8-bit Breadboard Computer

Introduction

The intention is that this document will serve as a reference manual, with enough detail to fully refresh the concepts that govern the operation of the CPU, while avoiding being a tome that covers every detail from the YouTube videos.

1 Clock Module

The clock module permits variable speeds, as well as single clock pulses using a pushbutton. This section will cover all iterations of the timer: *astable*, *bistable*, and *monostable*

1.1 Astable 555 Timer

In this configuration, the 555 timer is simply used with two resistors and a capacitor to set the time the output will be off and on. Figure 1 shows the circuit diagram and voltage across the capacitor.

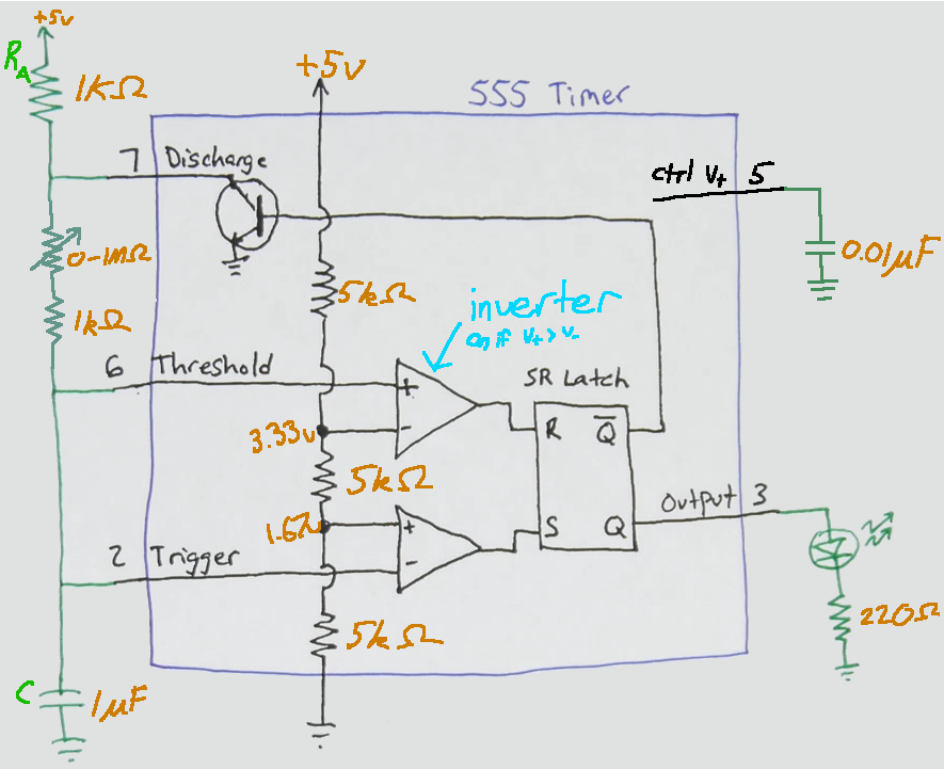

Figure 1: Astable 555 timer circuit diagram

The datasheet provides some eqution for determining the charge time (Equation 1), discharge time (Equation 2), total period (Equation 3), and frequency of oscillation (Equation 4).

$$t_1 = 0.693(R_A + R_B)C \tag{1}$$

$$t_2 = 0.693(R_B)C \tag{2}$$

$$T = t_1 + t_2 = 0.693(R_A + 2R_B)C \tag{3}$$

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)C} \tag{4}$$

The completed circuit is shown in Figure 2. Before the last few capacitors were added to the circuit, the oscilloscope probes were hooked to the positive lead of the capacitor, as well as the output of the SR latch. This data is shown in Figure 3. Note: the values on both scales of the oscilloscope data are not accurate, it's the shape that's important.

Of particular note in Figure 3 is the overshoot on the output of the SR latch. This is largely caused by the impedence and inductance in the relatively long power cable, as it causes a delay in the time for the power supply to supply enough power to turn the LED on (and do the other thing that happen inside the 555 chip). This can be remedied by placing a 0.1μF capacitor across the power lines of the breadboard. Similarly, the datasheet recommends placing a 0.01μF capacitor from pin 5 to GND, to reduce noise. A plot of this data is not included as it does not show up very clearly on a cheap oscilloscope.

Finally, note the variable resistor (in series with a 1kω resistor) in Figure 1, which allows the speed of the clock to be varied by controlling the flow of current into and out of the capacitor.

Table 1 shows the operation cycle of the circuit.

