The Nuts and Volts of BASIC Stamps

Parallax, Inc.
www.**parallaxinc**.com

Nuts and Volts
www.**nutsvolts**.com

Column #12, February 1996 by Scott Edwards:

Model Rocket Project Aims High With BS1-IC Instrumentation
Measuring Rocket Acceleration And Velocity And Truth and Consequences with IF/THEN

MODEL rockets and Stamp microcontrollers appeal to educators for many of the same reasons. Both offer:

- Hands-on experience with high techology.
- A starting point for discussions of science and math fundamentals.
- An inexpensive, but thoroughly educational class project.
- Impressive, tangible accomplishments that *wow* school administrators and parents.
- Enough fun and excitement to turn slack-jawed Nintendo burnouts into enthusiastic students!

You don't have to take my word for it. This month, I'm turning the first part of the column over to Dave Bodnar, Technology Coordinator for the Mount Lebanon School District of Pittsburgh, Pennsylvania. Last summer, Dave combined Stamp electronics with model-rocket pyrotechnics to create a couple of high-flying Stamp applications. Here's a description in Dave's own words:

Enclosed you will find the prototypes of the rocket sensors that I put together this summer. One senses the rocket's speed by measuring the speed of a propeller on the nose cone. The other measures acceleration.

The speed device uses an infrared phototransistor/emitter pair to detect the spinning of the propeller. A split wheel chops the light that normally goes between the emitter and detector. The two LEDs and buzzer are used to let the launch crew know when it is time to launch the rocket. The Stamp's internal memory is used to store the data that is collected. I have enclosed a disk with the results of the first few flights. Note that the power source for the speed sensor is a set of watch batteries in the base of the nose cone.

The system is operable and can be brought to life by connecting the sets of red/black wires. While making up the drawing of the circuit, I noticed that I had mislabeled the RESET and +5 pins and attached things incorrectly. Fortunately the RESET pin seems to supply enough current to run the circuit and pressing the reset button in the nose cone shorts the +5 and Ground to do a type of reset!

The acceleration sensor works on the same principle as the first device except that it senses the brightness of the light that falls on a CdS cell in the top of the hypodermic syringe. The bulb moves up and down the syringe based on the acceleration of the rocket. The syringe is normally covered with black tape to keep out light. The small switch on the top of the gold cells is the power switch. The cells are 2/3 of a 9-volt battery. They serve as a power source and a weight for the sensor.

The data is collected by connecting a small PC (I used an HP 95LX) via the serial port. The data is continuously output after the flight.

Figure 12.1 shows the basic construction of Mr. Bodnar's instrumented model rockets, while Listings 12.1 and 12.2 are the programs that run them.

Figure 12.1: Diagrams of the velocity and acceleration probes.
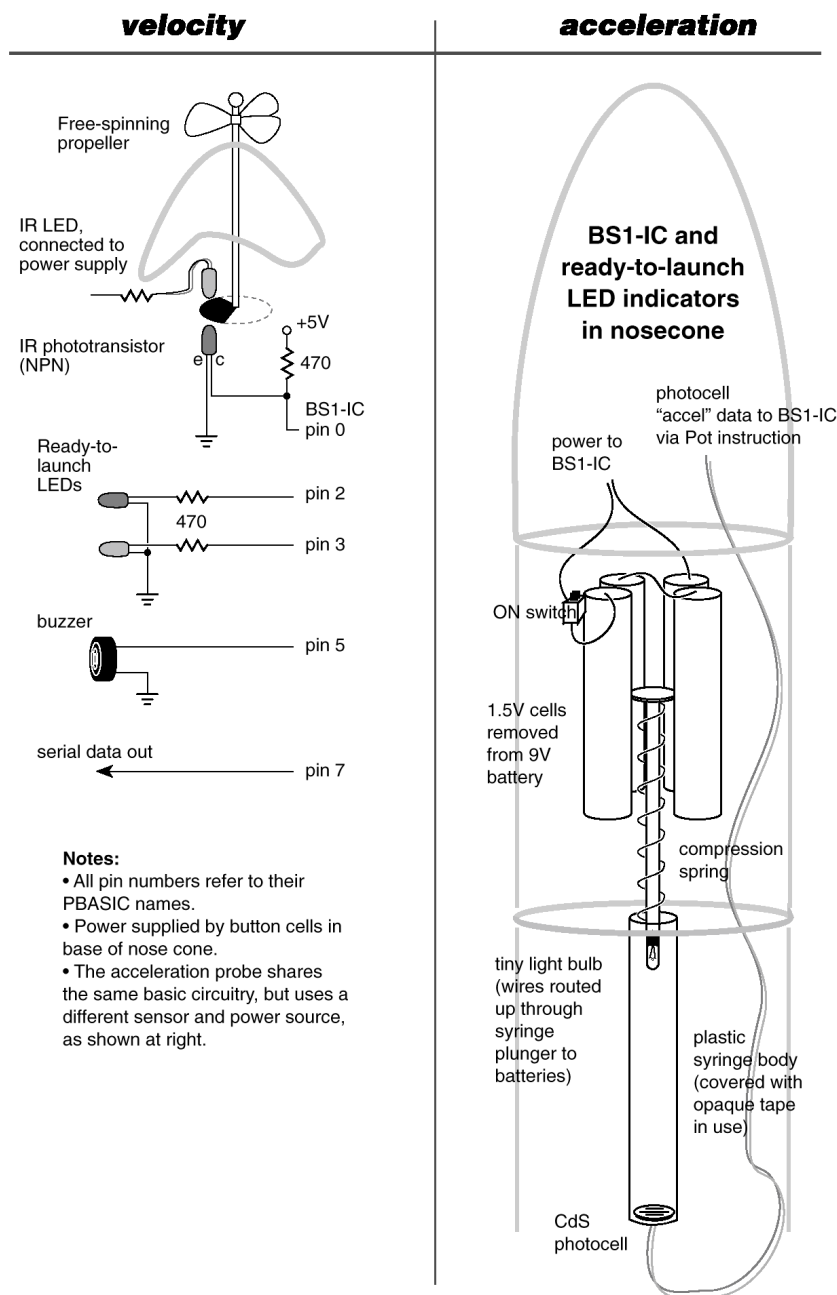
BASIC for Beginners

