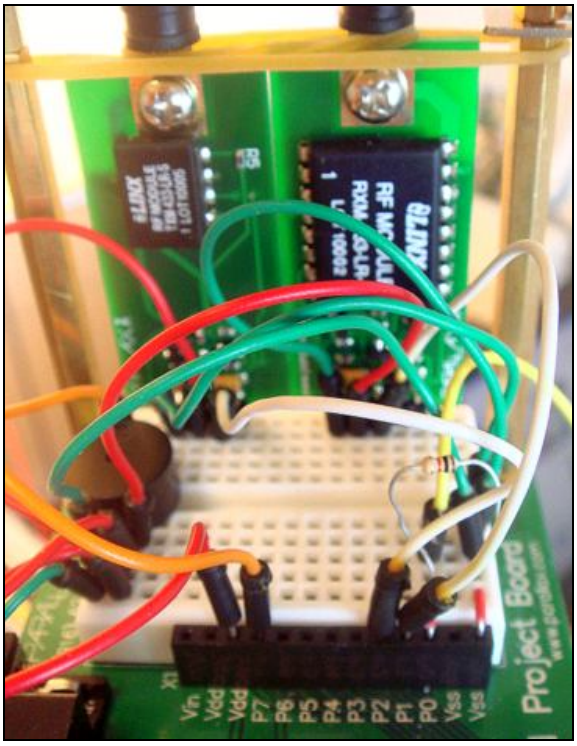
Concept, design and construction of the Stamp Baby Super computer spanned more than 9 months from 2008 through 2009. A commemorating plaque is placed at the top of the supercomputer designating the location of origin and the date, in English and Chinese.

Wireless Communications Transceiver *The Stamp SEED supercomputer* has a wireless communications interface made from a Parallax 433 Mhz RF wireless transmitter and a matching receiver. These are combined to create a bidirectional transceiver.

Computer ten handles the transceiver wiring and software to send and receive messages. According to Parallax specs, this assembly has a range of approximately 500 unobstructed feet. In New York City, the average length of a north-south block is 264-feet.

The transceiver has enough range to send commands past a city block or throughout a building with several rooms of same-equipped Stamp supercomputers! The transceiver has a transmit module on pin 2 and a receiver module on pin 3. Both modules are located topmost on the Skyscraper, on computer 10.





Transmitter and Receiver, side by side, are wired on computer ten. Follow the wiring in these photos or refer to the schematic. Each module takes one pin. This computer has only 3 out of 8 pins used. For testing, a base communicator was built.

This is refined into the Bstat, a full Base Station transceiver for wireless radio frequency communications built around a Basic Stamp 1. The Stamp Baby Super-computer and all of its peripherals are built around BS1s.

Transmitter and Receiver Specifications

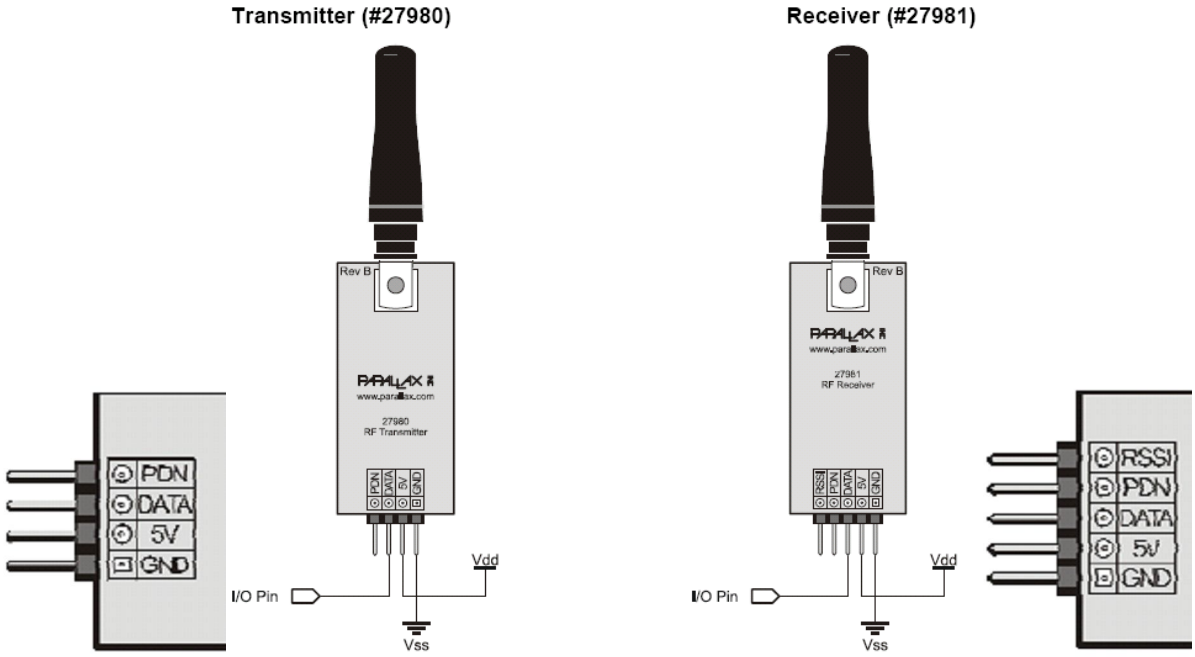
The Parallax 433.92 MHz RF Transmitter allows users to easily send serial data, robot control, or other information wirelessly. When paired with the matched RF Receiver, reliable wireless communication is as effortless as sending serial data. The power-down (PDN) pin may be used to place the module into a low power state (active low), or left floating (it is tied high internally). Some quoted specifications:
High-speed data transfer rates (1200 ~ 19.2k Baud depending on controller used)
SIP header allows for ease of use with breadboards
Compatible with all BASIC Stamp® modules (including BS1 and Javelin Stamp) and SX chips
As easy to use as simple SEROUT/SERIN PBASIC instructions
Power-down mode for conservative energy usage (longer battery life)
Line-of-sight range of 500 feet (or greater depending on conditions)

Transmitter Supply Current
at Logic High Input: 5.1 mA, at Logic Low Input: 1.8 mA, low power mode (PDN): 5 µA

Pin	Name	Function
1	PDN	Power Down - active low, pulled high internally so may be left floating
2	DATA	Data Output
3	5v	Power – connect to +5v DC (such as Vdd)
4	GND	Ground → 0 v (such as Vss)

Receiver Supply Current during Operation (High or Low): 5.2 mA, low power mode (PDN): 28 µA

Pin	Name	Function
1	RSSI	Received Signal Strength Indicator
2	PDN	Power Down - active low, pulled high internally so may be left floating
3	DATA	Data Input
4	5v	Power – connect to +5v DC (such as Vdd)
5	GND	Ground → 0 v (such as Vss)



The upper cage houses these two transceiver elements from Parallax. Awaiting wiring in this photo, each has a 1 ¾ inch antenna which is protected by the cage. Note the 1K ohm interfacing resistor from ground to P0. Also note there are tiny bus bars created on the solderless breadboard near the piezo speakers, for Vss and Vdd, appropriately color coded with green and red wiring.

Theory of Operation

Short for Radio Frequency, RF refers to the frequencies that fall within the electromagnetic spectrum associated with radio wave propagation. When applied to an antenna, RF current creates electromagnetic fields that propagate the applied signal through space. Any RF field has a wavelength that is inversely proportional to the frequency. This means that the frequency of an RF signal is inversely proportional to the wavelength of the field. The Parallax RF modules utilize a frequency of 433.92 MHz, this works out to be a wavelength of approximately 0.69 meters (2.26 feet, or 7.3e-17 lightyears). 433.92 MHz falls into the Ultra High Frequency (UHF) designation, which is defined as the frequencies from 300 MHz ~ 3 GHz. UHF has free-space wavelengths of 1 m ~ 100 mm (3.28 ~ 0.33 feet or 1.05e-16 ~ 1.05e-17 lightyears).

PDN Pulling the power down (PDN) line low will place the transmitter/receiver into a low-current state. The module will not be able to transmit/receive a signal in this state.

RSSI *(receiver only)* Received Signal Strength Indicator. This line will supply an analog voltage that is proportional to the strength of the received signal.

Calibration When initiating communication between the RF modules, a sync pulse should be sent to re-establish the radio connection between the modules. Sending several characters can accomplish this, however sending a pulse (which maintains a high state during the synchronization) is more efficient:

√ For BS1s the following code line sends an appropriate sync pulse:

```
PULSOUT 1, 300
```

√ For BS2s the following code line sends an appropriate sync pulse:

```
PULSOUT 8, 1200
```

√ For Javelin Stamp modules the following code line sends an appropriate sync pulse:

```
CPU.pulseOut(300, CPU.pin8);
```

(Note: this line assumes that I/O pin 8 is connected to the DATA line on the transmitter)

Source Code BASIC Stamp® 1 Example Programs The following BS1 programs will transmit (Tx) and receive (Rx) two word-sized counters (four bytes of data). DATA lines on each module should be connected to I/O pin 1 of each of the BS1 modules. The 5v and GND lines should be connected to a +5 v source (Vdd) and ground (Vss) respectively (for both Tx and Rx). The BS1 on the receiving end should remain connected to the computer to view the DEBUG statements.

```
' TxCode_v_1.0.bs1
'{$STAMP BS1}
'{$PBASIC 1.0}
'Parallax 433.92 MHz RF Transmitter (#27980) Sample Tx Code (BS1)
'Connect I/O Pin 1 of the BS1 to the DATA line on the RF module
'Connect +5v (Vdd) to the 5v line, connect Ground (Vss) to the GND line
'This code will transmit two word sized counters, byte at a time.
'Note the PULSOUT instruction: this helps the receiver sync with
'the transmitter, especially after lapses in communication.
'This code transmits at 2400 baud, inverted.
SYMBOL x = W1
SYMBOL y = W2
Start:
PULSOUT 1, 300 'Sync pulse for the receiver
SEROUT 1, N2400, ("!", B3, B2, B5, B4)
PAUSE 100
x = x + 1
y = y + 1
PAUSE 150
GOTO Start

'RxCode_v_1.0.bs1
'{$STAMP BS1}
'{$PBASIC 1.0}
'Parallax 433.92 MHz RF Receiver (#27981) Sample Rx Code (BS1)
'Connect I/O Pin 1 of the BS1 to the DATA line on the RF module
'Connect +5v (Vdd) to the 5v line, connect Ground (Vss) to the GND line
'This code will look for an "!", then read in four bytes.
'The first two bytes will be the high and low bytes of a word variable (x),
'the second two bytes will be the high and low bytes of a second word variable (y).
'The values of x and y will be sent out to the DEBUG terminal.
'This code receives at 2400 baud, inverted.
SYMBOL x = W1
SYMBOL y = W2
Start:
SERIN 1, N2400, ("!"), B3, B2, B5, B4
DEBUG x, y
GOTO Start
```

Power Distribution Vin, 9-volts DC power, enters through two front panel banana jacks and is evenly distributed to all ten boards. Each board can be switched off without disturbing the power distribution to the remaining computers. The positive or negative battery clips wires are connect in parallel. The mains power is switched off at the bench power supply or by removing the banana plugs and cable.

Power Draw

Power Draw of Individual Boards During Testing, number of boards versus current

No. of Computers	Current
1	06ma
2	13ma
3	19ma
4	26ma
5	32ma
6	39ma
7	45ma
8	52ma
9	58ma
10	65ma

Piezo Speakers and Current Draw

With wiring installed, each board draws 8 ma. All boards draw 80 ma. The power draw of piezo speakers run at the same time can be determined. In this experiment, one, then two, etc. piezo speakers were switched on and the

current draw was determined. The chart shows the number of piezo speakers switched on and current drawn. Since the draw is speaker and computer specific, the computer number is shown for identification.

Piezo Speaker	Current	Computer #
1	67ma	10
2	68	9
3	70	8
4	71	7
5	73	6
6	74	5
7	75	4
8	77	3
9	78	2
10	80	1

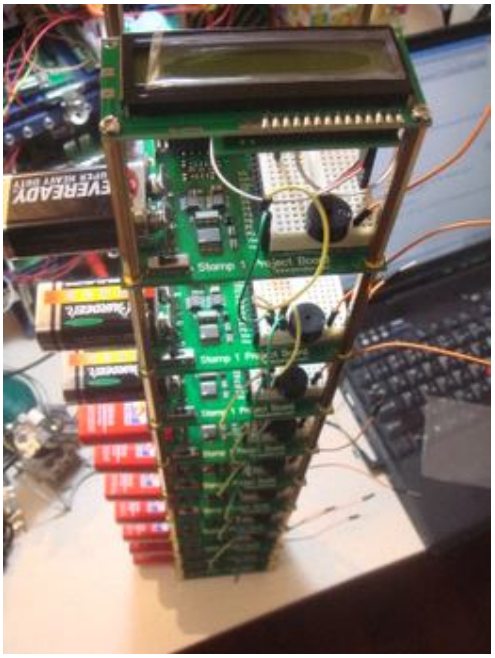
Sleep Nap and Wake
Basic Stamps have sleep, nap and wake modes. Only a few milliamps are required to keep the Baby Supercomputer running in wake mode without the piezo speakers or transceiver active. With a nine volt battery on each board at 6ma base drain, the Baby should run a long time! This is undoubtedly one of the lowest power Stamp supercomputers in existence at this time.

Demo Software The Baby is operated in several ways. One example is with ten programs running in parallel, driving a ten core cluster of processors and peripherals. Included software demos all ten computers talking to each other in Piezo Language. This demo begins with computer 1 at the bottom of the tower. It single beeps with the piezo at its specific frequency and then tells computer 2 to carry on. Computer two then speaks through its piezo speaker with its specific frequency and two beeps. Another demo is included for the transmitter and receiver, for wireless communications to other supercomputers, and computers and a Stamp base station.

The following BS1 programs will transmit (Tx) and receive (Rx) two word-sized counters (four bytes of data). DATA lines on each module should be connected to I/O pin 1 of each of the BS1 modules. The 5v and GND lines should be connected to a +5 v source (Vdd) and ground (Vss) respectively (for both Tx and Rx). The BS1 on the receiving end should remain connected to the computer to view the DEBUG statements.

```
' transmit.bs1
'{$STAMP BS1}
'{$PBASIC 1.0}
'Parallax 433.92 MHz RF Transmitter (#27980) Sample Tx Code (BS1)
'Connect I/O Pin 1 of the BS1 to the DATA line on the RF module
'Connect +5v (Vdd) to the 5v line, connect Ground (Vss) to the GND line
'This code will transmit two word sized counters, byte at a time.
'Note the PULSOUT instruction: this helps the receiver sync with
'the transmitter, especially after lapses in communication.
'This code transmits at 2400 baud, inverted.
SYMBOL x = W1
SYMBOL y = W2
Start:
PULSOUT 1, 300 'Sync pulse for the receiver
SEROUT 1, N2400, ("!", B3, B2, B5, B4)
PAUSE 100
x = x + 1
y = y + 1
PAUSE 150
GOTO Start
'-----------------------------------------------------------------------------
'Receive.bs1
'{$STAMP BS1}
'{$PBASIC 1.0}
'Parallax 433.92 MHz RF Receiver (#27981) Sample Rx Code (BS1)
'Connect I/O Pin 1 of the BS1 to the DATA line on the RF module
'Connect +5v (Vdd) to the 5v line, connect Ground (Vss) to the GND line
'This code will look for an "!", then read in four bytes.
'The first two bytes will be the high and low bytes of a word variable (x),
```

```
'the second two bytes will be the high and low bytes of a second word variable (y).
'The values of x and y will be sent out to the DEBUG terminal.
'This code receives at 2400 baud, inverted.
SYMBOL x = W1
SYMBOL y = W2
Start:
SERIN 1, N2400, ("!"), B3, B2, B5, B4
DEBUG x, y
GOTO Start
```



Initial tests were carried out with a serial green screen attached to the top most level. At right, a Parallax RF Radio transmitter and a receiver enables full bidirectional wireless communications with other supercomputers and Parallax Stamp boards outfitted with similar communications capability.