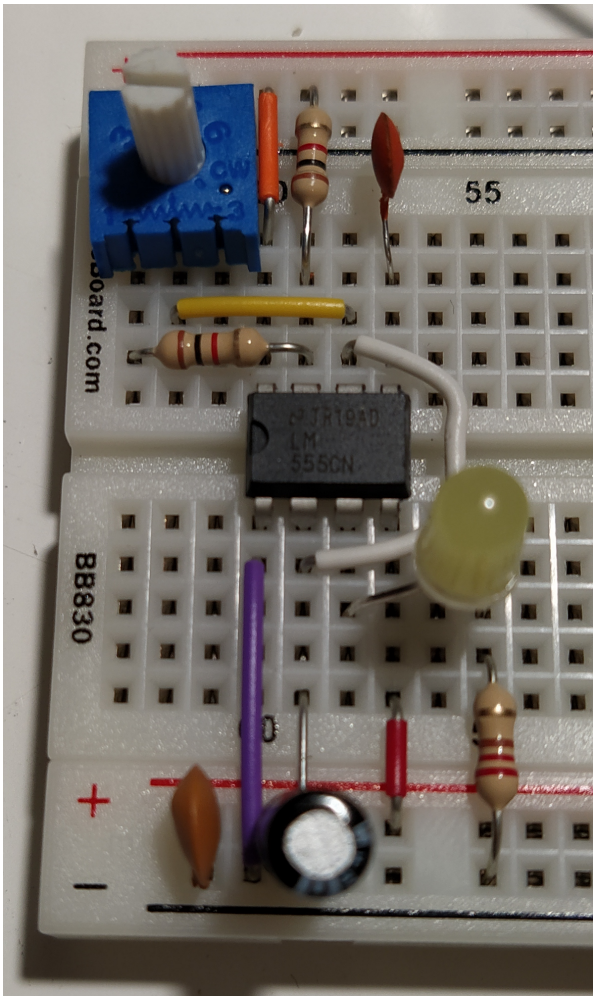

Figure 2: Astable 555 timer circuit

Time	Circuit Status	Capacitor Voltage	S	R	Q	$\overline{Q}$	Transistor
t_0	Power on	0	1	0	1	0	Off
t_1	Charging	> 1.67	0	0	1	0	Off
t_2	Discharging	> 3.33	0	1	0	1	On
t_3	Discharging	< 3.33	0	0	0	1	On
t_4	Charging	< 1.67	1	0	1	0	Off

Table 1: Astable 555 timer operation cycle

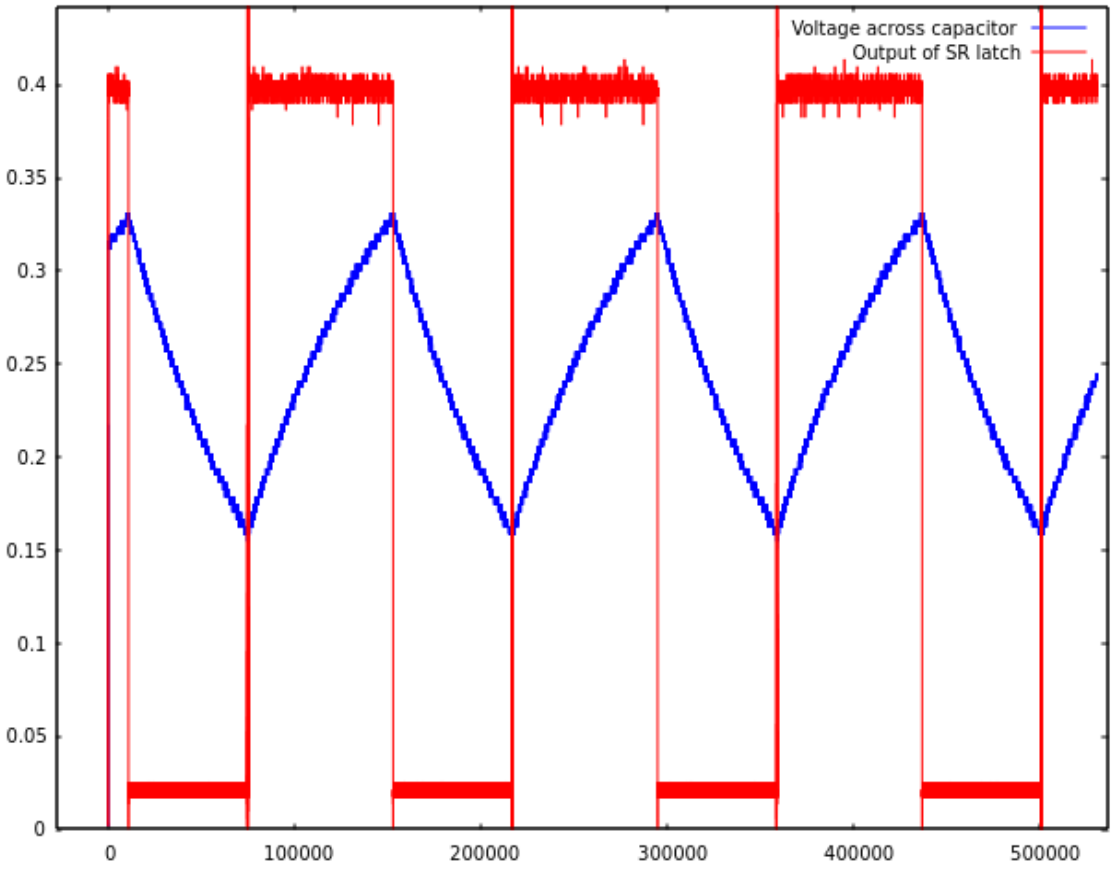

Figure 3: Astable 555 timer oscilloscope data

1.2 Monostable 555 Timer

To allow us to manually advance the clock by one clock cycle, we will use a 555 timer to debounce a pushbutton. This mode of operation is referred to as monostable operation, as the circuit settles to a stable state after the capacitor is discharged.

The monostable circuit diagram is shown in Figure 4, and the physical circuit is shown in Figure 5. The operation cycle is shown in Table 2, and a plot of data from the oscilloscope is shown in Figure 6.

