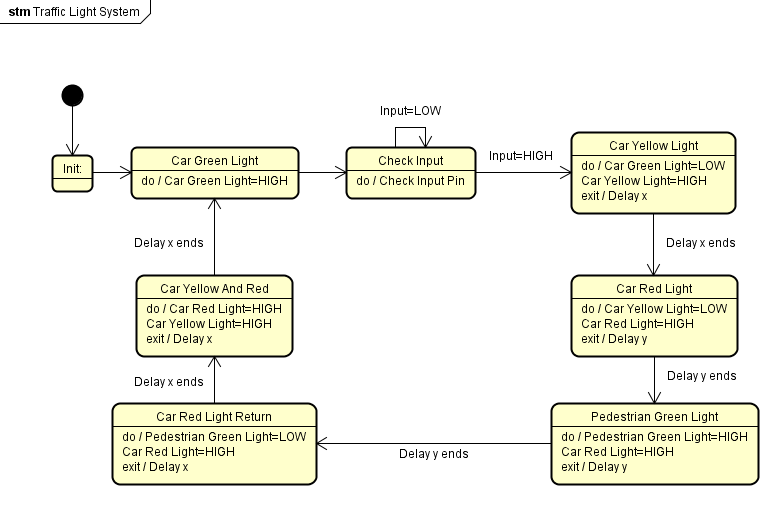
**Analysis of the problem**

This document aims to elaborate upon the problem laid out in the problem statement.

In order to solve the traffic light system problem, it should first be broken down into sub-problems, which can be done using a state-machine diagram:



This state-machine diagram outlines the flow of events from initial start to a completed loop of the system. Due to the nature of this system, there is no end to the state-machine. A small note regarding the “Delay y” shown above is that this delay, while not strictly necessary to solving the problem, better replicates the real-life system of a pedestrian crossing in that it is a very small delay to reduce potential accidents.

Now that the problem has been broken in to states, it can be split in to sub problems. These sub-problems are:

* How to produce output to a pin in AVR assembler
* How to read input from a pin in AVR assembler
* How to replicate a delay in AVR assembler

The last point can also be split in to two problems, since two different length delays are needed to solve this problem and since this fits better when assigning tasks to four members in a group.

Each problem will be solved using Atmel Studio’s simulation and then tested on a breadboard.

**How to produce output to a pin in AVR assembler**

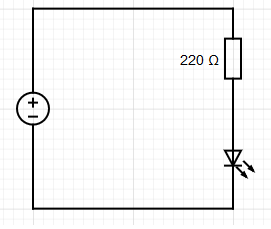
Output can be set on AVR assembler by first placing a value in memory using **LDI** (Load Immediate) and then **OUT** (Store register to I/O location). The 8-bit value placed in memory corresponds to the pin placement that will be set. By default, all pins are set to input which is the 8-bit value **0b0000 0000.** In order to change all pins on the **pinb** (for example) line to act as output I can use the following set of instructions:

**LDI R16 0b11111111**

**OUT ddrb R16**

**OUT portb R16**

In order to solve the traffic light problem, 4 outputs will be needed, each with an LED wired to them to serve as traffic lights. The circuit diagram for this is:



Using both the knowledge of pin setups found in the pinout document for the Arduino AT2560Mega and the above circuit diagram, the construction of the outputs can be completed.

**How to read input from a pin in AVR assembler**

Reading input from a pin requires that necessary pin be set to input mode. This is done by loading a value into a register then outputting the value to a pin. An example of this is:

**LDI R16 0b00000000**

**OUT ddrc R16**

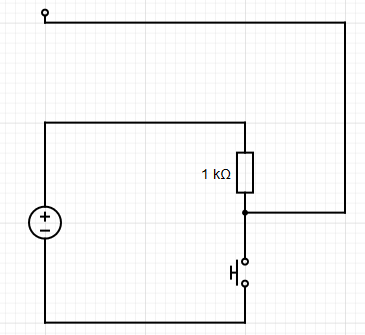
By default, all pins are already set as input, so this isn’t strictly necessary. The next step is to enable the internal pull-up resistors within the port, this is done by applying a high(1) value to that port as follows:

**LDI R16 0b11111111**

**OUT portc R16**

The pull-up resistors are used in order to set a default state to the pin. Without using the pull-up resistors, any amount of current between 0v and +5v may cause a false reading of that pin.

The circuit diagram of the switch being used to start the traffic light sequence is as follows:



The last step is how to read from the input pin that the switch is connected to. This is done by first reading the value from the pin, then comparing it to another value stored in memory:

**LDI R25 pinc**

**CPI R25 0b11111111**

From this, a loop for reading can be created which can be exited depending on the value read. This allows for a different flow of events based on the result of an input.

**How to replicate a delay in AVR assembly**

There are two main ways in which a delay can be created in AVR assembly using the Arduino AT2560Mega, timers and using all clock cycles in a loop. For the sake of this project, loops will be used to consume clock cycles to imitate a delay. The delays for this project do not need to be very specific so will not be calculated down to the exact number of clock cycles needed for a fixed period.

One way of creating a delay is to set a value in a register then decrement this value until it reaches 0, then break from this decrementing loop. One issue of this is that the highest value that can be stored in 8 bits is 255. With a small loop like the following:

**LDI R27 0xff**

**LoopStart:**

**DEC R27**

**BRNE LoopStart**

a small delay can be replicated. The problem with this method is that the delay is a very short duration due to the clock speed of the ATMega being 16mhz (16,000,000 clock cycles per second). The number of clock cycles of the above code is 764, so the delay this generates is roughly 0.05 millisecond. To increase the length of the delay, this loop can be nested inside of another so that this loop is executed a fixed number of times based on the value of the outer loop. An example of this is:

**LDI R26 0xff**

**First\_loop:**

**LDI R27 0xff**

**Second\_loop:**

**DEC R27**

**BRNE Second\_loop**

**DEC R26**

**BRNE First\_loop**

This code breaks after the second loop (which counts down from 255 to 0) has been executed 255 times, giving a total of 195.840 clock cycles used and a delay of 12,24 milliseconds. An additional layer of nesting can be added to allow for longer delays which will be of more use to this project.

With the knowledge of how to produce output, read input and generate a delay, all problems have been solved and the project can be physically built and coded.

**Test Plan**

Each aspect of this project will be tested both using Atmel Studio’s simulation option, as well as being physically tested on a breadboard. The simulation should allow better understanding of the number of clock cycles used by a single loop through the states and the breadboard will allow for the results to be seen visibly.

* The output will be tested on a breadboard by making an LED light up, as well as in the simulator to make sure that only the correct outputs are set to high.
* The input will be tested on a breadboard using a switch and an LED, as well as in the simulator where the input will be simulated by the changing of a value read by a pin.
* The delay will be tested on a breadboard by making an LED switch between being on and off with a delay between each, then on the simulator to check that the number of clock cycles being used matches what is expected.