Software Section How does a program get stored in the Stamp? Your code is tokenized (compressed) and stored within the Stamp's EEPROM memory. This memory is non-volatile, meaning it retains its program even without power. Every time the Stamp receives power, it starts running its code starting with the first executable line. This EEPROM can be rewritten immediately, without any lengthy erase cycle or procedure, by simply re-downloading the code from the Stamp editor again. Each location in the EEPROM is guaranteed for 10,000,000 write cycles before it wears out for the BS 1 and BS 2. For clarity, the following chart includes the possible ways to drive the BS1.

```
' SERIN Options of Driving the BS1
' ====================
' Baudmode       Baud
' Value   Symbol Rate Polarity
'
' 0       T2400  2400 TRUE
' 1       T1200  1200 TRUE
' 2       T600   600  TRUE
' 3       T300   300  TRUE
' 4       N2400  2400 INVERTED ***
' 5       N1200  1200 INVERTED
' 6       N600   600  INVERTED
' 7       N300   300  INVERTED
' SEROUT Options
' Baudmode       Baud
' Value   Symbol Rate Polarity
' 0       T2400  2400 TRUE     (always driven)
' 1       T1200  1200 TRUE     (always driven)
' 2       T600   600  TRUE     (always driven)
' 3       T300   300  TRUE     (always driven)
```

```
' 4       N2400  2400 INVERTED (always driven)
' 5       N1200  1200 INVERTED (always driven)
' 6       N600   600  INVERTED (always driven)
' 7       N300   300  INVERTED (always driven)
' 8       OT2400 2400 TRUE     (open drain, driven HIGH)
' 9       OT1200 1200 TRUE     (open drain, driven HIGH)
' 10      OT600  600  TRUE     (open drain, driven HIGH)
' 11      OT300  300  TRUE     (open drain, driven HIGH)
' 12      ON2400 2400 INVERTED (open source, driven LOW) ***
' 13      ON1200 1200 INVERTED (open source, driven LOW)
' 14      ON600  600  INVERTED (open source, driven LOW)
' 15      N300   300  INVERTED (open source, driven LOW)
```

Paralleled Demo Programs

The following paralleled demo programs load into the computers as indicated. Computer 1 is located at the bottom of the cluster. Each computer is an individual with unique rights to perform its tasks and interests, and has the ability to report and discuss what it has accomplished. It can also take some action, such as creating language with P*LAN* and its own piezo speaker.

Computer 1 Network Code

```
{$STAMP BS1}                          ' Computer 1, Network
{$PBASIC 1.0}                         ' Stamp 1 Code
INPUT 0                               ' set pin 0 to input
PAUSE 1000                            ' give other 2 time to wake up
GOSUB Snd                             ' subr sound/light
SEROUT 0,ON2400,("COM2",10,13)        ' cue next
Loop:                                 ' begin main loop
  SERIN  0,N2400,("COM1",10,13)       ' wait for cue
  GOSUB Snd                           ' subr sound
  SEROUT 0,ON2400,("COM2",10,13)      ' cue next
GOTO Loop                             ' again
Snd:                                  ' Make sound
  B0 = 30                             ' Sound Delay
  B1 = 120                            ' Tone
  SOUND 7, (B1,B0)                    ' Sound Peizo speaker
  PAUSE 500                           ' duration
RETURN                                ' Back
```

```
' {$STAMP BS1}                        ' Computer 2, Network
' {$PBASIC 1.0}                       ' Stamp 1 Code
' Open source mode, port resis to gnd.
' Com2 transmits if cued by Com1.
' Three COMs are on a network.
' Illustrates how all three COMs
' can talk to each other in Token Ring
' style.
INPUT 0                             ' set pin 0 to input
Loop:                               ' begin main loop
SERIN 0,N2400,("COM2",10,13)        ' wait for cue
GOSUB Snd                           ' subr to make sound
SEROUT 0,ON2400,("COM3",10,13)      ' cue next
GOTO Loop                           ' again
Snd:                                ' Make sound
B0=20                               ' Sound Delay
B1 = 110                            ' Tone
SOUND 7, (B1,B0)                    ' Sound Peizo speaker
PAUSE 500                           ' duration
RETURN                              ' End of subroutine
```

```
' {$STAMP BS1}                      ' Computer 3, Network
' {$PBASIC 1.0}                     '
' Open source mode, port resis to gnd.
```

```
' COM3 transmits if cued by COM1.
' Three COMs are on a network.
' Illustrates how all three dimensions
' can talk to each other in Token Ring
' style.
INPUT 0                          ' set pin 0 to input
Loop:                            ' begin main loop
SERIN 0,N2400,("COM3",10,13)     ' wait for cue
GOSUB Snd                        ' subr to make sound/light
SEROUT 0,ON2400,("COM1",10,13)   ' cue next
GOTO Loop                        ' again
Snd:                             ' Make sound
B0=10                            ' Sound Delay
B1 = 100                         ' Tone
SOUND 7, (B1,B0)                 ' Sound Peizo speaker
PAUSE 500                        ' duration
RETURN                           ' End of subroutine
```

Piezo Speaker Test Programs

```
' Baby Stamp Supercomputer
' Piezo test program - Machine Sounds Quantized Interlude
' by Dr. Humanoido  V1.0  02.28.09
' Load into computers 1 through 10, Piezo speaker is on pin 7
' {$STAMP BS1}
' {$PBASIC 1.0}
'END                        ' Silence the speaker
Mains:                      ' Main Code
SOUND 7, (B0,B1)            ' Sound off the Peizo speaker
FOR B0 = 1 TO 10            ' Variable B0 sound 20 interations
SOUND 7, (B1,B0)            ' Sound off the Peizo speaker
FOR B1 = 50 TO 100 STEP 10  ' B1 duration
SOUND 7, (B1,B0)            ' Sound off the Peizo speaker
PAUSE 10                    ' delay for sound and light
SOUND 7, (B1,B0)            ' Sound off the Peizo speaker
NEXT B1                     ' Next duration
SOUND 7, (B1,B0)            ' Sound off the Peizo speaker
NEXT B0                     ' Next interation
SOUND 7, (B1,B0)            ' Sound off the Peizo speaker
GOTO Mains                  ' Loop
```

```
' Second speaker test program – note special wiring!
' This program makes the stereotype computer sound.
' Have a ~40 ohm speaker connected to pin 7 via a 10-100uf cap.
' The other side of the speaker should be grounded.
' {$STAMP BS1}
' {$PBASIC 1.0}
symbol RND = w0
symbol note = b2
symbol spkr = 7
noise:      random RND  'Generate next random in rnd
note = RND & $7F        'Get lower 7-bits rnd into note, insures tone - not white note
SOUND spkr,(note,7)     'Output sound (note, duration) to spkr
GOTO noise              'loop to noise to do again, run three hours for maximum effect
```

Piezo Language (PLAN) Expanded Option The effectiveness of using a piezo speaker is really amazing – it draws only about 1 ma or less and it's ideal for battery operated devices and the current- limited capacities of stamp pins in combination. Its low cost and commonality makes it ideal for many uses. In an aggregate of ten stamps, the piezo overhead is only ten milliamps or less. Communicating with one speaker on each computer is possible using a series of tones with timing and number. The invention of a new audible piezo language was inevitable.

In the simplest form for the 1st test program in Piezo Language (PLAN), a one to one relationship is established between the computer number location and the number of beeps. Computer one at lowest level gets one beep as an

identifier, while computer 10 at uppermost level gets ten fast beeps. PLAN is an effective workable interpretive language for Stamp supercomputers. PLAN can be used for debugging, communicating with sound, sending signals, creating alerts, forms of sound talk, indicators of operations, various representations, audible key responses, and hardware flags.

PLAN has three classes of commands with a total number of 15 specific functions. For example, a quiver communiqué uses an added vibrato to the note. Sliders can go up or down as indicators. Warbles are used to create repeating up and down sounds. Assigning functions to the various elements of PLAN is the fun part. The following is a list of some basics and elements of programming in PLAN language using the BS1. PLAN must fit into a small share of the EEPROMs 256 bytes, to have room left over for application code. This is accomplished by using a lookup table of code, and placing the code snippets into the app. Below is a general outline of the capabilities during development followed by some code snippets for actual use.

PLAN Command List

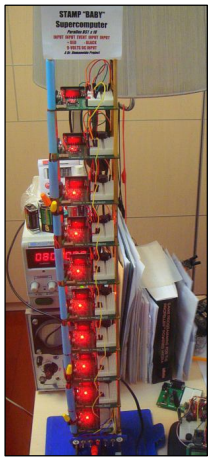
Audible Command List

BASIC	(singular occurrence)
SCALE	(chromatic scale)
SLIDE	(congruent sound to up, or down)
JINGLE	(short sound byte of familiar jingle)
WARBLE	(repeating up down frequency of two notes)
QUIVER	(added vibrato to a note)
STRING	(connects commands)
Representations	
SHOW	(computer location in supercomputer)
BEGIN	(chrome up)
OFF	(chrome down)
TX	(Transmitting (bips))
RX	(Receiving (bips))
Hardware Control	
SW1	(Switch Enabled (1 through 4))
SW0	(Switch Disabled (1 through 4))
PUSH	(Pushbutton Activated)

Notes- * this code uses commands particular to only the BS1 and is not available on the BS2 * piezo speakers from one to the next may have slight differences in resulting pitch and loudness and code may need adjusting.

Sound,(tone,duration,...tone, duration) Sound 0,(50,100) Sends a tone (50) for a duration of 100 out of pin 0. The best tone range to use with small speakers is 30 to 120. Speakers The duration can be from 0 to 255. Multiple tones may be programmed into one command set. Sound 0,(70,100,0,100,120,100) Note that the first and third sound sets produce a tone of 70 for a duration of 100 and 120 for a duration of 100 while the second set produces silence for a duration of 100. Using a tone of 0 produces no sound or a pause in sound between the first and third sound sets. The following is an approximate music scale: C 35, C# 40, D 44, D# 49, E 53, F 58, F #60, G 65, G# 68, A 72, A #75, B 78, C 81

Applications Section: Ideas and Details Apps may include adjunct expansion to the original Basic Stamp Supercomputer, a stand alone baby supercomputer model, cluster platform for developing powerful new projects, a ten part chorded symphonic tone orchestra (MIDI apps), a powerful sensing unit for up to 60 sensors which can be simultaneously driven in sets of ten, home control, and many robotics applications – or just go for the pure fun and educational value of experimenting with your own hobby supercomputer!



The appearance of twinkle power lights during the first startup was an unexpected surprise, yet normal response. You may notice at certain times the power LED blinks very briefly about every two and a half seconds. This is normal behavior. Refer to Application 1 or the END instruction in the Stamp manual or online help file for more information.

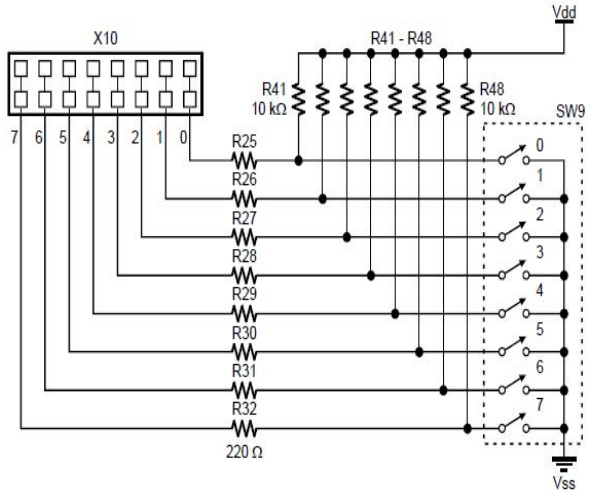
Application 1: Twinkle Power Lights!

When I first powered up the SEED, it looked like a twinkling Christmas tree! I didn't know why until further study. Here's how it works. The Basic Stamp 1s output pins will toggle briefly when using SLEEP, NAP or END. 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

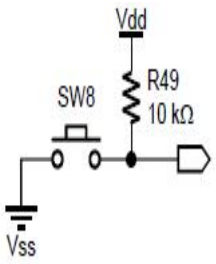
properly. The SLEEP and NAP commands also utilize the watchdog timer to periodically (every 2.3 seconds to be exact) "wake-up" the BASIC Stamp from its low-power mode. Upon reset, the I/O pins are set to inputs for approximately 18 ms before returning to their previous directions and states. If you have 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 because the interpreter chip enters a low-power state. To see the twinkling effect, load up the computers with the following program and run it, observing the power lights.

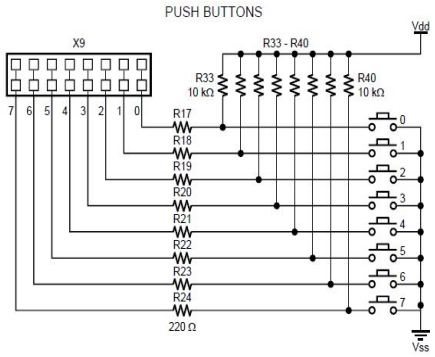
```
' twinkle.bs1
' {$STAMP BS1}
' {$PBASIC 1.0}
END
```

Application 2: Adding Push Buttons and Toggle Switches Schematics show a wad to add several pushbuttons and toggle switches. This is one good way to add a keypad or more data switches.



Left- This schematic, from the Parallax Professional Development board, illustrates how to connect eight toggle switches or pushbuttons (see below). The scheme is easily adapted to the BS1. Note, for the Stamp 1 Project board, the 220 ohm resistors are already in place.





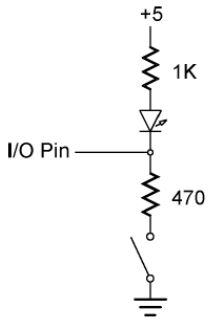
Application 3: Build a Jacob's Ladder of Sound! Program a repeating rising column of sound like the rising repeated electricity seen on a Jacob's Ladder science experiment. Instead of electricity, we'll use sound! Begin with the first Stamp board. Sound the piezo speaker. Stop. Repeat with the second board. Continue to the top board. Repeat. The sound can be varied from one computer level to the next. Try creating some special effects sound to make it more like the sound of an electrical Jacob's Ladder. For more effect, add LEDs.

Application 4: Solving the Riddle of Pullup and Pulldown Say you have a switch connecting a Stamp I/O pin to ground when it's closed. What's the voltage on the I/O pin when the switch is open? The answer is "I don't know". The I/O pin isn't connected really to anything at that point. As a result, the voltage on the I/O pin will be influenced by static charges in the vicinity, by induced 60Hz hum from power cords and wiring in the walls, and by other sources of electrical and magnetic noise nearby. A pullup resistor provides a known voltage to an I/O pin. Since the input circuit draws very little current (microAmps), the resistor can be high in value (10K to 100K typically) and connects the I/O pin to the supply voltage of the chip (+5V in this case). When the switch is actually closed, the high value limits the current through the switch and the switch can very easily bring the voltage of the I/O pin very very close to ground. You can reverse the roles of the switch and pullup to make a switch and pulldown resistor with the switch connecting the I/O pin to the supply voltage and the resistor providing a known zero voltage to the I/O pin when the switch is open. It's often easier to connect the switch to ground, so most circuits of this sort use a pullup resistor.