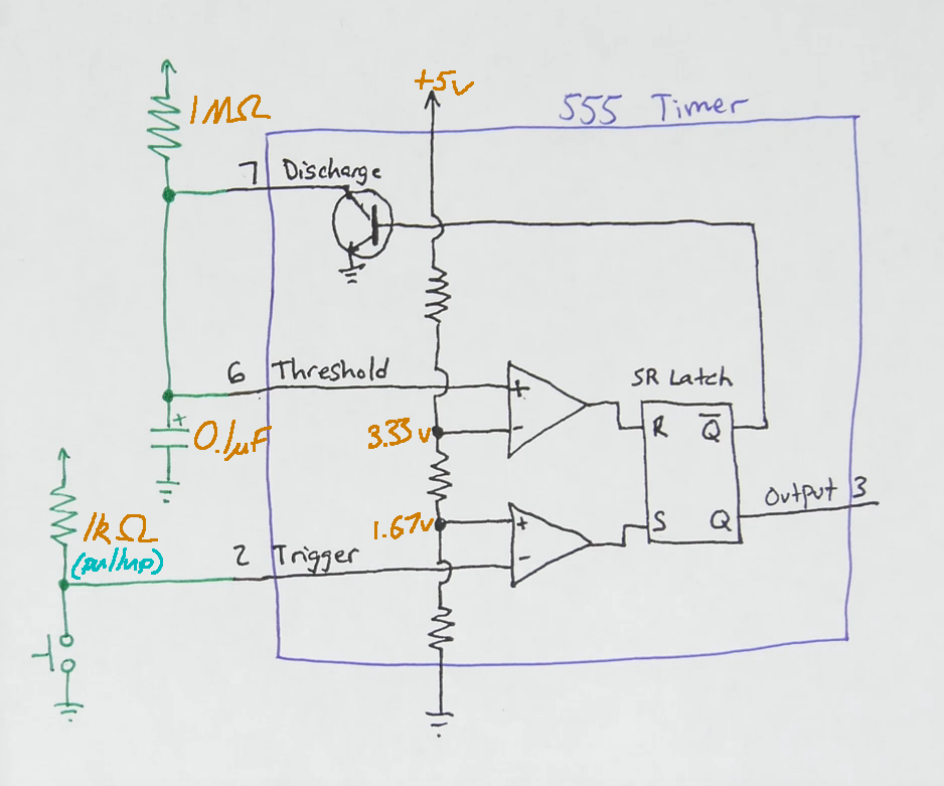

Figure 4: Monostable 555 timer circuit

Time	Circuit Status	Capacitor Voltage	S	R	Q	$\overline{Q}$	Transistor
t_0	Power on	0	0	0	x	x	x
t_1	Stable	0	0	0	0	1	On
t_2	Button Pressed	0	1	0	1	0	Off
t_3	Button Released	< 3.33	0	0	1	0	Off
t_4	Discharge	> 3.33	0	1	0	1	On

