Column #26, April 1997 by Scott Edwards:

Stamp Gives the Green Light To Efficient Programming

THE *ELECTRONICS Q&A* column here in *N&V* is an amazing resource. *Q&A* editor T. J. Byers will go to any length to find the answers to his readers' questions. Recently, he came to me.

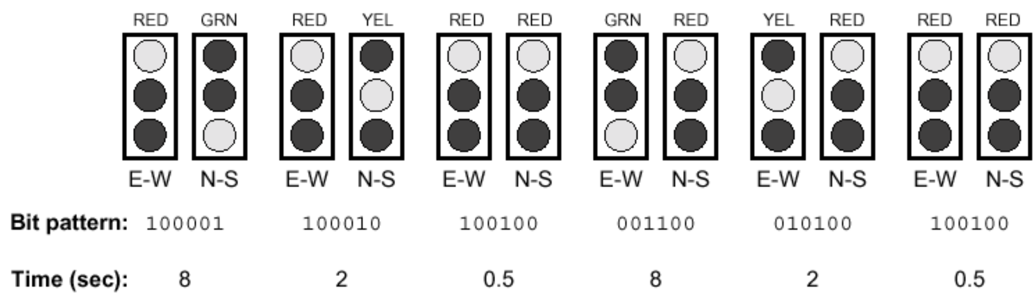
A reader had come into possession of a real stoplight, and wanted to know how to build a circuit that would realistically sequence the red, yellow and green lights. T.J. half kiddingly suggested a player-piano arrangement of motors, cams and switches, and referred the question to me for a Stampified solution.

So this month we'll learn how to sequence a traffic light, with special emphasis on storing and retrieving data with Lookup tables. We'll also have a peek at new Stamp peripherals that store data, keep time, and control motors.

Playing Traffic Cop

It hardly seems necessary to discuss what a traffic signal does, since we spend way too much of our time looking at examples—usually lit up red in our direction for an interminable time.

Figure 26.1: Stoplight sequence



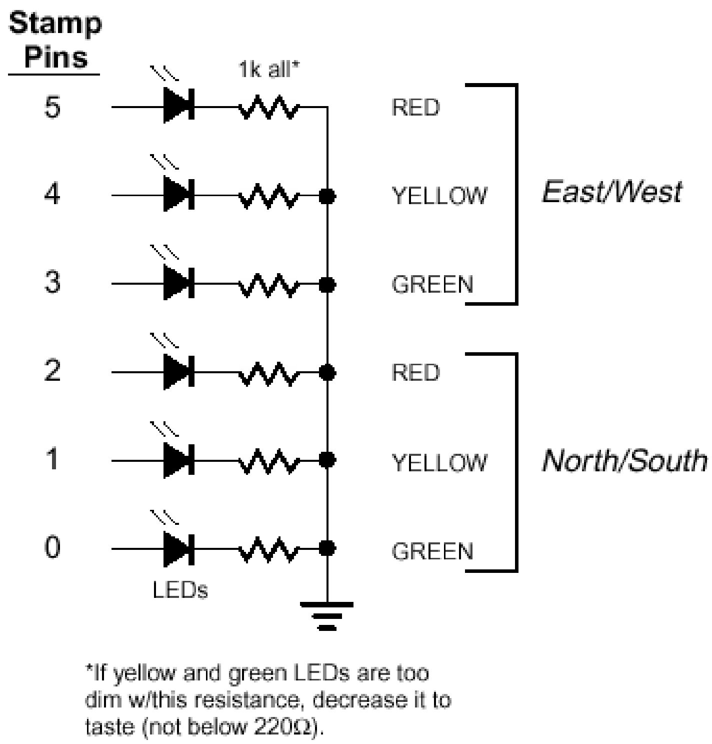
But it's my habit to describe a problem by making sketches and jotting notes and calculations before I set out to write a program. In this case, I drew a pairs of traffic signals at a hypothetical intersection. One light would control a north-south street, the other east-west.

I identified six states for the lights in a normal traffic sequence, as shown in Figure 26.1. For the sake of simplicity, I decided that this intersection would be the timer-controlled variety, not demand-controlled by the presence or absence of traffic. After all, the reader probably wants his light to sequence continuously, without the need for somebody to pull up in a Chevy.

The lights remain in each of the six states for varying amounts of time, ranging from less than a second for both-red, through 2 seconds for yellow, to 8 seconds for red/green. I picked the times arbitrarily. I made a note to make sure that the program allowed any timing parameter to be changed easily.

Figure 26.2 shows how I rigged a simulated stoplight with red, yellow and green LEDs. Note that you may have to fiddle with the series resistor values in order to get more-or-less equal brightness from the three different colors of LEDs. Each color of LED has a different forward voltage and efficiency.

Figure 26.2: Hookup for program listings 26.1 and 26.2



Equipped with my two models—a mental model of stoplight operation and a physical model of the lights themselves—I was ready to program.

Looking at my sketch (Figure 26.1), I determined that the job boiled down to retrieving two pieces of information from a lookup table; the patterns of the six lights and the length of time they should remain in that pattern. PBASIC includes a Lookup instruction that allows you to fetch data from a table based on its position or *inde*x.An obvious approach would be to prepare two lookup tables, one with bit patterns and the other with times.

However, I wanted to illustrate a couple of PBASIC capabilities that many users forget: (1) Lookup-table entries can be up to 16 bits long, and (2) The STAMP2 host program can perform compile-time math that can make a program more readable without taking up additional program memory.