Application 5: Using a Pin for both Input and Output - Sharing Pins (reference NV38) From time to time, we'll run out of pins before we run out of code space. This is especially true with the BS1. With careful code writing and I/O hardware, we can share one or more pins. The trick is modifying the Dirs variable mid-stream. We don't normally do this, so we need to be cautious. Take a look at the circuit below.

Circuit to display status of I/O pin - detect switch input, output to LED

This circuit allows us to display the status of the I/O pin when it's an output, and to read it when it's configured as an input. Notice that this circuit is configured for negative logic. What this means is that a low output will light the LED, a high output will turn it off. Don't be alarmed by this; you can easily change positive logic (HIGH = On) to negative logic with the ^ (Exclusive OR) operator (0 ^ 1 is 1). Okay, let's use this circuit. Program Listing 38.1 is a very simple program that waits for you to push the button connected to Pin0, then flashes the LED five times. Notice that the Dirs variable is reset to configure the pin before each section. In case you're wondering, the purpose of the 470-ohm resistor is to protect the I/O pin in case you push the button when the pin is in an output condition. If you pressed the button when the Stamp had placed a HIGH (5 volts) on the pin, you'd have a short to ground. The resistor limits the current through this short to a safe level. For another good example of pin sharing, refer to the Stamp Application Note #1, "LCD User-Interface Terminal."

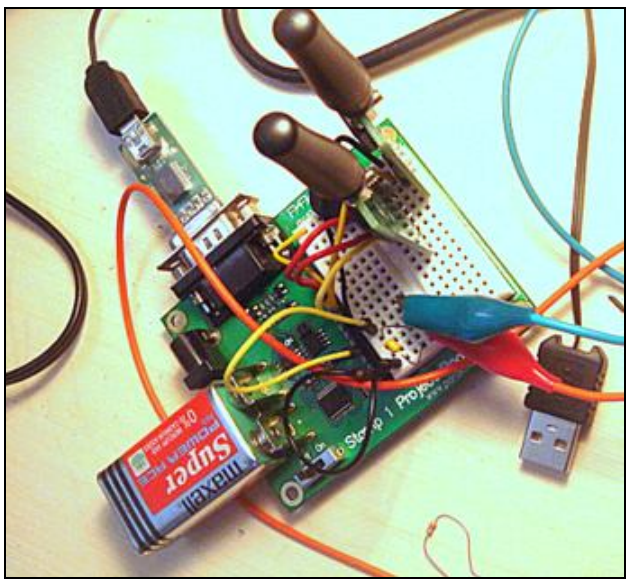


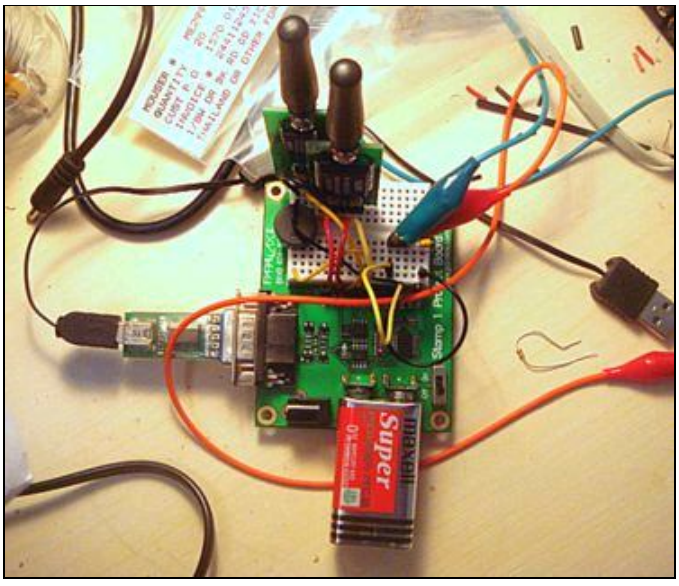

```
' Program Listing 38.1
' Reference- Nuts & Volts: Stamp Applications, April 1998
GetPin: Dirs = %00000000 ' all pins are inputs
IF Pin0 = 1 THEN GetPin ' wait for pin to go low
Blink: Dirs = %00000001 ' make pin 0 an output
FOR B0 = 1 TO 5
PAUSE 500
Pin0 = 0 ' LED on
PAUSE 500
Pin0 = 1 ' LED off
NEXT
GOTO GetPin ' do everything again
```

Application 6: Create a Physically Rising Chromatic Scale Assign one of ten notes to computer one through ten. Activate each piezo speaker in succession, one at a time, to create a chromatic scale that rises from computer 1 to computer 10.

Application 7: Create Ten-Note Chords Assign one note to each of ten computers and sound off for a couple seconds. It's possible to created sophisticated music.

Application 8: Build a Bstat Base Station The base station communicates with the SEED supercomputer to send and receive commands and information. Its range is approximately within a full city block. The station includes a BS1 or BS2 processor, board, keyboard, display, and a Parallax 433 Mhz Transmitter/Receiver pair for bidirectional wireless communications in serial format.



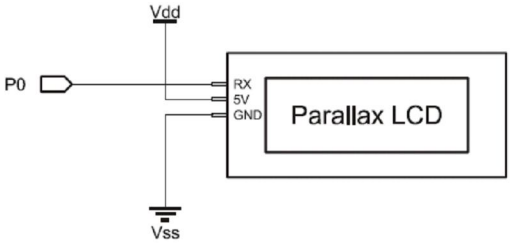


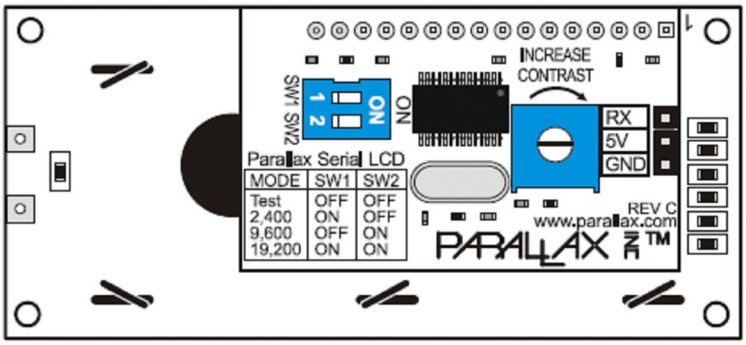
This is the development of the first *Bstat* Base Station. The transceiver is wired and tests are in progress. This platform is multi-purposed, handling the development of Pin 1 enumeration for the Stamp Seed Supercomputer. Yellow wires are signal, black is ground, and red is +5V.

Application 9: Add a Parallax 2x16 Serial Green Screen

A Parallax 2x16 Serial LCD (Non-Backlit) tem code 27976 is added with a simple one wire connection to pin 6 on computer board #2. Make wire connections to Vss and Vdd on the computer board. Set the LCD to 2400 baud. This display provides basic text wrapping so that your text looks right. You have full control over all of its advanced LCD features, allowing you to move the cursor anywhere on the display with a single instruction and turn the display on and off in any configuration. It directly supports ASCII Dec 32-127, the same visible characters as the BASIC Stamp Editor's Debug Terminal. In addition, you may define up to eight of your own custom characters to display anywhere on the LCD. This display and many applications and programming examples are featured in Smart Sensors and Applications Parts & Text Kit (#28029).

- Clear 2x16 character display, directly supports ASCII DEC characters 32-127
- Eight user-definable custom characters, cursor moves with single command
- Baud mode selector/ adjustable contrast on the back of the display
- Power requirements: +5 VDC, 20 mA
- Selectable asynchronous serial baud rates: 2400, 9600, 19200
- Dimensions: Approx. 1.5 x 3.15 in (38 x 80 mm)
- Operating temp range: -4 to +158 °F (-20 to +70°C)
-





The screen is easily added to the top of the Skyscraper (see photo), however, if using the screen for more function than just debugging and setup, it is recommended to att the screen to the bottom, directly above the base, using the two angle iron mounts (remove the wire clamps).

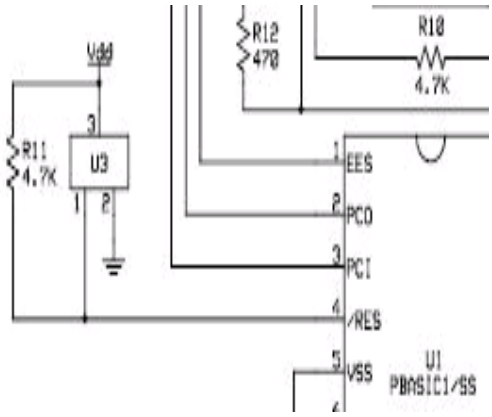
Application 10: Add a Parallax 2x16 Parallel Green Screen

A Parallax 2x16 parallel LCD (Non-Backlit) Item code 603-00006 is a better choice in terms of minimal power draw, less than 1 ma according to the Hitachi specifications sheet. The HD44780 2x16 Parallel LCD can be controlled via a 4-bit or 8-bit parallel interface. It allows you to display text, number data and up to 8 custom created characters. It includes a 6-inch ribbon cable with 14-pin connector and plugs into a 2x7 space on the solderless breadboard. Power Requirements: 5 VDC, communication: 4-bit or 8-bit Parallel Interface, dimensions: 3.25 x 1.75 x 0.25 in (85 x 45 x 6 mm), operating Temperature: -32 to +158 °F (0 to +70 °C)

Application 11: Demonstrating the Hidden Low Power Mode

I like to experiment with Stamps and build Stamp projects. Last night while working on a new project with a BS1 Project Board, the 9-volt battery went down to almost nothing, however the Stamp continued to work and function properly. I first noticed the battery condition by the dim power LED. The battery was reading 1.2 volts under load.

In this application, the power is reduced to a mere 2.2 volts and the entire board collective continues to operate code correctly to sound off all piezo speakers in unison. Note the Stamp SEED Supercomputer has all power LEDs lit and functioning. Right- U3 brownout detector.



At that time, I went over to the new SEED Stamp Supercomputer and attached a Bench Power Supply with adjustable voltage. All ten processors were powered up. Starting from 9-volts, power was backed off gradually. A program was running which kept all ten piezo speakers making lots of noise. Each power LED was lit, and the stamp collective continued to perform well down to 2.2 volts. The amps draw was at elevated to 181 ma. As the volts were backed off to 2.1, one of the boards fell off and its power LED extinguished. At 2.0, then 1.8, two more boards fell out of the collective. The experiment was repeated and this time the processors fell out at 2.0 volts. I have the revision B Stamp 1 Project Board and the latest schematic for revision C. Both schematics (rev b and rev c) show the brownout detector in the PIC16C56. However, as you can see in the microscope's photo, every reset pin on every project board in the Stamp Seed Supercomouter (Rev B) has no brownout detector, and no connection to pin 4 (res). Leaving off the brownout detector on the BS1 is a huge advantage and a good judgment call by Parallax Engineers.

Application 12: Long Term Memory Using EEPROM

```
' The following WRITE command stores the value 245 at location 100:
WRITE 100, 245
```

The EEPROM is organized as a sequential set of byte-sized memory locations. The WRITE command normally only stores byte-sized values into EEPROM. This does not mean that you can't write word-sized values, however. A word consists of two bytes, called a low-byte and a high-byte. If you wanted to write a word-sized value, you'll need to use two WRITE commands and a word-sized value or variable. For example,

```
SYMBOL value = W0
SYMBOL valLo = B0
SYMBOL valHi = b1
value = 1125
WRITE 0, valLo ' write low byte
WRITE 1, valHi ' write high byte
```

WRITE location,data: Store data in EEPROM location. • Location is a variable/constant (0–255) that specifies which EEPROM location to write to. • Data is a variable/constant (0–255) that is stored in the EEPROM location. The EEPROM is used for both program storage (which builds downward from address 254) and data storage (which builds upward from address 0). To ensure that your program doesn't overwrite itself, read location 255 in the EEPROM before writing any data. Location 255 holds the address of the first instruction in your program. Therefore, your program can use any space below the address given in location 255. For example, if location 255 holds the value 100, then your program can use locations 0–99 for data. Sample Program:

```
read 255,b2          ' Get location of last program instruction.
loop:
b2 = b2 - 1          ' Decrement to next available EEPROM location
serin 0,N300,b3      ' Receive serial byte in b3.
WRITE b2,b3          ' Store received serial byte in next EEPROM location.
if b2 > 0 then loop ' Get another byte if there's room.
```

WRITE Location, DataItem: Write DataItem into Location in EEPROM. On the BS2 family, multiple DataItems may be written to consecutive locations. Location is a variable/constant/expression* (0 - 255 on BS1, 0 - 2047 on all other BASIC Stamp modules) that specifies the EEPROM address to write to. DataItem is a variable/constant/expression* specifying the value to be stored. Note: Expressions are not allowed as arguments on the BS1. Range of EEPROM locations = 0 to 255. Maximum number of writes per location = 10 million.

On the BS1, location 255 holds the address of the last instruction in your program. Therefore, your program can use any space below the address given in location 255. For example, if location 255 holds the value 100, then your program can use locations 0–99 for data. You can read location 255 at run-time or simply view the Memory Map of the program before you download it.

The EEPROM is used for both program storage (which builds downward from address 255 on BS1, 2047 on all other BASIC Stamps) and data storage (which builds upward from address 0). The **WRITE** instruction stores a value to any EEPROM address. Any location within the EEPROM can be written to (including your PBASIC program's locations) at run-time. This feature is mainly used to store long-term data to EEPROM; data stored in EEPROM is not lost when the power is removed.

The following **WRITE** command stores the value 245 at location 100:

```
WRITE 100, 245
```

The EEPROM is organized as a sequential set of byte-sized memory locations. With the BS1, the **WRITE** instruction only stores byte-sized values into EEPROM. This does not mean that you can't write word-sized values, however. A word consists of two bytes, called a low-byte and a high-byte. If you wanted to write a word-sized value, you'll need to use two **WRITE** commands and a word-size value or variable (along with some handy modifiers). For example:

```
SYMBOL  value   = W1                    ' word-sized variable
SYMBOL  valLo   = B2                    ' B2 is the low-byte of W1
SYMBOL  valHi   = B3                    ' B3 is the high-byte of W1

Main:
  value = 1125

  WRITE 0, valLo
  WRITE 1, valHi
  END
```

On the BS1, location 255 holds the address of the last instruction in your program. Therefore, your program can use any space below the address given in location 255. For example, if location 255 holds the value 100, then your program can use locations 0-99 for data.

Demo Program (WRITE.bs1)

```
' WRITE.bs1
' This program writes a few bytes to EEPROM and then reads them back out
' and displays them in the Debug window.
' {$STAMP BS1}
' {$PBASIC 1.0}
SYMBOL addr = B2 ' address
SYMBOL value = B3 ' value
Main:
WRITE 0, 100 ' write some data to locations 0 - 3
WRITE 1, 200
WRITE 2, 45
WRITE 3, 28
Read_EE:
FOR addr = 0 TO 3
READ addr, value ' read value from address
DEBUG #addr, ": ", #value, CR ' display address and value
NEXT
END
```

EEPROM differs from RAM, the memory in which variables are stored, in several respects: 1. Writing to EEPROM takes more time than storing a value in a variable. Depending on many factors, it may take several milliseconds for the

EEPROM to complete a write. RAM storage is nearly instantaneous. 2. The EEPROM can only accept a finite number of write cycles per location before it wears out. Table 5.123 indicates the guaranteed number of writes before failure. If a program frequently writes to the same EEPROM location, it makes sense to estimate how long it might take to exceed the guaranteed maximum. For example, on the BS2, at one write per second (86,400 writes/day) it would take nearly 116 days of continuous operation to exceed 10 million. 3. The primary function of the EEPROM is to store programs (data is stored in leftover space). If data overwrites a portion of your program, the program will most likely crash. Check the program's memory map to determine what portion of memory your program occupies and make sure that EEPROM writes cannot stray into this area.