Table 2: Monostable 555 timer operation cycle

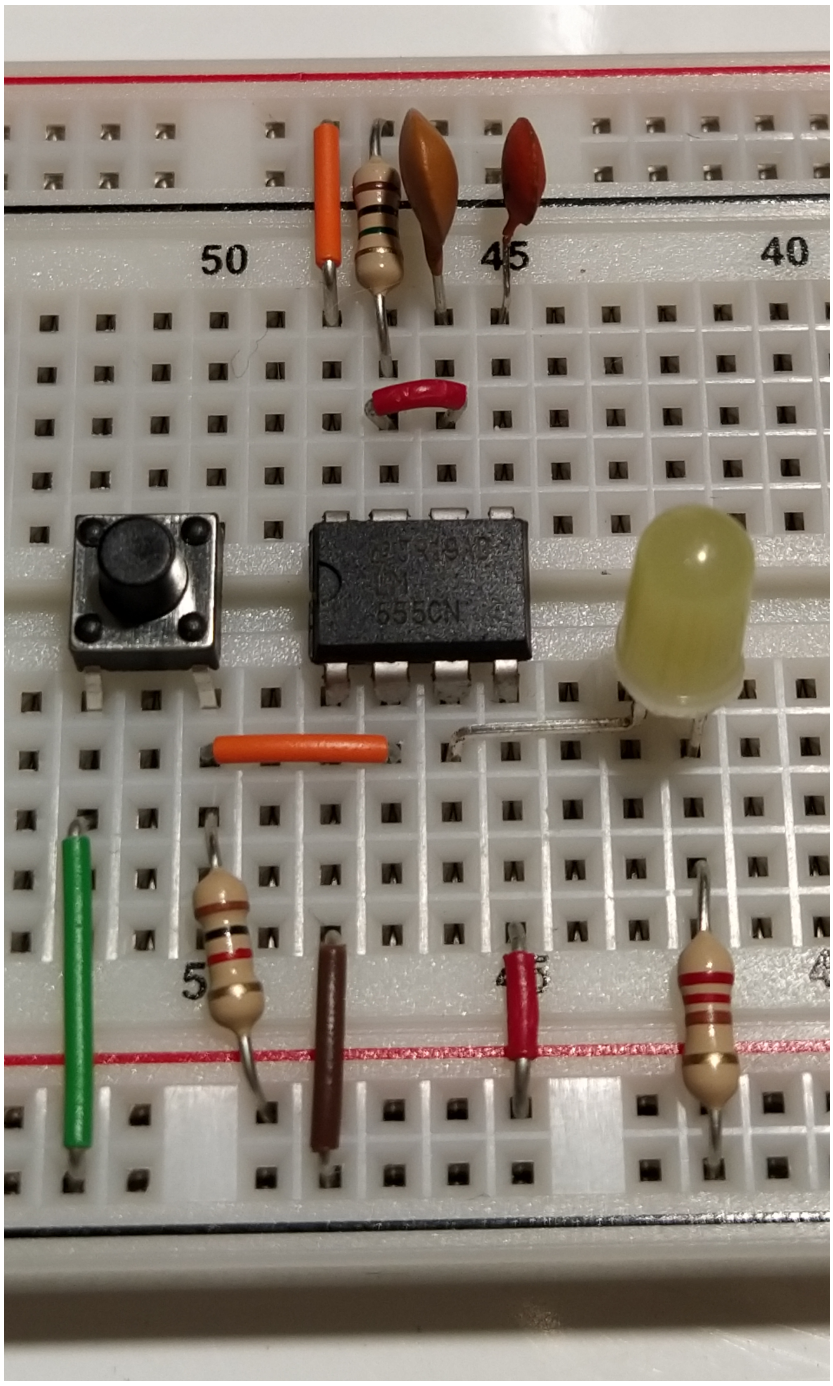

Figure 5: Monostable 555 timer circuit

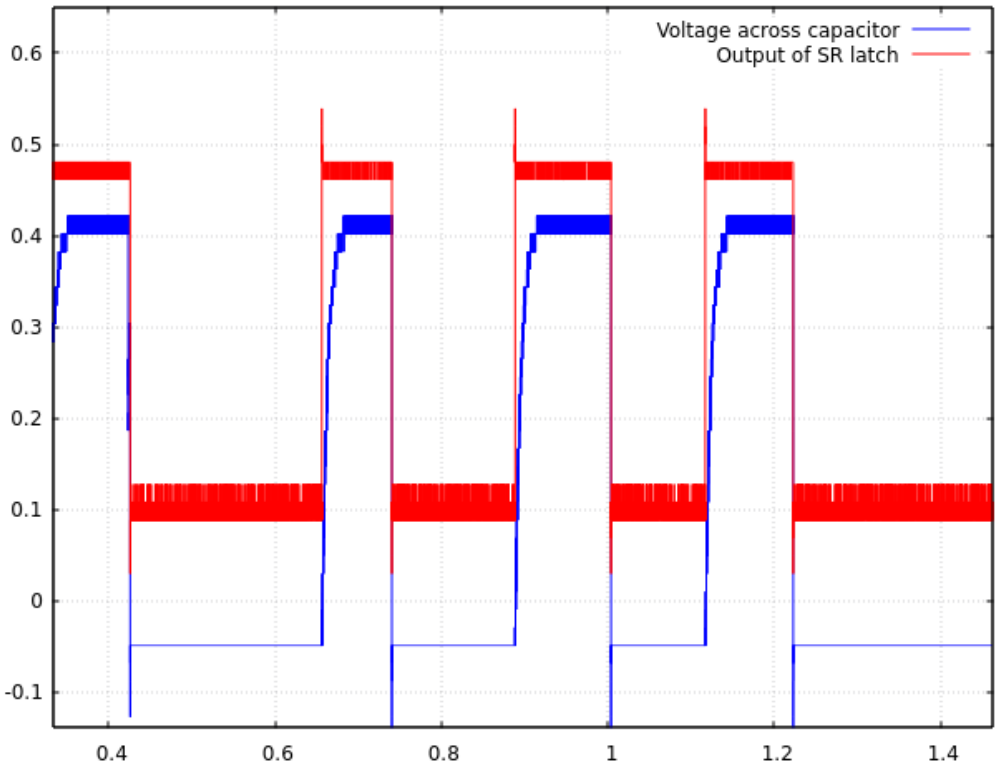

Figure 6: Monostable 555 timer circuit

1.3 Bistable 555 Timer

We need to be able to switch between our two clocks, which we can use a switch and some logic for. The switch, however, must be guaranteed to not bounce, or otherwise we could end up multiple clock cycles ahead of where we want to be.

To achieve this, we will just utilise the SR latch inside the 555 timer, and a *break-before-make* double throw switch, with the centre pole connected to ground, and the two contacts connected to the 555's *trigger* and *reset* lines respectively. The circuit diagram for this circuit is shown in Figure 7, and the real circuit is shown in Figure 8.