Listings 26.1 and 26.2 are the result. The programs are thoroughly commented, so I won't repeat that stuff here. Suffice to say that these are very compact programs with plenty of room left over for your customization.

```
' Program Listing 26.1. Stoplight control for BS1
' Program: STOPLITE.BAS (Sequence a stoplight from a lookup table.)
' This program generates proper green-yellow-red sequencing for a
' pair of traffic signals controlling an intersection. I refer
' to one street as "EW" (east-west) and the other as "NS" (north-
' south). Pins are connected to LEDs as follows:
'      pin5   EW/red         pin2   NS/red
'      pin4   EW/yellow      pin1   NS/yellow
'      pin3   EW/green       pin0   NS/green
' ====Constants===
' The program uses six 16-bit constants to represent the states
' of the lights (lower 6 bits) and the length of time to leave
' the lights in those states (upper 10 bits). The usual way
' to create such constants is to define the bit patterns
' and the times separately, then have the compiler add or
' logically OR them together. Unfortunately, the simple STAMP
' host program doesn't have this feature, so we'll have to do
' it by hand. Here's how the constants are organized:
'              Duration (ms) Pattern of lights
'                        \     /
'                 |=========|====|
SYMBOL NSgo    = %0010000000100001   ' NS green/EW red, 8192 ms.
SYMBOL NSyel   = %0000100000100010   ' NS yellow/EW red, 2048 ms.
SYMBOL allRed  = %0000001000100100   ' NS red/EW red, 512 ms.
SYMBOL EWgo    = %0010000000001100   ' NS red/EW green, 8192 ms.
SYMBOL EWyel   = %0000100000010100   ' NS red/EW yellow, 2048 ms.
' ===Variables===
SYMBOL seq = b11                     ' Current state (0-5) of sequence.
SYMBOL lkup = w4                     ' Number from lookup table.
' ===Program===
dirs = %00111111                     ' Set lower six pins to output.
again:                               ' Endless loop.
for seq = 0 to 5              ' For each of six stored patterns/times..
  lookup seq,(NSgo,NSyel,allRed,EWgo,EWyel,allRed),lkup ' Get bits.
  pins = lkup & %00111111     ' Copy lower 6 bits to pins.
  lkup = lkup & %1111111111000000 ' Strip off lower 6 bits.
  pause lkup                  ' Set delay to upper 10 bits.
next                          ' ..and get the next entry from the table.
goto again                    ' Repeat endlessly.
```

```
' Program Listing 26.2. Stoplight control for BS2
' Program: STOPLITE.BS2 (Sequence a stoplight from a lookup table.)
' This program generates proper green-yellow-red sequencing for a
' pair of traffic signals controlling an intersection. I refer
' to one street as "EW" (east-west) and the other as "NS" (north-
' south). Pins are connected to LEDs as follows:
'       P5      EW/red         P2      NS/red
'       P4      EW/yellow      P1      NS/yellow
'       P3      EW/green       P0      NS/green
' ====Constants===
' The program uses six 16-bit constants to represent the states
' of the lights (lower 6 bits) and the length of time to leave
' the lights in those states (upper 10 bits). Here's how the
' constants are organized:
'             Duration (ms) Pattern of lights
'                        \          /
'                    |=========|====|
' The BS2 host software permits compile-time math (math done on
' the PC before downloading to the Stamp), which we'll use to
' combine two sets of constants--one representing light patterns
' and another times. This allows you to change the timing of
' the lights (or the bit patterns, if you wired the lights
' differently) without worrying about how the bits are packed
' into their 16-bit packages.

NSgrn          con %00100001          ' Make NS green, EW red.
NSyel          con %00100010          ' Make NS yellow, EW red.
allRed         con %00100100          ' Make both lights red.
EWgrn          con %00001100          ' Make EW green, NS red.
EWyel          con %00010100          ' Make EW yellow, NS red.
NsgoTime       con 8192               ' Set NS green duration
YelTime        con 2048               ' Set duration of any yellow.
EWgoTime       con 8192               ' Set EW green duration.
redOverlap     con 512                ' Set red/red overlap time.

' The bit-pattern and timing constants are combined as follows:
' The time is logically ANDed with %1111111111000000, which
' clears the lower 6 bits to 0s while leaving the upper 10
' bits intact. The result is logically ORed with the 6-bit
' light pattern, which copies the 1s of the pattern into the
' lower 6 bits. If this ANDing and ORing is unfamiliar, check
' out Stamp Applications #14, April 1996 for a quick lesson
' in Boolean logic. (See the N&V web site or contact the
' magazine for back issues.)

top10          con %1111111111000000       ' Mask off lower 6 bits.
btm6           con %0000000000111111       ' Mask off upper 10 bits.
NSgo           con NSgoTime & top10 | NSgrn ' 16-bit time/bit pat.
NSwarn         con yelTime & top10 | NSyel         ' "
allStop        con RedOverlap & top10 | allRed     ' "
EWgo           con EWgoTime & top10 | EWgrn        ' "
EWwarn         con yelTime & top10 | EWyel         ' "
' ===Variables===
seq    var    nib    ' Current state (0-5) of stoplight sequence.
```

```
lkup    var     word    ' Number from lookup table.
' ===Program===
DIRS = %00111111       ' Set lower six pins to output.
again:                 ' Endless loop.
for seq = 0 to 5       ' For each of six stored patterns/times..
  lookup seq,[NSgo,NSwarn,allStop,EWgo,EWwarn,allStop],lkup ' Get bits.
  OUTS = lkup & btm6  ' Copy lower 6 bits to pins.
  pause lkup & top10  ' Set delay to upper 10 bits.
next                   ' ..and get the next entry from the table.
goto again             ' Repeat endlessly.
```