READ location,variable: Read EEPROM location and store value in *variable*. • Location is a variable/constant (0–255) that specifies which location in the EEPROM to read from. • Variable receives the value read from the EEPROM (0–255). The EEPROM is used for both program storage (which builds downward from address 254) and data storage (which builds upward from address 0). To ensure that your program doesn't overwrite itself, read location 255 in the EEPROM before writing any data. Location 255 holds the address of the last instruction in your program. Therefore, your program can use any space below the address given in location 255. For example, if location 255 holds the value 100, then your program can use locations 0–99 for data. Sample Program:

```
READ 255,b2 ' Get location of last program instruction.
loop:
b2 = b2 - 1 ' Decrement to next available EEPROM location
serin 0,N300,b3 ' Receive serial byte in b3
write b2,b3 ' Store received serial byte in next EEPROM location
if b2 > 0 then loop ' Get another byte if there's room left to store it.
```

It's likely that circuits with 9V batteries will last far longer without the detector, because of the Microchip rating down to a mere 2 volts. It also opens up new possibilities for running some circuits (the piezo speaker in this example) with a couple tiny AAA batteries. It is also likely two watch cell batteries could power the stamp without any 5V sensor requirements.

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

LPM Discovery! There is a hidden low power mode on Rev B Stamp 1 Project Boards and it was accidentally discovered when running some tests and experiments during the Stamp Seed Supercomputer development. According to the board's schematic, a brownout detector is in place. But apparently after Microchip updated the internal brownout detector, Parallax engineers dropped the IC installment. Since the Microchip will operate down to 2 volts, it opens up many new possibilities for low power operating.

The collective of ten BS1 boards can run in a lower power mode, instead of 9-volts. Using the bench lab power supply, begin with 9-volts and run the piezo speaker test in each board. Now back off the voltage. What is the lowest voltage where all boards continue to function?

This is a great advantage. Circuits will run with less voltage and run longer. When a 9V battery drops to 4V, the board will continue to function. It will, however, not operate add-on sensors with 5V power requirements. This opens up a new chapter in Basic Stamp 1s. It is also possible to operate the boards with tiny batteries such as two AAs or two AAAs, perhaps even watch batteries.

Application 13: Handling RAM as Short Term Memory

The BASIC Stamp I 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 above. The PBASIC 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.

Application 14: Setting the Ports

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 PBASIC programs, you'll use the symbols described above to read and write the BASIC 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 BASIC 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 'Define label "switch" for I/O pin 0
symbol flag = bit0   'Define label "flag" for bit variable bit0
symbol count = b2    'Define label "count" for byte variable b2
```

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 1 module) 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 1 microcontroller) 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 second 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 microcontroller 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 1 microcontroller) 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 that 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 module. OUT0 through OUT15 (or PIN0 through PIN7 on the BASIC Stamp 1 module) 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 1 module) 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. Only pins 5, 3, 2 and 1 will output the voltage set in the OUTS variable because they are the only pins defined as outputs by the DIRS variable. Review the BASIC Stamp manual for more examples of this.

Application 15: Developing Stamp Supercomputer Math

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 x 4 would be solved as 2 + (3 x 4), since multiplication takes priority over addition. The result would be 14. However, since the BASIC Stamp solves expressions from left to right, it would be solved as (2 + 3) x 4, for a result of 20. When writing your programs, please remember that the left-to right evaluation of expressions may affect the results. 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 1 module, 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 cannot 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.

```
Mathematical Functions List
---------------------------
+ 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
```

```
Math 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
```

How are negative numbers handled? Does the BASIC Stamp module handle signed numbers and arithmetic? Yes. The BASIC Stamp microcontroller 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. 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.

Application 16: Mystery of the Vanishing EEPROM
The BS1 is handled very differently from the BS2. Take for example, the EEPROM. If you run one program and write data into the EEPROM, the entire EEPROM is overwritten when you run a second program. The second program will store zeros in the locations where the first program stored data. The entire eeprom is overwritten when you RUN the program. So if those are indeed two different programs, the second one will store 0's in the locatiions where the first one stored 7's. Erasing & Retaining EEPROM Memory. How do I erase the BASIC Stamp module's program space? In the BASIC Stamp 1 module an erase cycle is performed at the beginning of every programming process. Downloading a program will completely set the EEPROM memory spaces to 0 and erase the previous contents. Therefore, any data logging operation into EEPROM memory will require the Stamp to remain on with a program still functioning, in order to retrieve the data. For example, simply run the following code, or any code, to erase EEPROM memory.

```
' {$STAMP BS1}
' {$PBASIC 1.0}
END
```

Application 17: Setting the Format of Pins
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
```

Application 18: Creating a Variable or Pin Alias
In the BASIC Stamp 1 module 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.

Application 19: How to reference a specific bit within a byte or word variable
On the BASIC Stamp 1 module 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).

Application 20: Determining Program Memory
How much space does each PBASIC command take? Each command takes a variable amount of space within the BASIC Stamp module'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 1 module, 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 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. How big of a program can I store in the BASIC Stamp module? The BASIC Stamp 1 module has 256 bytes of program storage; enough for 80 to 100 lines of PBASIC1 code.

Application 21: Performing Memory Dump
The BSAVE function saves the "binary" or byte code image to a file (256 bytes for BS1). Another way is to use a small routine running on the stamp to dump the contents of its own memory out the serial port to your PC to be captured as hex data.

```
symbol index BO
Symbol value B1
For index = 0 to 255
Read index, value
Debug value
Next
```

You just need to figure out a way to trigger this routine, so that it will not affect the normal operation of your program. Also the Editor can be used to see a picture view of the entire memory contents.

Application 22: Transmitting and receiving serial messages between BS1 computers
If you connect two BS1 boards as follows:

vss - vss
vdd - vdd
vin - vin
pin0 - pin0
vss to 1K R to pin 0 (inverted)

then you can send serial messages back and forth. In the most simple form, the transmitter and receiver programs, loaded one into each BS1, are very effective. Load the programs, power down, then power on at the same time and monitor the receiving end on the debug screen. The message is a text “g” and is sent out and received at 2400 baud inverted.

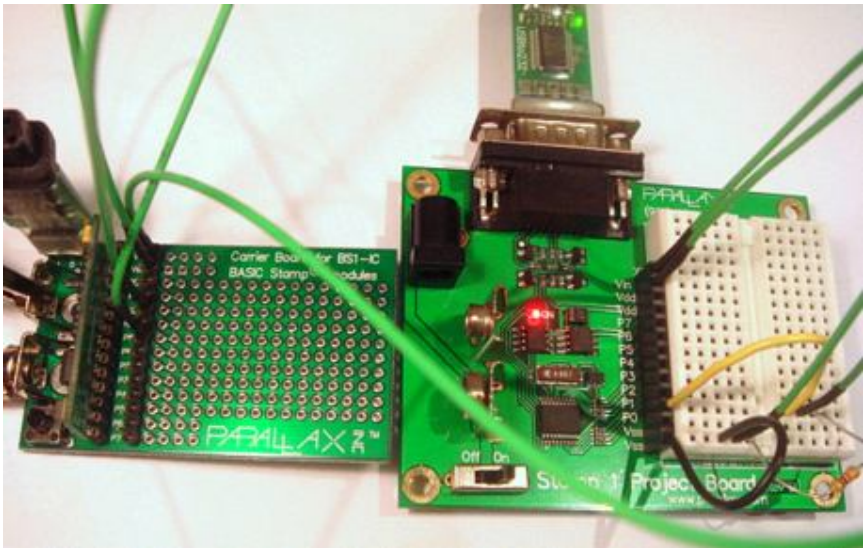
```
' {$STAMP BS1}
' {$PBASIC 1.0}
' Serial Transmitter
PAUSE 5000
SEROUT 0, ON2400, ("g",10,13)
END

' {$STAMP BS1}
' {$PBASIC 1.0}
' Serial Receiver
SYMBOL x = B0
SERIN  0,N2400, x
DEBUG #@x
```

Syntax & Function Study
Another program for syntax and function study:

```
' SERIN.BS1
' This program waits for the characters "A", "B", "C", and "D" to arrive
' at the serial input (inverted, 2400 baud, N81), followed by a number,
' then a carriage-return or some other non-number character.  The number is
' then displayed in the Debug window.
' {$STAMP BS1}
' {$PBASIC 1.0}

SYMBOL  SIn             = 0
SYMBOL  Baud            = N2400
SYMBOL  result          = W1
Main:
  SERIN SIn, Baud, ("ABCD"), #result
  DEBUG #result, CR
  GOTO Main
  END
```

Application 23: Build a Minuscule Stamp Supercomputer

Pump up the array with your BS1 stamp collection! This is a preview of the new Minuscule Stamp Supercomputer, comprised of only two BS1 boards. It's main purpose is for testing various software in developing the Stamp SEED Supercomputer, without the need to load ten programs into ten processors. The two core Minuscule needs no tower and the boards can set next to each other on a desk next to the PC. The provided transmitter and receiver software is minuscule and serially enables one core to text communicate with the other at 2400 baud. The application is included with the Stamp SEED Supercomputer because most of the applications, principles and programming are the same. Plus, the Minuscule has no manual, so it gets its own featured space.



Introducing the *Minuscule Stamp Supercomputer!* It's a perfect example of multiple BS1 boards connected together for sending and receiving serial signals (text) according to the transmit and receive programs listed below. The 2X boards chosen for this Minuscule Stamp Supercomputer, MSS, include the Carrier Board with a BS1-IC Basic Stamp Module, and the Stamp 1 Project Board Rev. B. Both boards are programmed through the PC's USB port using Parallax adapters, a BS1 Serial Adapter, and a USBto232 adapter.

```
' {$STAMP BS1}    ' Minuscule Stamp Supercomputer
' {$PBASIC 1.0}   ' Serial Transmitter – Send a Text Character
PAUSE 5000        ' Load into computer 1
SEROUT 0, ON2400, ("g",10,13) ' Transmit a text character + CR + LF
END               ' Quit

' {$STAMP BS1}    ' Minuscule Stamp Supercomputer
' {$PBASIC 1.0}   ' Serial Receiver – Receive a Text Character
SYMBOL x = B0     ' Load into computer 2
SERIN  0,N2400, x ' Receive one text character
DEBUG #@x         ' Show text on the debug screen
END               ' Quit
```

There is no schematic for the MSS because the connections are so simple. Just follow the chart below using two BS1 boards and some wire leads. The only component required is a 1K ohm resistor.

vss - vss
vdd - vdd
vin - vin
pin0 - pin0
vss to 1K R to pin 0 (inverted)

Build the MSS and have lots of fun testing all your new code! Remember to leave some comments about your experiences at the Parallax Project Forum.

Application 24: Serial Communications by Computer Name

This program set will communicate by computer name. The transmitter sends a code equal to a computer name, along with data numbers. If the other computer recognizes its own code, the data numbers will be received.

```
' This program transmits the string "ABCD" followed by a number and a
' carriage-return at 2400 baud, inverted, N81 format.
' {$STAMP BS1}  ' Transmitter
' {$PBASIC 1.0}
SYMBOL  SOut            = 0
SYMBOL  Baud            = N1200
SYMBOL  value           = W1
Setup:
```

```
value = 1
Main:
  SEROUT SOut, Baud, ("ABCD", value)
  value = value + 1
  PAUSE 250
  GOTO Main
  END
--------------------------------------------------------------------------------
' SERIN.BS1     ' Receiver
' This program waits for the characters "A", "B", "C", and "D" to arrive
' at the serial input (inverted, 2400 baud, N81), followed by a number,
' then a carriage-return or some other non-number character.  The number is
' then displayed in the Debug window.
' {$STAMP BS1}
' {$PBASIC 1.0}
SYMBOL  SIn             = 0
SYMBOL  Baud            = N1200
SYMBOL  result          = W1
Main:
  SERIN SIn, Baud, ("ABCD"), result
  DEBUG result
  GOTO Main
  END
```

Application 25: One Talks and many Will Listen

Using the transmitter program, seen above, loaded into one stamp – and the receiver program loaded into the remainder of computers, you can make one a talker and all the others (or as many Stamps as you want) to do the listening.

Application 26: Number Talking and Listening

The following two programs are capable of sending and receiving a number. This can be used for transferring data from one computer to another or many. With one number talker program, all or some of the other computer may run listener programs at the same time. But only one talker program is allowed, i.e. talker programs cannot talk at the same time.

```
' {$STAMP BS1}  ' number_talker.bs1    transmits a number
' {$PBASIC 1.0} ' 1200 baud, inverted, N81 format
SYMBOL  SOut   = 0     ' serial interface on pin 0
SYMBOL  Baud   = N1200 ' communicate inverted 1200 baude
SYMBOL  id     = W1    ' computer id number
id = 123               ' one value for id
Main:
SEROUT SOut, Baud, (id)
PAUSE 250
GOTO Main
--------------------------------------------------------------------------------
' {$STAMP BS1}          ' number_listener.bs1  receive number & display it
' {$PBASIC 1.0}         ' inverted, 1200 baud, N81
SYMBOL  SIn     = 0     ' serial communications on pin 0
SYMBOL  Baud    = N1200 ' 1200 baud
SYMBOL  result  = W1    ' the received number
Main:                   ' main loop
SERIN SIn, Baud, result ' receive the number
DEBUG result            ' display the number
GOTO Main               ' loop
```

Application 27: Direct Addressing by Personal Name

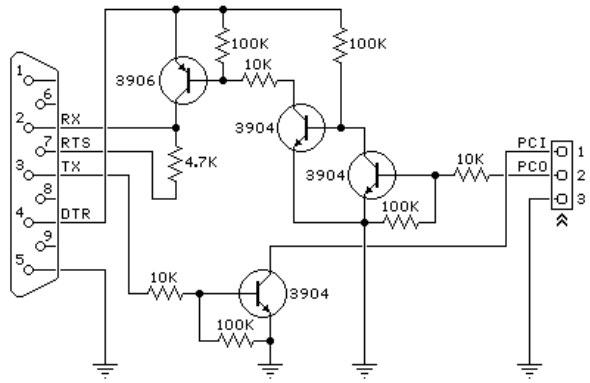
You can use serout code in one program to directly call out to another computer, one out of many, by personal name. In these two programs, one computer will call CHIP and send some work data. CHIP will hear his name, respond by saying he is CHIP, and then show the work data he received.