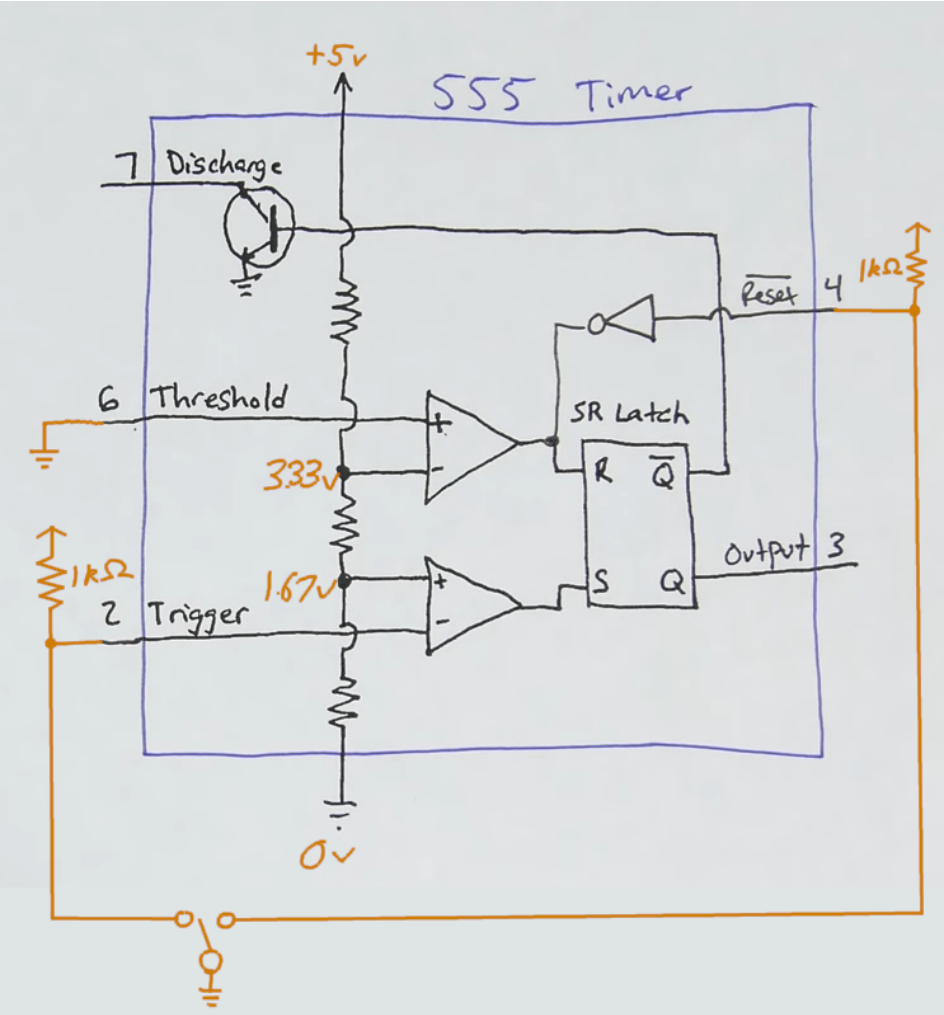

Figure 7: Bistable 555 timer circuit diagram

If we also tie the threshold line to ground (such that it is never $> 3.33V$), the *trigger* line being pulled low will set the latch, while the *reset* line being puled low will reset the latch. Because the switch is a *break-before-make* switch, any bouncing will bounce between floating and that side of the switch, but once the first contact has been made, the output of the latch will be static until the switch first touches the other contact. Thus we have an effective circuit for the switch, where one of the contacts sends the output high, while the other sends it low, with no bouncing.

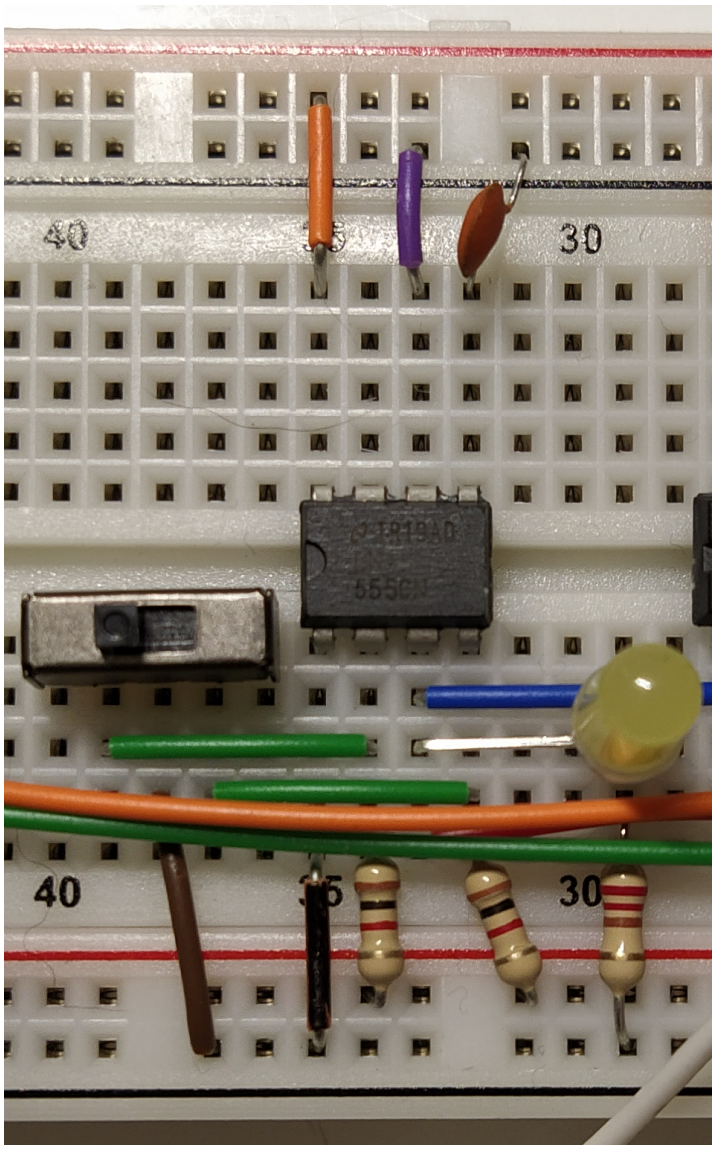

Figure 8: Bistable 555 timer circuit

1.4 Clock logic

We will use some combinational logic to combine our clock-select switch with the two clock signals. To achieve this, we will *AND* the astable clock with the clock-select signal, and the monostable clock with the inverted clock-select signal; *OR*ing the results together to produce the clock signal. Finally, we *AND* this with a halt signal (inverted) such that bringing the halt line high will stop the computer.