Last month, I equated a BASIC program to a to-do list. The computer starts at the top of the program and performs the instructions in order of appearance.

However, the real power of programming is the ability to make decisions. The program is still a list of actions to do, but the order in which they're performed (and whether some of them are performed at all) can be based on conditions that exist at the time the program is run.

In BASIC, you describe decisions and their consequences with the IF/THEN instruction. The *syntax* of this instruction—the correct way to use and express this instruction in your programs—is:

```
IF condition THEN label
```

In this syntax, *condition* is a comparison that can be true or false; for example $2 = 10$ (read as *two is equal to te*n) is an example of a false condition. The comparison operators, also called relational operators, available in PBASIC are:

= equal		<> not equal
> greater than		< less than
>= greater or equal		<= less or equal

If you have trouble remembering which way the < should point, just use this mnemonic: the symbol is the mouth of a greedy alligator who always snaps up the larger of the two meals offered. So x > y reads *x is greater than* y.

When PBASIC processes an IF/THEN instruction, it determines whether the condition is true or false. If it's true, the next instruction the program executes will be the one specified by the label following THEN. If the condition is false, PBASIC will just go on to the next instruction in the program in normal to-do list fashion.

Let's put this information to work in a simple application. Suppose we're monitoring a burglar alarm system that's wired so that a 0 appears on pin1 when all doors and windows are shut, and a 1 when any are opened. When the system is turned on, opening a door or window should trigger the alarm. Here's a straightforward way to express that in PBASIC:

```
Monitor:
   IF pin1 = 1 THEN Alarm
GOTO Monitor
Alarm:
   SOUND 7,(90,100,120,100)
GOTO Alarm
```

In the section labeled Monitor, if pin1 is 0, meaning all is secure, the program just goes back to Monitor and checks the pin again. However, if pin1 is 1, meaning a door or window has been opened, the program goes to the instructions that start with the label Alarm. Here PBASIC generates a two-tone sound through pin7 to attract attention. (Don't worry about understanding the specifics of instructions like Sound—they're just fill-in-the blank forms for performing a simple action.)

Before I go on, I want to explain what labels like Monitor and Alarm are and how they work. You can see from the example that decisions and repeated actions depend on telling PBASIC to go to a specific point in the list of instructions that makes up a program. In old versions of BASIC each line began with a number, so you specified the "where" of a GOTO or IF/THEN instruction with the number of the destination line.

Line numbers don't say much about the "what" or "why" of these destinations, so modern BASICs like PBASIC use labels instead. A label is just a word that you, the programmer, have picked to mark a place in the program. To let PBASIC know that it's a label, you end the chosen word with a colon (:). There are other rules about labels, but we'll leave it at that for now.

I want to leave you with a further thought about IF/THEN: Suppose we wanted another condition that could trigger the alarm—say a panic button on pin2. It would normally read 0, but if you heard a strange noise in the bushes, you could push the button to put a 1 on pin2 to trigger the alarm. In other words, you want the alarm to go off if there's a door/window open OR the panic button is pressed. A small change to the IF/THEN instruction does the job:

```
Monitor:
   IF pin1 = 1 OR pin2 = 1 THEN Alarm
GOTO Monitor
```

Looky there: the idea of OR is expressed the same in PBASIC as in plain English! From this simple foundation, we'll examine the whole system of truth and consequences called Boolean logic in the next installment.