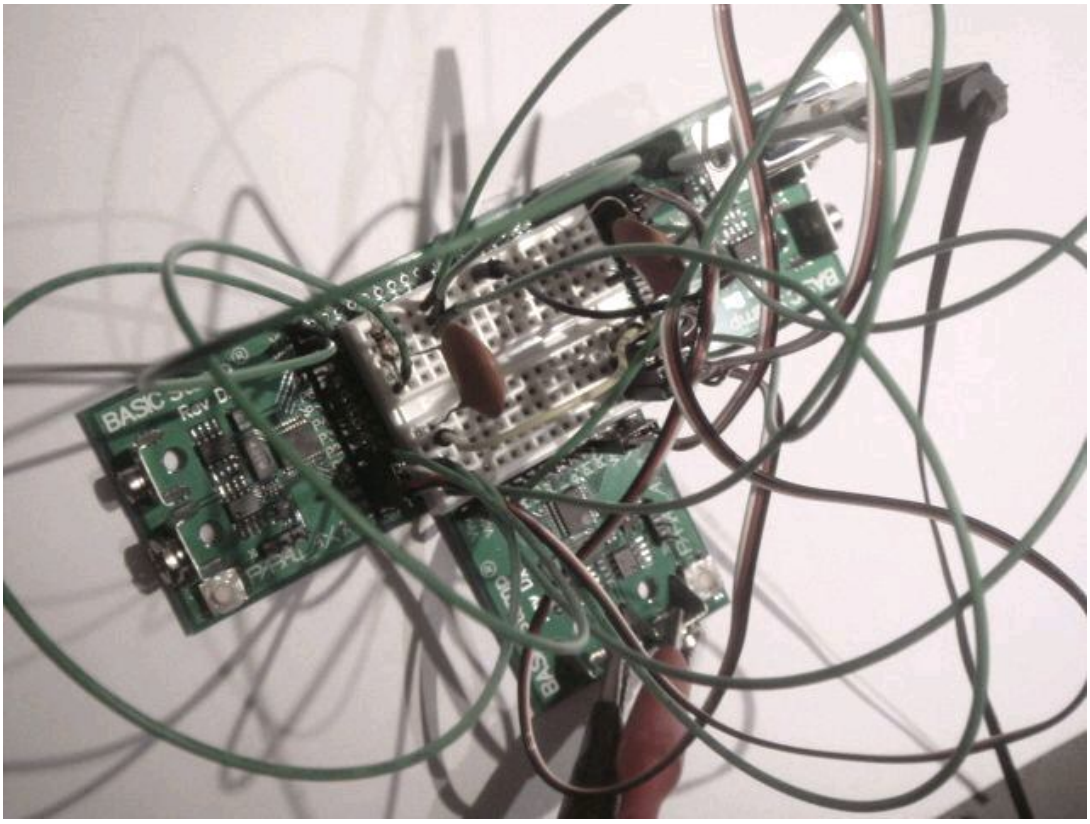
```
' {$STAMP BS1}  ' chip_talker.bs1  transmits computer name "Chip" followed by numbers
' {$PBASIC 1.0} ' 2400 baud, inverted, N81 format
SYMBOL  SOut            = 0
```

```
SYMBOL  Baud            = N1200
SYMBOL  value           = W1
Main:
  SEROUT SOut, Baud, ("CHIP", value) ' CHIP, are u there?
  value = value + 1
  PAUSE 250
  GOTO Main
  END
-------------------------------------------------------------------------------------
' {$STAMP BS1}  ' chip_listener.bs1  Chip waits for his name "C", "H", "I", and "P"
' {$PBASIC 1.0} ' and will display his response. (inverted, 2400 baud, N81)
SYMBOL  SIn             = 0
SYMBOL  Baud            = N1200
SYMBOL  result          = W1
Main:
  SERIN SIn, Baud, ("CHIP"), result ' wait to hear the name CHIP
  DEBUG "CHIP here ", result        ' It's me! These are numbers you sent.
  GOTO Main
  END
```

Application 28: Build your own Basic Stamp One Serial Interface

Below is the schematic for a three pin to DBM serial interface. Starting with Version 2.1, the BASIC Stamp Editor/ Compiler supports programming BASIC Stamp 1 modules through a standard serial connection. Programming BS1 modules via the serial connection requires an adapter to handle the voltage level and mechanical requirements of the BS1. For convenience, Parallax manufactures an adapter that plugs into the end of a serial cable and connects to the programming header of the BS1. The BS1 Serial Programming Adapter is Parallax part #27111.

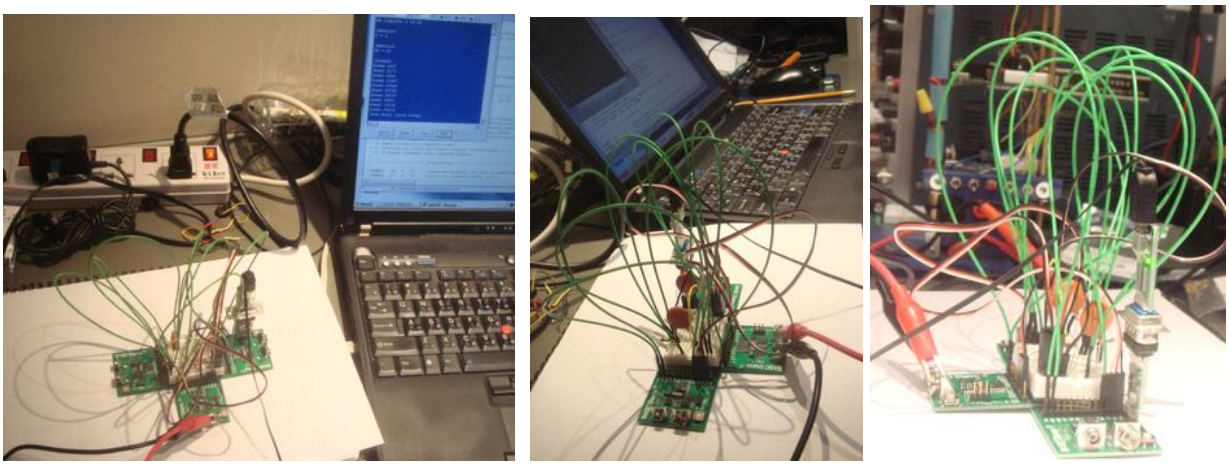




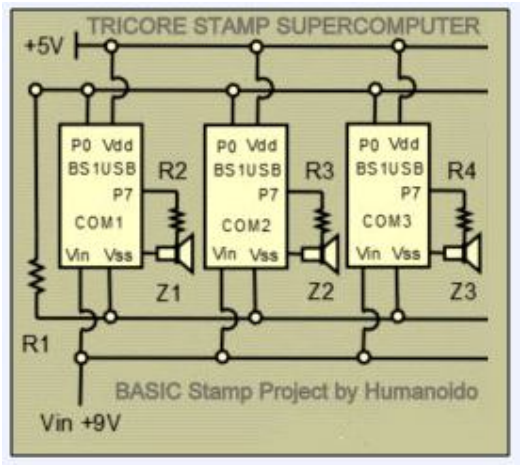
Application 29: Build a TriCore Stamp Super computer

Two cores are not be enough for testing and developing more features of SEED. The TSS was made to provide three static cores for communications testing across a one line interface. To build a TriCore, take three Basic Stamp Rev Dx boards and attach one solderless breadboard with tape. Put two boards together end to end and tape both sides.

Set the third board, with solderless breadboard, on top as illustrated. Affix with tape. Wire all vss to vss, vdd to vdd, vin to vin, and pin 0s together. The deterministic pin circuit is built the same as for the Stamp SEED Supercomputer using the number one pins. The STS is capable of running the SEED Artificial Intelligence program.

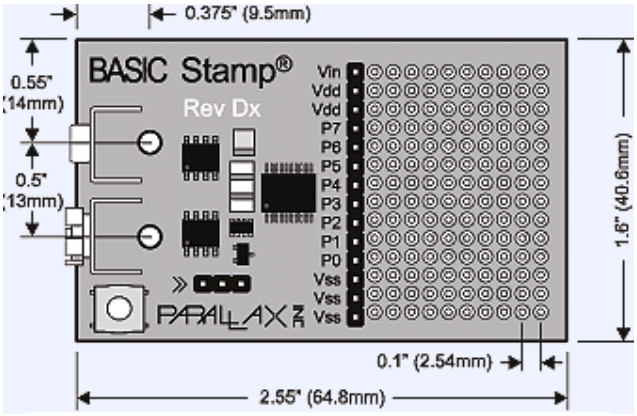


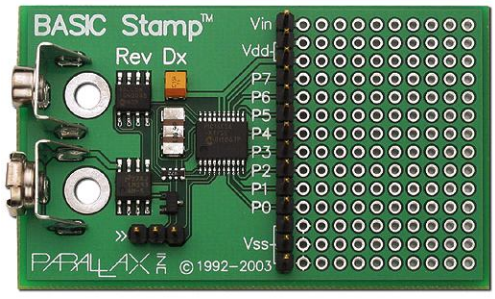
Views of the Stamp TriCore Supercomputer. The TriCore is used for testing and more rapid development of SEED. Its wiring is similar to the Stamp Seed Supercomputer and runs self enumerating deterministic software. One collection of two boards (a USBto232, and a BS1 Serial Adapter) is moved from board to board during downloads and debugging.



Left- simplified Stamp TriCore Supercomputer STS schematic

TriCore Parts List
3 Basic Stamp Rev Dx Board
1 Solderless Breadboard
3 Wire Cable Leads Female Ends for 3 Pins
8 Wire Cables Male to Female – Green
1 Roll Tape
1 Resistor 1K ohms
2 Jumper Wires
2 Ceramic Disk Capacitor .1uf
1 Resistor 2K ohms
1 BS1 Serial Adapter
1 USBto232 Adapter





Build up three Basic Stamp Rev Dx boards and add a solderless breadboard to make a TriCore.

Self Enumeration pin circuits and parts are listed separately below. * this resistor is already located on the Stamp 1 Project Board.

Self Enumeration Schematic

```
'               *
'              220
' pin 1 <-----^^^^^----- no circuit, open       COMPUTER 1 ID = 00
'           resistor
'
'               *        R2       C2
'              220       1K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 2 ID = 17
'           resistor  resistor  capacitor       brown-black-red
'
'
'               *        R3       C3
'              220       2K       .1uf
' pin 1 <-----^^^^^-----^^^^^-----||----- Vss   COMPUTER 3 ID = 28
'           resistor  resistor  capacitor       red-black-red
'
```

TriCore Supercomputer AI Software Listing

```
' TriCore_AI.bs1
' TriCore AI Artificial Intelligence v10.0
' Stamp AI for the BS1 TriCore Supercomputer
' by Dr. Humanoido July 20, 2009

' ---------------------- Directives --------------------------------

' {$STAMP BS1}             ' Self deterministic
' {$PBASIC 1.0}            ' Self enumerating

' ---------------------- Descriptions ------------------------------

'     Performance of Life Cycle
' 1)  read pin 1 and determine id                        (self enumerating)
' 2)  determine computer number based on id range        (deterministic)
' 3)  remember computer id & number                      (memorize)
' 4)  recall memory of computer id & #                   (remember)
' 5)  nap and pseudo random dream in "Vers Libre"        (nap, dream)
' 6)  sleep fuzzy clock - its computer number in seconds (sleep)
' 7)  wake to send its computer number and id            (wakeup, talk)
' 8)  Listen for others' computer numbers                (listen)
' 9)  if heard, remember other computer numbers and id's (know what is heard)
' 10) do some work, retrieve the other ids...           (work)
' 11) go into suspended animation                       (hibernation)

' ---------------------- Declarations ------------------------------

SYMBOL  result  =   B0    ' (random) Nap & Dream - a random number
SYMBOL   n      =   BIT1  ' (random) bit1 number is 0 or 1 from B0 above
SYMBOL   id     =   B1    ' pin read id number of this computer (0-126)
SYMBOL   c      =   B2    ' determined computer number (1-10)
SYMBOL   v      =   B3    ' id range value for each computer, random dreams counter
SYMBOL   h      =   B4    ' no. of heard/missed coms (never heard - was sleeping)
SYMBOL   x      =   B5    ' slush value for thinking, recycled as needed
SYMBOL  epin    =   1     ' Enumerating pin
SYMBOL  ipin    =   0     ' Rx Tx Interface pin for serial in and out
SYMBOL  ppin    =   7     ' Piezo speaker pin
SYMBOL  baud    =   N1200 ' Baud rate, inverted, n81 format

' ---------------------- Self Enumeration --------------------------
```

```
LOW   epin       ' get id from self enumerate pin, read rc circuit = ID number
PAUSE 2000       ' cap charge & settle time on pin 1
POT   epin,54,id ' pin 1, scale, determine id, read pot (POT PotPin, Scale, read_value)

' ---------------------- Determine Computer Number ------------------

Loop:                                          ' loop for all ten computer
possibilities
LOOKUP c, (12,23,35,45,55,76,95,114,122,134), v ' lookup logged id range values v
c = c + 1                                      ' next range lookup address
IF id >=  v THEN Loop                          ' if id larger than range value, try
again
DEBUG "Hello human!",CR,"Me, Artificial Intelligence Life Form ",CR,"computer ", #c,"id
",#id,CR,CR        ' show computer number and id!

' ---------------------- Memory Exercise ----------------------------

' Memory Map   /eeprom 84% full
' 00             computer id     (enumerating)
' 01             computer number (deterministic)
' 02-21          other computers, alternating id & computer # (see below)

' IDEAL DATALOGGING MEMORY MAP FROM 02 TO 21 (FOR COMPUTER #1 ONLY)
' MEMORY LOCATION  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18
19  20  21
' DATA STORED      ID1 C1  ID2 C2  ID3 C3  ID4 C4  ID5 C5  ID6 C6  ID7 C7  ID8 C8  ID9
C9  ID10 C10

WRITE 0,id                                      ' memorize id number
WRITE 1,c                                       ' memorize computer number
READ 0,id:DEBUG "(my 1st memory)", c,CR,CR      ' show computer number is remembered
READ 1,c :DEBUG "(my 2nd mem)",id,CR,CR,"(Nap Dreams)" ' show computer id is remembered

' ---------------------- Nap & Random Dream -------------------------

' NAP INFO
' =========
' Nap#  Nap Time
' 0     18    ms
' 1     36    ms
' 2     72    ms
' 3     144   ms
' 4     288   ms
' 5     576   ms
' 6     1.152 s
' 7     2.304 s

v = 0                          ' init a new counter
result = id                    ' seed pseudo random # generator with the id
Again:                         ' loop
v=v+1                          ' count the dreams
IF v=11 THEN FuzzyClock        ' do 10 dreams
RANDOM result                  ' randomize a byte
BRANCH n, (Dream1, Dream2)     ' go to bit 0 or 1, BRANCH value, (Case_0, Case_1,
Case_2)
Dream1:                        ' begin 1st dream
NAP 0                          ' nap reduce 1ma to 25us
DEBUG CR, "Sweet ",#n,#result  ' show dream & random number
GOTO Again                     ' next random dream
Dream2:                        ' begin second dream
NAP 0                          ' nap 18ms, reduce 1ma to 25us
DEBUG CR, "Dream ",#n,#result  ' next random dream or end of dream, show rnd numb
GOTO Again                     ' repeat

' ---------------------- Fuzzy Clock --------------------------------
```

```
FuzzyClock:                    ' this computer will sleep equal to c
x=c*3                          ' sleep formula
DEBUG CR,CR,"(Sleeping)",CR    ' display sleeping mode
SLEEP x                        ' its computer number in seconds
DEBUG "(wake up)",CR           ' Display clock status

' ---------------------- Announce id & c to Everyone ----------------

SEROUT ipin,baud,("!",id,c)    ' wake up, announce this computer's ID and #,
DEBUG "(talk) ",#id,#c,CR      ' show id & c were sent, computer 10 is last to wake

' ---------------------- Listen for Neighbors id & c ----------------

' Written for the 3 computer TriCore Supercomputer, change for seed supercomputer
' if c = 1   x = 1 to 2  listen for two computers  x = 1 to (3-c)
' if c = 2   x = 1 to 1  listen for one computer
' if c = 3   goto end    do not listen, this is the last computer
' cannot count itself as listening to itself (set for TriCore supercomputer)

v=2                            ' init the write addr, start at 2
Listen:                        ' listen to other computers speaking their id's
IF c = 3 THEN Think            ' jump if computer 3, cannot listen as sleeping
x = 3 - c                      ' equation to determine looping
FOR h = 1 TO x                 ' do the loops
SERIN  ipin,baud,("!"),id,c    ' look at serial pin 1 for incoming id & computer #
DEBUG "(hear) ",#id,#c,CR      ' show received id & c
GOSUB Memorize                 ' remember what you hear
NEXT                           ' next iteration
GOTO Recall                    ' recall what is heard

' ---------------------- Memorize Neighbors id & c ------------------

Memorize:                      ' place heard computer's id and c in memory
DEBUG "(memorizing)",CR,CR     ' memorize the id and c values
WRITE v,id                     ' datalog to eeprom, v is eeprom addr = 2
v=v+1                          ' inc eeprom addr
WRITE v,c                      ' datalog to eeprom
v=v+1                          ' increment memory loc
RETURN                         ' listen again

' ---------------------- Recall Neighbors ---------------------------

Recall:                        ' Read/display neighbors id & c
DEBUG "(recall neighbors)",CR  ' monitor recall
v=1                            ' init mem loc
DEBUG "mem loc, id/com ",CR    ' display memory location, id, computer number
Repeat:                        ' recall memory subroutine
v = v+1                        ' begin at mem loc 2
READ v,x                       ' recall v and id
DEBUG #v,#x,CR                 ' display mem loc, read val
PAUSE 500                      ' slow to see screen vals
IF v>6 THEN Think              ' CHANGE THIS TO REFLECT WHICH SUPERCOMPUTER!
GOTO Repeat                    ' again

' ---------------------- Do Some Thinking/Work ----------------------

Think:                         ' think routine
DEBUG CR,"(Thinking)",CR       ' announce thinking
READ 1,c                       ' recall computer number from memory
DEBUG "I am computer ",#c,CR   ' announce my computer number
READ 0,id                      ' recall id from memory
DEBUG "My id is ",#id,CR       ' announce my id
READ 1,c                       ' recall this computer number
x=c*3                          ' formula for length of sleep
DEBUG "I slept  ",#x," sec",CR ' how long did I sleep?
h=3-c                          ' formula for the number of heard coms
```

```
DEBUG "I heard  ",#h," com",CR ' announce # of computers heard
h=2-h                          ' formula for the number of missed coms
DEBUG "I missed ",#h," com",CR ' announce computers not heard
DEBUG "I had    10 dreams ",CR ' announce computers not heard
GOTO Halt                      ' end

' ---------------------- Suspended Animation ------------------------

Halt:                          ' heart beat every 2.6 seconds
DEBUG CR,"(bye)"               ' monitor mode
END                            ' maintain state

' -------------------------------------------------------------------
```