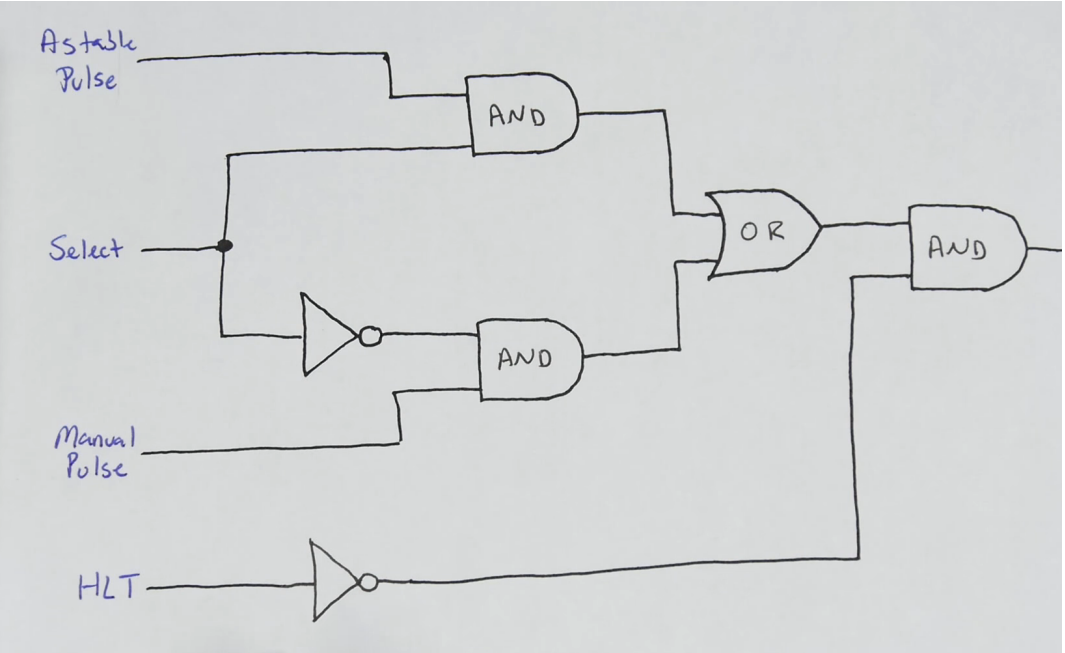

Figure 9: Clock logic diagram

Figure 11 shows the logic used in the clock module, and Figure **??** shows the implementation on the breadboard. This circuit (like any circuit) could be made entirely from NAND gates, saving a chip compared to the real implementation, shown in Figure **??**. For this project, saving one IC is not really of concern, and the complete NAND implementation is harder to follow, so the simple (conceptually) implementation was chosen.