Column #12: Model Rocket Project Aims High with BS1-IC Instrumentation

```
' Listing 12.1, Model Rocket Velocity Probe, by Dave Bodnar
REM Rocket telemetry program MINIMUM memory implementation
REM for use with PROPELLER/SPEED sensor only
REM uses internal STAMP memory for data storage
' RKT_SPD1.BAS (was RKT_MIN5.BAS) D. Bodnar 7-6-95 6:53
Symbol Prop = 0        'pin 0 for propellor
Symbol LED1 = 2        'pin to show ready for launch
Symbol LED2 = 3        'pin (another) to show ready for launch
Symbol Buzz = 5        'pin for Piezo buzzer
Symbol Ser_out = 7     'pin for serial output of data (orange wire)
Symbol Delay = 8000    'better at about 8000 - 1000 for testing only
Symbol STOPmem=2
realstart:
  read 255,b11 'memory end location for writing
'debug b11,cr
  b11=b11-1
  b10=b11                'make copy for later
  LOW buzz
  high LED2:high LED1 'Both on FIRST
  pause Delay            'RED only on (LED1)
  HIGH buzz
  low LED2               'GREEN only on (LED2)
  pause Delay            'LAUNCH ready
  LOW buzz
  high LED2:low LED1
  pause Delay
  Low LED2
  HIGH buzz

waitforlaunch: 'stay here till LAUNCH detected
  high LED1              'flash RED while waiting
  pulsin prop,1,w2
  low LED1
  if w2=0 then waitforlaunch:
  High LED2
  LOW buzz
start:
gosub GETnWRITEit:
  if b11 >STOPmem then start: 'loop till RAM full
  Low LED2
  HIGH buzz
doneloop:
  b9=b10                 'debug "start of end",cr
  LOW buzz
  serout ser_out,n2400,("START",#b9,13,10) 'send "Start" & amount of RAM
  pause 2000
loop:
  HIGH buzz
  read b9, b7:b9=b9-1:read b9,b8
' debug #b9,"+1 hi=",#b7," lo=",#b8
  w2=b7*256 + b8
' debug " ",#w2,cr
  LOW buzz
  serout ser_out, n2400,(#w2,13,10)
```

```
' high LED1:Pause 20:low LED1:pause 20
  b9=b9-1
  if b9 > STOPmem then loop
   pause 5000
goto doneloop:
GETnWRITEit:
  HIGH buzz:PAUSE 40
  pulsin prop, 1,w2 'take reading
  b7=w2/256
  w1= b7*256
  b8=w2-w1
' debug #b11,"hi=",#b7," lo=",#b8, " 16bit=",#w2,cr
  Write b11,b7:b11=b11-1:write b11,b8:b11=b11-1
  LOW buzz:PAUSE 40
Return
```

```
' Listing 12.2, Model Rocket Acceleration Probe, by Dave Bodnar
REM Rocket telemetry program MINIMUM memory implementation
REM uses internal STAMP memory for data storage
' RKT_ACC1.BAS (was RKT_MIN5.BAS) D. Bodnar 7-6-95 6:27
Symbol Accel = 1      'pin 1 for acceleration
Symbol LED1 = 2       'pin to show ready for launch
Symbol LED2 = 3       'pin (another) to show ready for launch
Symbol Ser_out = 7    'pin for serial output of data (orange wire)
Symbol Delay = 8000   'better at about 8000 - 1000 for testing only
realstart:
  read 255,b11 'memory end location for writing
  b10=b11-1              'make copy for later
  high LED2:high LED1 'Both on FIRST
  pause Delay           'RED only on (LED1)
  low LED2              'GREEN only on (LED2)
  pause Delay           'LAUNCH ready
  high LED2:low LED1
  pause Delay
  Low LED2
  gosub GETnWRITEit:
  b6=b2-3
waitforlaunch: 'stay here till LAUNCH detected
  high LED1            'flash RED while waiting
  pot accel,170,b2
  low LED1
  if b2>b6 then waitforlaunch:
High LED2
start:
  gosub GETnWRITEit:
  low LED2:pause 20:high LED2:pause 5
  if b11 >1 then start: 'loop till RAM full
  Low LED2
doneloop:
  b9=b10
  serout ser_out,n2400,("S",#b9,13,10) 'send "Start" & amount of RAM
  pause 2000
loop:
```

```
  read b9, b2:
  serout ser_out, n2400,(#b2,13,10)
  high LED1:Pause 20:low LED1:pause 20
  b9=b9-1
  if b9 > 1 then loop
  pause 5000
  goto doneloop:
GETnWRITEit:
  pot accel,170,b2                  'take reading
  Write b11,b2:b11=b11-1
Return
```