Application 30: Putting Your Supercomputer to Sleep

Enter sleep mode for a specified number of seconds. • Seconds is a variable/constant (1–65535) that specifies the duration of sleep in seconds. The length of sleep can range from 2.3 seconds (see note below) to slightly over 18 hours. Power consumption is reduced to about 20 µA, assuming no loads are being driven. Note: The resolution of Sleep is 2.304 seconds. Sleep rounds the seconds up to the nearest multiple of 2.304. Sleep 1 causes 2.3 seconds of sleep, while Sleep 10 causes 11.52 seconds (5 x 2.304). Sleep lets the Stamp turn itself off, then turn back on after a specified number of seconds. The alarm clock that wakes the Stamp up is called the watchdog timer. The watchdog is an oscillator built into the BASIC interpreter. During sleep, the Stamp periodically wakes up and adjusts a counter to determine how long it has been asleep. If it isn't time to wake up, the Stamp goes back to sleep. To ensure accuracy of sleep intervals, the Stamp periodically compares the period of the watchdog timer to the more accurate resonator timebase. It calculates a correction factor that it uses during sleep. Longer sleep intervals are accurate to ±1 percent. If your Stamp application is driving loads during sleep, current will be interrupted for about 18 ms when the Stamp wakes up every 2.3 seconds. The reason is that the reset that awakens the Stamp causes all of the pins to switch to input mode for approximately 18 ms. When the BASIC interpreter regains control, it restores the I/O direction dictated by your program. If you plan to use End, Nap, or Sleep in your programs, make sure that your loads can tolerate these periodic power outages. The simplest solution is to connect resistors high or low (to +5V or ground) as appropriate to ensure a supply of current during reset.

```
' Sleep.bs1
SLEEP 3600 ' Sleep for about 1 hour.
goto xyz ' Continue with program after sleeping.
```

Single Processor Specifications by Parallax

The BASIC Stamp 1 module uses a PIC16C56a from Microchip Technology Inc.
Processor Speed: 4 MHz
Program Execution Speed: ~2,000 PBASIC instructions/sec.
RAM Size: 16 Bytes (2 I/0, 14 Variable)
EEPROM (Program) Size: 256 Bytes; ~80 PBASIC instructions
Current Draw @ 5 VDC: 1mA Run, 25 µA Sleep
Number of I/O Pins: 8
Source/Sink Current per I/O: 20 mA / 25 mA
Source/Sink Current per unit: 40 mA / 50 mA
PBASIC Commands: 32
PCB Size: 3 ¼ " x 2 ½ " (with 1/8" corner mounting holes)
1/8" corner mounting holes surrounded by grounded islands
Project Area: 1 3/8 x 2" solderless breadboard or through-hole mounting pads
Rev B boards: the power LED remains illuminated when the BASIC Stamp is in sleep mode.
The BS1 Project Board consists of a surface-mounted BASIC Stamp 1 on a 3 1/4" x 2 1/2" (8.3 x 6.4 cm) printed circuit board. The board includes a 9V battery clip, a mechanically interlocked 2.1 mm power jack, DB-9 connector for programming, and LM2936 regulator providing 40 mA for your projects.

Actual Conditions Single Processor
Current Draw @ 5 VDC: 8 mA Running
Actual Conditions Ten Processors
Current Draw @ 5 VDC: 80 mA Run
Ten Processor Specifications
Ten Cores
Core Speed 40 Mhz
Program Execution Speed: ~20,000 PBASIC instructions/sec.

RAM Size: 160 Bytes (20 I/0, 140 Variable)
EEPROM (Program) Size: 2,560 Bytes; ~800 PBASIC instructions
Current Draw @ 5 VDC: 80mA Run
Number of I/O Pins: 80
Techniques for 1600 inputs, or 160 I/Os
Source/Sink Current per I/O: 20 mA / 25 mA
Source/Sink Current per unit: 40 mA / 50 mA
Source/Sink Current per supercomputer I/O: 400 mA / 500 mA

kiloFLOPS	10^3
megaFLOPS	10^6
gigaFLOPS	10^9
teraFLOPS	10^{12}
petaFLOPS	10^{15}
exaFLOPS	10^{18}
zettaFLOPS	10^{21}
yottaFLOPS	10^{24}

Comparative Speed
Don't expect too much speed compared to the worlds fastest supercomputers or even desktop computers! This is a hobby supercomputer and is super in terms of comparison to one Stamp. If you want to convert the actual speed to FLOPS, the ***speed chart*** will illustrate the example (at left).

The SEED runs at about 20,000 IPS. FLOPS is floating point operations per second. A FLOP takes a longer time compared to an IPS, or instruction per second. Approximating IPS to FLOPS, the SEED is about 20 kiloFLOPs or one fifth of a megaFLOP. By comparison, the Cray 1 supercomputer was in the MIPs range, or million instructions per second. So one could roughly say the SEED is only approaching the speed of the Cray supercomputer.

Upgrades and Improvements
Before releasing the project, several improvements were made. This is a log of those changes.

- Added slit soda straws as wire holders on brass spacers. Heavier metal clips were removed.
- Spacers were increased to 6cm length. This makes it easy to reach in and rewire the board.
- The control panel was relocated from the top to the bottom, for easy use when setting on a desk top.
- Spacers cage added at the Skyscraper top protects the wireless transceiver and antennae.
- Plastic mounts were added to reinforce the cage.
- The LCD was removed after debugging and testing to minimize portable power requirements.
- A Crown was installed to frame up the front two spacers and add an English/Chinese logo
- The cage lid was trimmed in the front to accommodate the Crown
- Banner moved to the lid to make more transceiver wiring space

Cocooning for Transport
It is advised to stretch wrap the entire *Basic SEED Supercomputer* when moving it outside from one location to another. This prevents wires from being dislodged and protects the internal space of each computer. It dust seals the inside to prevent contamination from getting into the components and breadboards.

Cocooning Procedure
Use a roll of kitchen Seran-Wrap or similar plastic stretch wrap. Start at the top of the tower and wrap around in a circular direction, slowly moving from top to bottom. See photos. To minimize punctures during transport, layer the wrap. The entire supercomputer will now fit diagonally into a standard medium size luggage with space for packing on the sides, top and bottom.

Coming Out of the Cocoon
When removing the wrap, do not cut it with a knife or scissors or wires may become damaged. It's a simple matter to carefully unwrap it. Check all components and wiring, first by visual inspection, and next by running tests.



The SEED is cocooned for transporting by stretch wrapping with common kitchen household plastic wrap, similar to Saran Wrap. This is the same method used to shrink wrap heavy pallets for warehouse distribution.

Resources and Links

Parallax - Basic Stamp 1 Project Board Manual
Parallax - Basic Stamp Manual
Parallax - Basic Stamp 1 Applications
Parallax - 433Mhz Transmitter Manual
Parallax - 433Mhz Receiver Manual
Parallax - The Elements of PBASIC Style
Parallax - Stampworks
Parallax - Collection of Nuts & Volts Basic Stamp 1 Articles
Parallax – What's a Microcontroller?
Parallax – Forum
Scott Edwards – Programming and Customizing the Basic Stamp Computer
Al Williams – Microcontroller Projects with Basic Stamps
Data Sheets for Sensors and Project Components
Vrossi - Advanced Boe-Bot with 2 Stamps
NGL – The BST Basic Stamp Tower
Mike2545 – The Tower Platform
Humanoido – Stamp SEED Supercomputer, Parallax Forum
Humanoido – Minuscule Stamp Supercomputer – Manual for the Stamp Seed Supercomputer
Humanoido - BSS in Penguin Tech Magazine
Humanoido - BSS Supercomputer - Parallax Forum
Humanoido - BSS Supercomputer Thread, Penguin Robot Web Site
Humanoido - StampOne News! A Three Dimensional Stamp Computer
Humanoido – 3DSC A Three Dimensional Stamp Computer – Parallax Forum
Humanoido - Penguin with 12 Brains, Penguin Robot Web Site
Humanoido - Penguin with 12 Brains, Parallax Forum
Humanoido - Penguin Robot Web Site
Wikipedia – EMI RFI
Wiki-Answers.com – Length of a City Block in New York City

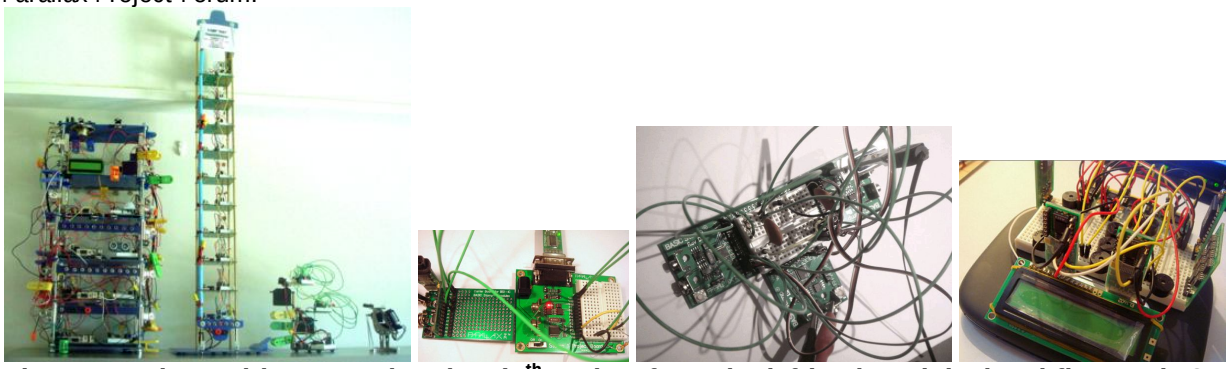
Parallax Basic Stamp 1 Applications Source

Note #01 LCD User-Interface Terminal.
Note #02 Interfacing an A/D Converter.
Note #03 Hardware Solution for Keypads.
Note #04 Controlling and Testing Servos.
Note #05 Practical Pulse Measurements.
Note #06 A Serial Stepper Controller.
Note #07 Sensing Temperature with a Thermistor.

Note #08 Sending Messages in Morse Code.
Note #09 Constructing a Dice Game.
Note #10 Sensing Humidity and Temperature.
Note #11 Wireless Infrared Communication.
Note #12 Cheap Sonar Rangefinding with the Stamp.
Note #13 Using Serial EEPROMS with the Stamp.
Note #14 Networking Multiple Stamps.
Note #15 Using PWM for Analog Output.
Note #16 Keeping Stamp Source Private.
Note #17 The Solar-Powered Stamp.
Note #18 One pin, Many switches. How to sense 8 switches using only 1 pin.
Note #19 Using Button Effectively. Delay, auto, repeat.
Note #20 An Accurate Timebase. Uses 32,768hz Xtal.
Note #21 Fun with Trains. Controls the movement of 3 model trains.

A Growing family of Supercomputers

Work is progressing on yet another Basic Stamp supercomputer, with new features. A family of generations is being developed. For the latest updates and information about these Stamp supercomputer projects, see thread postings in the Parallax Project Forum.



For size comparison with a Penguin robot (4th project from the left)– the original and first Basic Stamp Supercomputer, followed by the Stamp SEED Supercomputer, and the Three Dimensional Stamp Computer. Right 3 Photos- Minuscule Stamp Supercomputer, Stamp Tricore Supercomputer, and the Tiny Stamp Supercomputer.

The Basic Stamp Supercomputer family				
Acronym Full Name	**No. of Processors**		**Type**	**Features**
	Original	**Expand**		
BSS Basic Stamp Supercomputer	10	22	BS2 Mix	1st Stamp Supercomputer
SSS Stamp Seed Supercomputer	10		BS1	1st AI Supercomputer
MSS Minuscule Stamp Supercomputer	2		BS1	Smallest Core Supercomputer
STS Stamp Tricore Supercomputer	3		BS1	Supercomputer for Testing
TSS Tiny Stamp Supercomputer	7		BS1USB	1st USB Supercomputer

Stamp Baby Supercomputer Announcement 06.10.09

Ten months in the making, and nicknamed the "SEED," the *Stamp SEED Supercomputer* is a significantly different evolved project from the original BSS Supercomputer. Intended for use with the new created Hive Algorithm, a spinoff of Miniscule AI for Artificial Intelligence, it has no Master, and uses a completely new tower design named the *Skyscraper*.