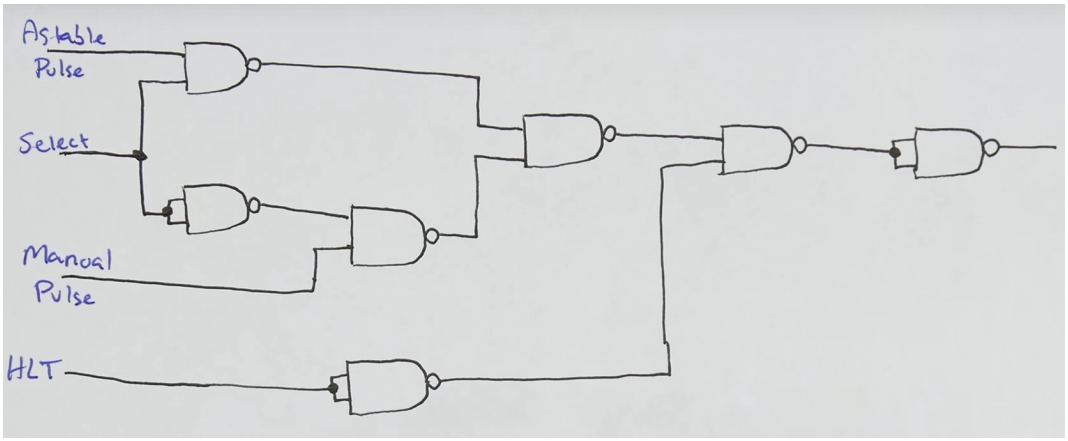

Figure 10: Clock logic diagram using only NAND gates

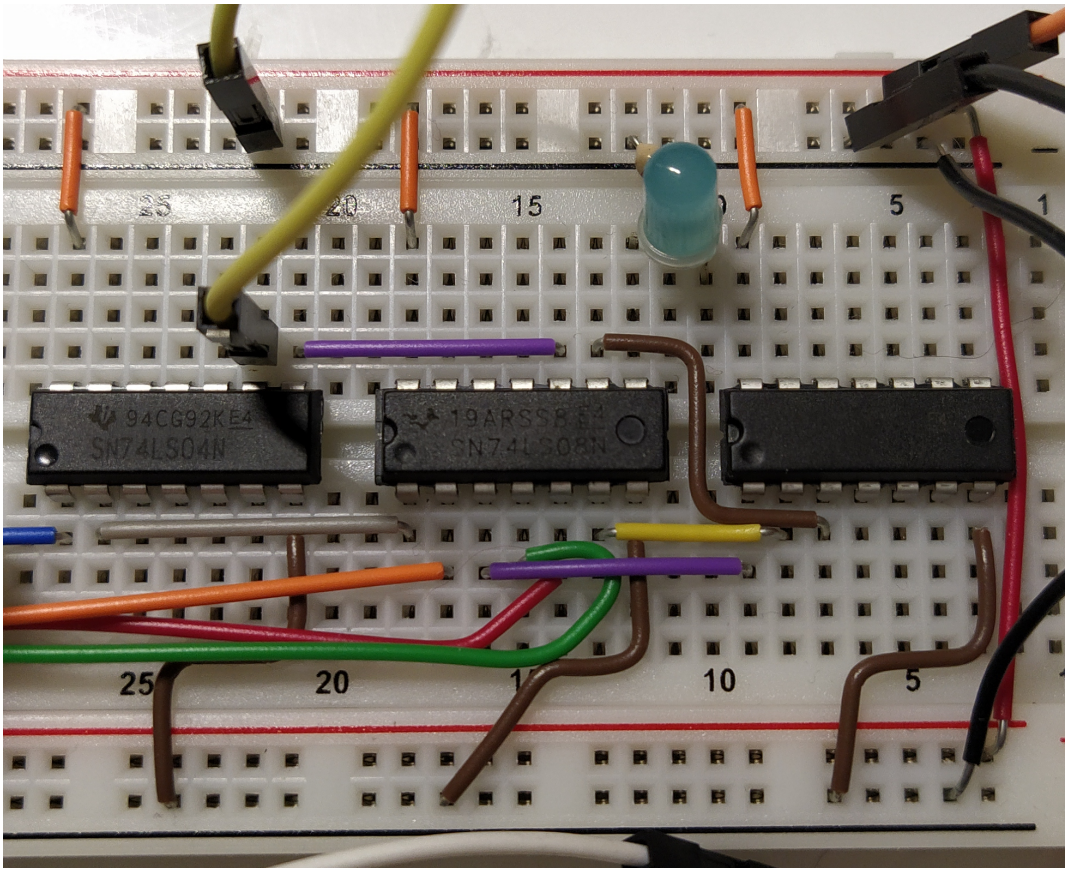

Figure 11: Clock logic

2 Registers

2.1 Register Architecture

Our computer will have three 8-bit registers. The A and B registers, for storing arbitrary data; and the *instruction* register, for storing the 4-bit instruction, and (optional) 4-bit argument, usually a memory address. The architecture of a single one of our registers' bits is a D flip-flop, with the input fed back from the output, unless an *enable* line is brought high. Two of these circuits are shown in Figure 12, showing the logic flow when the enable line is low and high.