It communicates bidirectional with the BSS, other supercomputers and Stamps, and a home base station, *Bstat*, using a Parallax 433 Mhz Radio Frequency Transmitter and Receiver pair. This makes the SEED so powerful, it can direct commands at an entire group of computers and supercomputers located in an office building, given the wireless range over a full city block.

Programming is significantly different too, as the Stamps use another version of PBASIC. Signal LEDs and the LCD are not needed. The piezo speaker is retained for signaling, debugging and communicating with the operator in the new Piezo Language PLAN. The Assembly manual includes, *Project Description, History & Details*, *Photos & Schematics*, *Detailed Assembly Guide*, *and Complete Software including Tests and Demos.*

A Very Short Guide to PBASIC 1 Instructions - Programming the BS1 Supercomputer

. BRANCH Branch to address specified by offset
. BUTTON Monitor/manage button input; branch if button in target state
. DEBUG Send variables and/or messages to PC for viewing
. EEPROM Store user data in available EEPROM space
. END Terminate program and enter low-power mode until reset
. FOR...NEXT Create numerically controlled loop
. GOSUB Unconditional branch to a subroutine
. GOTO Unconditional branch to program address
. HIGH Make pin an output high
. IF...THEN Compare and conditionally branch to program address
. INPUT Make pin an input
. LET Optional designator for assignments
. LOOKDOWN Search target in table; set output variable to target location
. LOOKUP Set output variable to table data specified by offset
. LOW Make pin output low
. NAP Enter low-power mode for short period
. OUTPUT Make pin an output
. PAUSE Suspend program for 1 to 65,535 milliseconds
. POT Read a variable resistance and scale result
. PULSIN Measure width of an input pulse
. PULSOUT Output timed pulse by inverting pin for some time
. PWM Output analog level (requires external RC network for filtering)
. RANDOM Generate a pseudo-random number
. READ Read byte from EEPROM location
. RETURN Return from a subroutine
. REVERSE Reverse pin state; make input if output, output if was input
. SERIN Receive serial data, 300 to 2400 baud, N81 format
. SEROUT Transmit serial data, 300 to 2400 baud, N81 format
. SLEEP Enter low-power mode for 1 to 65,535 seconds
. SOUND Generate tone or white noise
. TOGGLE Make pin an output and toggle current state
. WRITE Write byte to EEPROM location

By Request - Interview with Dr. Humanoido
Project Developer - the Series of Basic Stamp Supercomputers

Q – How did you get the idea to build a Basic Stamp supercomputer?
A – I always wanted to build a more powerful computer using Basic Stamps. This is spun off from early robot years when my humanoid robots needed a more powerful brain. All the credit goes to Parallax and Ken Gracey. I took his creation, a Parallax Toddler robot parts kit, and put together a humanoid. For more function, I gave it multiple Basic Stamps, and believe it or not, that was the forerunner of today's Stamp supercomputers!

Q – A self enumerating determinist supercomputer is a big jump from the previous stamp supercomputer. What prompted you to initiate this breakthrough?
A – I was reading Chip Gracey's main supercomputer forum thread and learned this was the way to go in new design, due to many new advantages. So basically I give all credit to Chip Gracey.

Q – Is putting this system into operation an easy task?
A – Sometimes the most simple things in life turn out to be the most challenging and the converse is true also, the most complicated things turn out to be the most simple.

Q – Exactly what are the advantages of this new supercomputer? What do you call it? The Self Enumerating Deterministic Basic Stamp Supercomputer?
A – Well, that's a big name and originally I simply called the software the *Hive Algorithm*, but simply put, it means we no longer need to load many different computer programs into each computer board in the supercomputer. It only takes one program now, the same program loaded into every computer no matter how many computers are in the supercomputer, 2, 10, 20... There's a short paper I wrote that explains how it works.

Q – Tell us something interesting about that.
A – Well, I decided to do all this with the Basic Stamp 1. The EEPROM allows a maximum of maybe 80 program statements. I was worried about overhead and things fitting. By the time the program was actually written and reworked several times, it became small code with space left over. That's space other people can use for testing things and doing some real supercomputing.

Q – What can you say about the software?
A –It's designed for a supercomputer with ten BS1s. The code sections in the test version are many. You'll find EEPROM Read, Write, Datalogging, Sound Generation, Serial Communication on the Net, Table Storage, RC Circuit Reads, and Scaling. It's probably a lot to fit in but see for yourself - there's actually space left over.

Q – We heard you had an unusual experience with the latest supercomputer, when you first switched it on. What happened?
A – I plugged in the banana jack power cable and the supercomputer lit up like a Christmas Tree (it was a good thing) with all ten computers and bright red LEDs. But something more was going on before loading any programs. With a cycle rate of about 2 ½ seconds, all the lights would twinkle. I must say, it was beautiful and totally unexpected, but a perfectly normal response from the Basic Stamp One Boards. After loading a program into each computer, the effect quit. So I studied what happened, and wrote a one line program to repeat the effect. There's a write-up about it in the SEED manual.

Q – What are your plans for the future of Basic Stamp Supercomputers?
A – I plan to work with the Basic SEED developing new apps, and keep going on projects with the first BSS Basic Stamp Supercomputer. The posting at the forum seems to have generated some interest and I like to post new information there. At the same time now, another Basic Stamp supercomputer is under development. I believe it will be the 5th and final one in the series.

Q – Are you saying there will be three Basic Stamp Supercomputers?
A – Yes, and then more. That's why I started writing about clustering together supercomputers by wireless radio frequency.

Q – What is the difference from one Stamp supercomputer to the next?
A – They are each totally different with unique features. Mainly, no two are the same and each introduces new features and concepts. The first one can use any Basic Stamp 2 in its collective. The second one uses only Basic Stamp ones. The third is super unique and you'll have to wait and see.

Q – If you were to gaze into the crystal ball, what do you see after your final Basic Stamp Super-computer?
A – I recently got hooked on the *Genius Chip* – a Propeller chip made by Parallax company. It's a revolutionary new 160 MIPS processor with eight cores that has unlimited potential.

Q – Thank you Dr. Humanoido for you time in participating with this interview.
A – Thanks sincerely. It was all my pleasure. I especially want to thank everyone at Parallax, Ken Gracey for introducing me to the Basic Stamp and Propeller chip, Chip Gracey for making my life more complicated, and everyone who has posted helpful and humorous information at the Parallax Forum.

Basic Stamp 1 Instruction List

```
Stamp Instruction Set: BASIC Stamp - Instruction List
When a program is loaded, the eeprom is erased. When a program is first run, all
variables are cleared to 0.  The variables total to 16 bytes of RAM and are arranged as
follows:

Data            Words   Bytes   Bits
-------------------------------------------------------------------------
0000 0000       PORT    PINS    PIN0 - PIN7  (PINS.0, PORT.0 - PINS.7, PORT.7)
0000 0000               DIRS    DIR0 - DIR7  (DIRS.0, PORT.8 - DIRS.7, PORT.15)

0000 0000       W0      B0      BIT0 - BIT7  (B0.0, W0.0 - B0.7, W0.7)
0000 0000               B1      BIT8 - BIT15 (B1.0, W0.8 - B1.7, W0.15)

0000 0000       W1      B2
0000 0000               B3
```

```
0000 0000       W2      B4
0000 0000               B5

0000 0000       W3      B6
0000 0000               B7

0000 0000       W4      B8
0000 0000               B9

0000 0000       W5      B10
0000 0000               B11

0000 0000       W6      B12
0000 0000               B13

PORT is the I/O port word:

        PINS and PIN0-PIN7 are the port (RB) pins (buffered to overcome the
        read-modify-write problem).  When these variables are read, RB is read
        directly.  When these variables are written to, the corresponding RAM
        is written to, which is then transferred to RB before each instruction.

        DIRS and DIR0-DIR7 are the port (RB) direction bits: 0=input, 1=output
        (buffered to overcome the write-only nature of the TRIS registers).
        This byte of data is transfered to !RB before each instruction.

W0-W6, B0-B13, and BIT0-BIT15 are for general-purpose use.

W6 (B12,B13) is used as a four-level stack if any GOSUB/RETURNs are executed.

-------------------------------------------------------------------------------
ANALOG                  8-bit resolution I/O via resistor-capacitor circuits
-------------------------------------------------------------------------------

PWM  pin,duty,cycles

        Output pwm then return pin to input.  This is used to output analog
        voltages (0-5V) via a pin connected to a resistor connected to a cap
        connected to ground; the resistor-cap junction being the analog output.
        PWM should be executed periodically to update/refresh the analog
        voltage.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Duty is a variable/constant (0-255) which specifies analog level.
        - Cycles is a variable/constant (0-255) which specifies number of
          cycles.  Each cycle takes about 5ms.

POT  pin,scale,variable

        Digitize a 5-50K ohm pot with a .1uf cap and scale result.  This is a
        method of analog input via a pin.  The pin must connect to the center-
        tap of the pot, whose side tap connects to the cap which connects to
        ground.  Use the 'Alt-P' command to determine the scale, while the
        stamp is connected and the pot circuit is in place.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Scale is a constant (0-255) which is used to scale the initial result
          by n/256 (result is then limited to 255).
        - Variable receives the result (0-255).

-------------------------------------------------------------------------------
SOUND                   Direct audio output using a cap and high-z speaker
-------------------------------------------------------------------------------

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

```
        Play notes with durations.  The pin must connect to the plus side of a
        => 10uf cap whose minus side connects to a => 40 ohm (or peizo) speaker
        whose other terminal connects to ground.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Note(s) are variables/constants (0-255) which specify type and
          frequency.  Note 0 is silent for the duration.  Notes 1-127 are
          ascending tones.  Notes 128-255 are ascending white noises.
        - Duration(s) are variables/constants (0-255) which specify duration.

------------------------------------------------------------------------------
SERIAL                  Input and output, plus networking via open-drain/source
------------------------------------------------------------------------------

SERIN  pin,baudmode,(qualifier,qualifier...)
SERIN  pin,baudmode,(qualifier,qualifier...),{#}variable,{#}variable...
SERIN  pin,baudmode,{#}variable,{#}variable...

        Serial input with optional qualifiers (8 data, no parity, 1 stop).

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Baudmode is a variable/constant (0-7) which specifies the mode:
                T2400           true input
                T1200           true input
                T600            true input
                T300            true input
                N2400           inverted input
                N1200           inverted input
                N600            inverted input
                N300            inverted input
        - Qualifiers are optional variables/constants (0-255) which must be
          received in exact order before subsequent bytes can be received and
          stored in variables.
        - Variable(s) receive the result(s) (0-255).  Optional #'s are for
          inputting ascii decimal numbers into variables, rather than raw
          characters.

SEROUT  pin,baudmode,({#}data,{#}data...)

        Serial output (8 data, no parity, 1 stop).

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Baudmode is a variable/constant (0-15) which specifies the mode:
                T2400           true output             always driven
                T1200           true output             always driven
                T600            true output             always drn
                T300            true output             always drn
                N2400           inverted output         always driven
                N1200           inverted output         always driven
                N600            inverted output         always driven
                N300            inverted output         always driven
                OT2400          true output             open-drain
                OT1200          true output             open-drain
                OT600           true output             open-drain
                OT300           true output             open-drain
                ON2400          inverted output         open-source
                ON1200          inverted output         open-source
                ON600           inverted output         open-source
                ON300           inverted output         open-source
        - Data are variables/constants (0-255) which provide the data to be
          output.  Optional #'s are for outputting ascii decimal numbers,
          rather than raw characters.

------------------------------------------------------------------------------
DIGITAL OUTPUT
------------------------------------------------------------------------------
```

```
OUTPUT  pin

        Make pin output.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

LOW  pin

        Make pin output low.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

HIGH  pin

        Make pin output high.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

TOGGLE  pin

        Make pin output and toggle state.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

REVERSE  pin

        Reverse the direction of a pin.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

PULSOUT  pin,time

        Output a timed pulse by inverting a pin for some time.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Time is a variable/constant which specifies the period (0-65535) in
          10us units.

------------------------------------------------------------------------------
DIGITAL INPUT
------------------------------------------------------------------------------

INPUT  pin

        Make pin an input

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.

PULSIN  pin,state,variable

        Measure an input pulse.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - State is a variable/constant (0 or 1) which specifies which edge must
          occur before beginning the measurement in 10us units.
        - Variable receives the result (1-65536).  If timeout occurs (.65536s)
          the result will be 0.

BUTTON  pin,downstate,delay,rate,bytevariable,targetstate,address

        Debounce button, auto-repeat, and branch if button is in target state.

        - Pin is a variable/constant (0-7) which specifies the i/o pin to use.
        - Downstate is a variable/constant (0 or 1) which specifies what
          logical state is read when the button is pressed.
```

```
        - Delay is a variable/constant (0-255) which specifies down-time before
          auto-repeat in BUTTON cycles.
        - Rate is a variable/constant (0-255) which specifies the auto-repeat
          rate in BUTTON cycles.
        - Bytevariable is the workspace.  It must be cleared to 0 before being
          used by BUTTON for the first time.
        - Targetstate is a variable/constant (0 or 1) which specifies what
          state (0=not pressed, 1=pressed) the button should be in for a
          branch to occur.
        - Address is a label which specifies where to go if the button is in
          the target state.

------------------------------------------------------------------------------
EEPROM                  Non-volatile data storage in unused program memory
------------------------------------------------------------------------------

READ  location,variable

        Read eeprom location contents into variable.  location 255 holds
        highest location minus one that is available to user for data storage.

        - Location is a variable/constant specifying a byte-wise address
          (0-255).  Data storage begins at 0 and builds upward, while program
          space builds downward from 255.
        - Variable receives the data byte read.

WRITE  location,data

        Write data into eeprom.

        - Location is a variable/constant specifying a byte-wise address
          (0-255).  Data storage begins at 0 and builds upward, while program
          space builds downward from 255.
        - Data is a variable/constant which provides the data byte to be
          written.

------------------------------------------------------------------------------
TIME                    Accurate-enough timer functions
------------------------------------------------------------------------------

PAUSE  milliseconds

        Pause for some time.  The duration of the pause is as accurate as the
        resonator time-base, however extra time (perhaps 1ms) will be spent
        gathering and executing the adjacent instructions.  In other words,
        the higher the milliseconds, the relatively less-significant is the
        extra time.  Therefore, accurate-enough timer functions can be
        implemented by PAUSE'ing, say, 100 (ms), then incrementing a variable
        which counts 1/10 seconds, checking for the end-condition, and if not
        true, looping back to the PAUSE.  In the case of 100 (ms), the error
        might be +1%.  This could be remedied by reducing 100 to 99 (based on
        a trial test), or increasing 100 to, say, 250, and then counting 1/4
        seconds instead.

        - Milliseconds is a variable/constant (0-65535) which specifies how
          many milliseconds to pause.

------------------------------------------------------------------------------
NUMERICS                Math, translation, pseudo-random generation
------------------------------------------------------------------------------

{LET}  variable = {-}value ?? value ...

??:     +       ;add
        -       ;subtract
        *       ;multiply (returns low word of result)
```

```
        **      ;multiply (returns high word result)
        /       ;divide (returns quotient)
        //      ;divide (returns remainder)
        MAX     ;make less than or equal to
        MIN     ;make greater than or equal to
        &       ;and
        |       ;or
        ^       ;xor
        &/      ;and not
        |/      ;or not
        ^/      ;xor not

        Perform variable manipulation (wordsize-to-wordsize).  Math is
        performed strictly from left to right.

        - Variable will be operated on.
        - Value(s) are variables/constants which operate on Variable.

LOOKUP  offset,(data0,data1...dataN),variable

        Lookup data specified by offset and store in variable (if in range).

        - Variable receives the result (if any).
        - Offset is a variable/contant which specifies which data# (0-N) to
          place in Variable.
        - Data are variables/constants.

LOOKDOWN  target,(value0,value1...valueN),variable

        Get target's match number (0-N) into variable (if match found).

        - Variable receives the result (if any).
        - Target is a variable/constant which will be compared to Values.
        - Values are variables/constants.

RANDOM  wordvariable

        Generate next pseudo-random number in a word variable.

        - Wordvariable is both the workspace and the result.

------------------------------------------------------------------------------
LOOPING
------------------------------------------------------------------------------

FOR  variable = start TO end {STEP {-}increment}
NEXT {variable}

        Establish FOR-NEXT loop.

        - Variable will be used as a counter
        - Start is the initial value of Variable
        - End is the terminal value of Variable
        - Increment is an optional value which overrides the default counter
          delta of +1.  If Increment is preceeded by a '-', it will be assumed
          that End is greater than Start, and therefore, Increment will be
          subtracted each time through the loop rather than added

------------------------------------------------------------------------------
BRANCHING
------------------------------------------------------------------------------

IF variable ?? value {AND/OR variable ?? value ...} THEN address

        ?? can be =, gt=,lt=,gt,lt (symbols changes slightly 'html' conflict)
                                   (Don McKenzie.)
```

```
        Compare and conditionally branch.

        - Variable(s) will be compared to Value(s).
        - Value is a variable/constant.
        - Address is a label which specifies where to go if condition is true.


BRANCH  offset,(address0,address1...addressN)

        Branch to address specified by offset (if in range).

        - Offset is a variable/constant which specifies which Address# to use
          (0-N).
        - Addresses are labels which specify where to go.

GOTO  address

        Go to address.

        - Address is a label which specifies where to go.

GOSUB  address

        Go to subroutine at address.  Up to 16 GOSUBS are allowed and they
        can be nested up to 4 deep.

        - Address is a label which specifies where to go.

RETURN

        Return from subroutine.

------------------------------------------------------------------------------
POWER CONTROL           Achieves maximum battery life
------------------------------------------------------------------------------

NAP  period

        Nap for a short period.  Power consumption is reduced to an extent
        dependent upon the length of the period.

        - Period is a variable/constant which determines the duration of the
          reduced-power nap.  The duration will be 2^period * ~18ms.  Period
          can range from 0 to 7.

SLEEP  seconds

        Sleep for some seconds (resolution is ~2.3s, accuracy is ~99.9%).
        Power consumption is reduced to 1/100th normal (assuming no loads are
        being driven).

        - Seconds is a variable/constant which specifies the duration of sleep
          in seconds (0-65535).

END

        Sleep terminally until the power cycles (program re-runs) or the pc
        connects.  Power is reduced to an absolute minimum (assuming no loads
        are being driven).

------------------------------------------------------------------------------
DEBUGGING               Used in program-development
------------------------------------------------------------------------------

DEBUG  cls,"Text",cr,var,$var,%var,#var,#$var,#%var
```

```
        Display information in the debug window upon encounter during
        execution.

        - Text to be printed can be expressed in quotes; in addition, 'cls'
          (clear screen) and 'cr' (carriage return) can be used.

        - Variables can be printed with their current values by simply naming
          the variable.  Decimal is the default, but '$' can be used for hex
          and '%' can be used for binary.  If a '#' preceeds a variable (with
          optional '$' or '%' in front) the variable name will not be printed,
          instead only the value will be printed.

-------------------------------------------------------------------------------
EEPROM                  Preloads EEPROM data locations
-------------------------------------------------------------------------------

EEPROM  {location},(data,data...)

        Preload EEPROM data locations (those not used by the program).  This
        is not an instruction, but rather a means to preload otherwise cleared
        eeprom locations.

        - Location is an optional constant (0-255) which specifies where to
          begin storing the data in the eeprom.  If no location is specified,
          storage continues from where it last left off.  If no location was
          initially specified, storage begins at 0.  Data storage builds upward
          from 0 while program storage builds downward from 255.
        - Data are constants (0-255) which will be stored in the eeprom.

-------------------------------------------------------------------------------
SYMBOL                  Establishes alias names for variables and constants
-------------------------------------------------------------------------------

SYMBOL  symbolname = value

        Assign a value to a new symbolname.

        - Symbolname is a text string which must begin with an alpha-character
          or '_'.  After the first character, it can also contains number
          characters ('0'-'9').
        - Value is a variable or constant which is being given an alternate
          symbolname.
```

PBASIC Programming Guidelines for the Stamp 1

Like most versions of the BASIC programming language, PBASIC is very forgiving and the compiler enforces no particular formatting style. As long as the source code is syntactically correct, it will compile and download to the BASIC Stamp without trouble. Why, then, would one suggest a specific style for PBASIC? Consider this: Over three million BASIC Stamp modules have been sold and there are several thousand members that participate in the Parallax online forums. This makes it highly likely that you'll be sharing your PBASIC code with someone, if not co-developing a BASIC Stamp project. Writing code in an organized, predictable manner will save you -- and your potential colleagues -- a lot of time; in analysis, in troubleshooting and especially when you return to a project after a long break. The style guidelines presented here are just that: guidelines. They have been developed from style guidelines used by professional programmers using other high-level languages such as Visual Basic®, C/C++, and Java™. Use these guidelines as-is, or modify them to suit your individual needs. The key is selecting a style that works well for you or your organization, and then sticking to it. Do It Right The First Time: Many programmers, especially new ones, fall into the "*I'll knock it out now and fix it later.*" trap. Invariably, the "*fix it later*" part never happens and sloppy code makes its way into production projects. If you don't have time to do it right, when will you find time to do it again? Start clean and you'll be less likely to introduce errors in your code. And if errors do pop up, clean and organized formatting will make them easier to find and fix. Be Organized and Consistent: Using a blank program template will help you organize your programs and establish a consistent presentation. Use Meaningful Names: Be verbose when naming constants, variables and program labels. The compiler will allow names up to 32 characters long. Using meaningful names will reduce the number of comments and make your programs easier to read, debug and maintain.

Naming Constants: Begin constant names with an uppercase letter and use mixed case, using uppercase letters at the beginning of new words within the name.

```
SYMBOL          AlarmCode   = 25
```

Naming Variables: Begin variable names with a lowercase letter and use mixed case, using uppercase letters at the beginning of new words within the name. Avoid using W0 (B0 and B1) so that bit variables (Bit0..Bit15) are available for use in your programs. Bit variables 0..15 overlay W0, so the use of W0 may cause undesired effects.

```
SYMBOL          waterLevel  = W1
```

Variable Type Declarations: Variable type is declared by aliasing the SYMBOL name to an internal variable of a specific size.

```
SYMBOL          status      = Bit0
SYMBOL          ovenTmp     = B2
SYMBOL          rndVal      = W2
```

Conserve BASIC Stamp user RAM by declaring the variable type required to hold the expected values of the variable.

```
SYMBOL          bitVal      = BIT0              ' 0 - 1
SYMBOL          byteVal     = B2                ' 0 - 255
SYMBOL          wordVal     = W2                ' 0 - 65535
```

Program Labels: Begin program labels with an uppercase letter, used mixed case, separate words within the label with an underscore character and begin new words with a number or uppercase letter. Labels should be preceded by at least one blank line, begin in column 1 and must be terminated with a colon (except after GOTO and THEN [in classic PBASIC] where they appear at the end of the line and without a colon).

```
Print_String:
  READ eeAddr, char
  IF char = 0 THEN Print_Done
  DEBUG char
  eeAddr = eeAddr + 1
  GOTO Print_String

Print_Done:
  RETURN
```

PBASIC Keywords: All PBASIC language keywords, including SYMBOL, CON, VAR, PIN and serial/debugging format modifiers (DEC, HEX, BIN) and constants (CR, LF) should be uppercase.

```
Main:
  DEBUG "BASIC Stamp", CR
  END
```

Indent Nested Code: Nesting blocks of code improves readability and helps reduce the introduction of errors. Indenting each level with two spaces is recommended to make the code readable without taking up too much space. Note: The dots are used to illustrate the level of nesting and are not a part of the code.

```
Main:
..FOR testLoop = 1 TO 10
....IF checkLevel >= Threshold THEN LED_Okay
....lowLevel = lowLevel + 1
....GOTO Loop_Delay

LED_Okay:
....LEDokay = IsOn

Loop_Delay:
....PAUSE 100
..NEXT
```

```
..IF testMode = Yes THEN Main
```

Control Codes			Printing Characters					
Name/Function	*Char	Code	Char	Code	Char	Code	Char	Code
null	NUL	0	<space>	32	@	64	`	96
start of heading	SOH	1	!	33	A	65	a	97
start of text	STX	2	"	34	B	66	b	98
end of text	ETX	3	#	35	C	67	c	99
end of xmit	EOT	4	$	36	D	68	d	100
enquiry	ENQ	5	%	37	E	69	e	101
acknowledge	ACK	6	&	38	F	70	f	102
bell	BEL	7	'	39	G	71	g	103
backspace	BS	8	(	40	H	72	h	104
horizontal tab	HT	9	)	41	I	73	i	105
line feed	LF	10	*	42	J	74	j	106
vertical tab	VT	11	+	43	K	75	k	107
form feed	FF	12	,	44	L	76	l	108
carriage return	CR	13	-	45	M	77	m	109
shift out	SO	14	.	46	N	78	n	110
shift in	SI	15	/	47	O	79	o	111
data line escape	DLE	16	0	48	P	80	p	112
device control 1	DC1	17	1	49	Q	81	q	113
device control 2	DC2	18	2	50	R	82	r	114
device control 3	DC3	19	3	51	S	83	s	115
device control 4	DC4	20	4	52	T	84	t	116
negative acknowledge	NAK	21	5	53	U	85	u	117
synchronous idle	SYN	22	6	54	V	86	v	118
end of xmit block	ETB	23	7	55	W	87	w	119
cancel	CAN	24	8	56	X	88	x	120
end of medium	EM	25	9	57	Y	89	y	121
substitute	SUB	26	:	58	Z	90	z	123
escape	ESC	27	;	59	[	91	{	124
file separator	FS	28	<	60	\	92	|	125
group separator	GS	29	=	61	]	93	}	126
record separator	RS	30	>	62	^	94	~	127
unit separator	US	31	?	63	_	95	<delete>	128

* Note that the control codes have no standardized screen symbols. The characters listed for them are just names used in referring to these codes. For example, to move the cursor to the beginning of the next line of a printer or terminal often requires sending linefeed and carriage return codes. This common pair is referred to as "LF/CR."

Review the Basic Stamp Supercomputer at the listed sources.

Sources

Email: penguin (dot) robot (at) yahoo (dot) com

Stamp SEED Supercomputer http://forums.parallax.com/forums/default.aspx?f=21&m=361377

Basic Stamp Supercomputer http://forums.parallax.com/forums/?f=21&m=308220&g=308220#m308220

Minuscule Stamp Supercomputer http://forums.parallax.com/forums/default.aspx?f=21&m=365789

TriCore Stamp Supercomputer http://forums.parallax.com/forums/default.aspx?f=21&m=366864

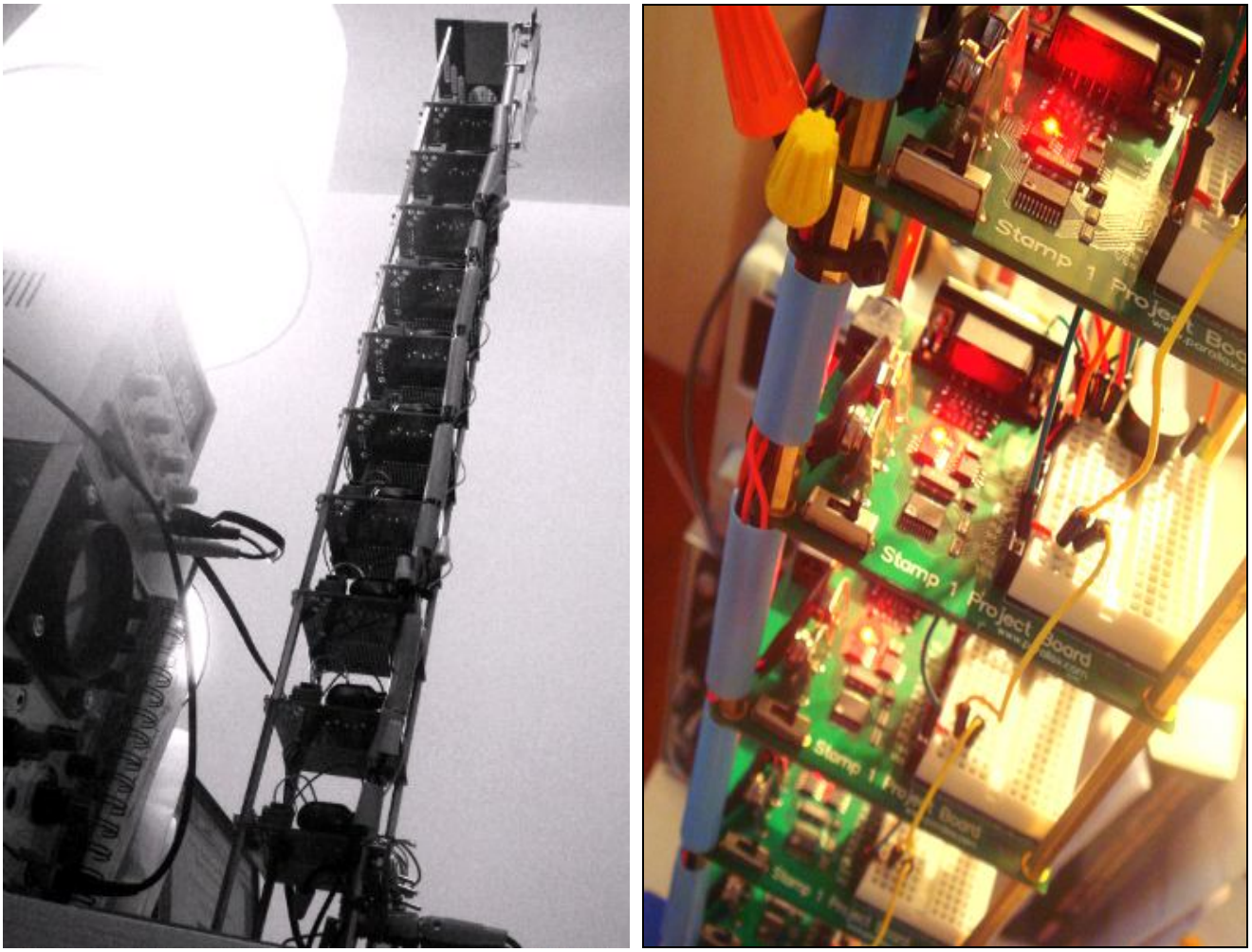
Three Dimensional Computer http://forums.parallax.com/forums/default.aspx?f=21&m=343484

Penguin with 12 Brains http://forums.parallax.com/forums/default.aspx?f=10&m=308600

Penguin Tech http://forums.parallax.com/forums/default.aspx?f=10&m=313398

StampOne News! http://forums.parallax.com/forums/default.aspx?f=5&m=342718

Penguin Robot Society http://www.robotinfo.net/penguin/



The Stamp Seed Supercomputer SSS in action.

Made with Parallax Stamps