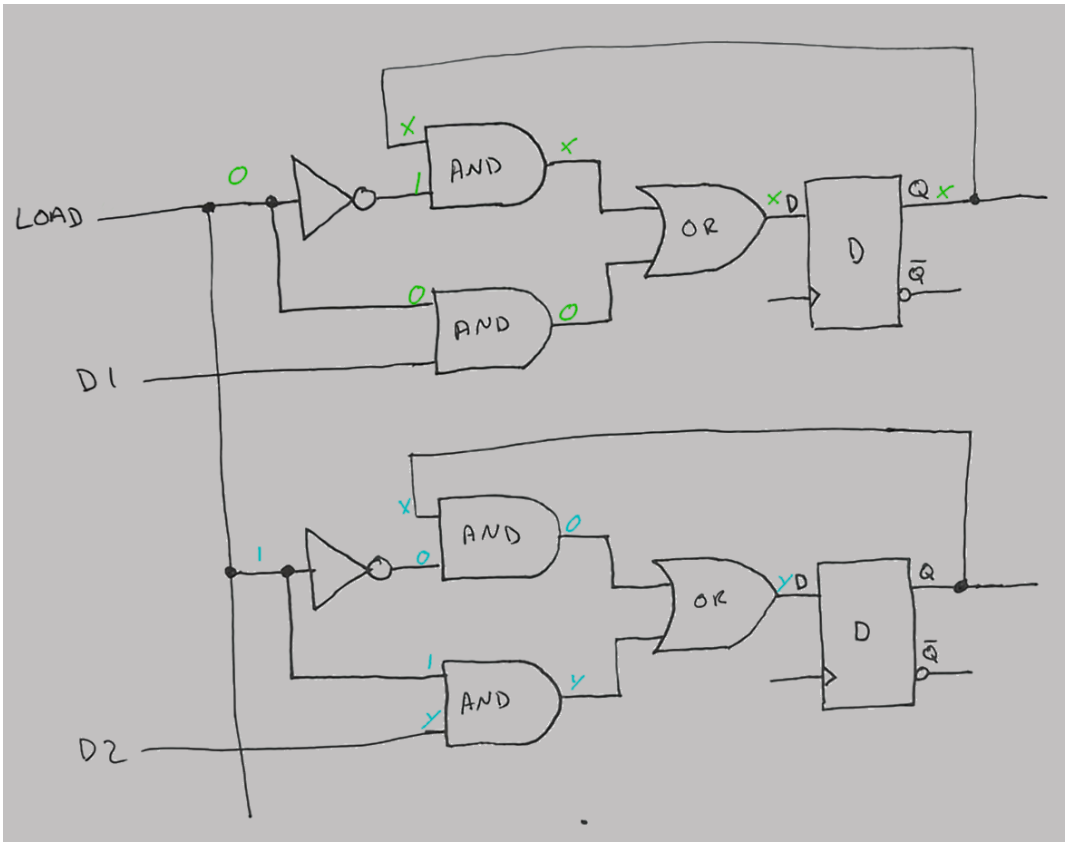

Figure 12: Register architecture

2.2 Register Implementation

Building the registers using discrete logic chips would take an inordinate number of breadboards, so in the interest of not wasting time, money, effort, or sanity, we will use two *74LS173* chips (per register), which each contain a prepackaged 4-bit register, using the exact architecture shown above. The circuit for each will be identical, except that the instruction register will only put the four least significant bits on the bus, as there is no situation in which it is necessary for it to put the instruction out on the bus. Figure 13 shows the circuit diagram for a data register.

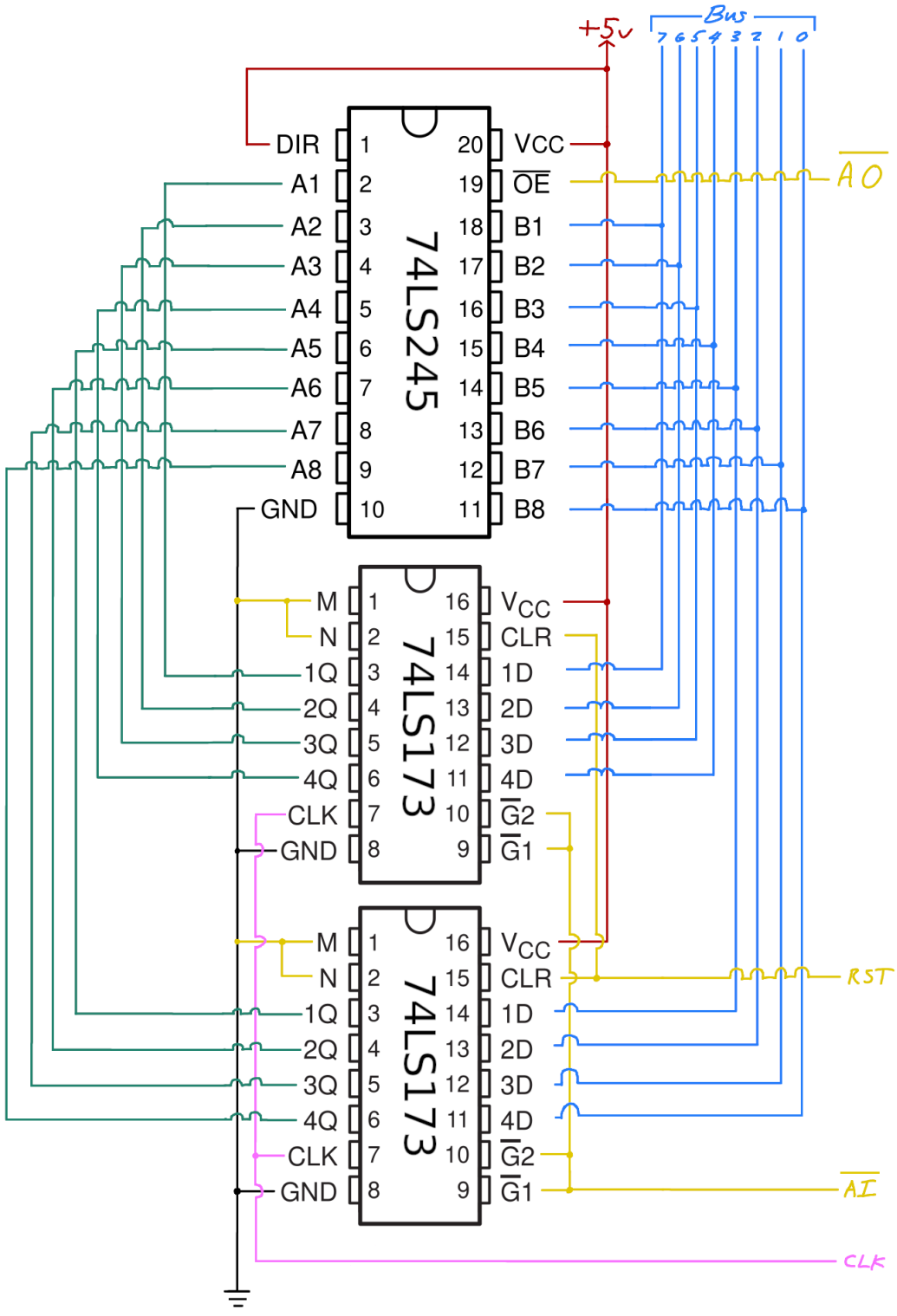

Figure 13: Register circuit diagram