University of Hull
School of Engineering and Computer Science
500089: Electronics and Interfacing

Lab 1: Introduction to Arduino Hardware Interfacing

Introduction

In this first lab is you will start to connect electronic components to an Arduino microcontroller and write programs to control the electronics. Most of the electronics you develop will be taken from the Arduino starter kit you will be given in the first laboratory.

These are available to take home, if any of your components break during the module please see a lab demonstrator or module tutor for a replacement. If you lose your kit then replacements can be bought from amazon. The kit contains most of the components you will need to study this module, any specialist components or modules needed to complete the assignment will be provided. You do not need to buy any hardware to complete the module, however if you like to do most of your assignment work at home it may be worth investing in some simple tools.

Please follow the instructions carefully for all the exercises. If you get the wiring wrong your programs will not work and there is a chance that you will destroy the delicate circuitry in the device that you are using. We are using Arduino Uno compatible devices from Elegoo. These are interchangeable with Arduino devices.

Remember to bring a notebook to all labs and draw out every circuit before you build it. This is essential for debugging and will be the first thing a lab demonstrator asks to see if you request help.

In this laboratory you will:

- Connect the Arduino to the computer and configure the software.
- Use a breadboard to make a simple electronic circuit
- Write a program to control an electronic device, in this lab an LED and seven segment display

These laboratories are built on work done by Rob Miles, Peter Robinson and James Walker who have all taught this module in the past, however there are changes this year to specifically guide you towards the assignment.

Getting Started

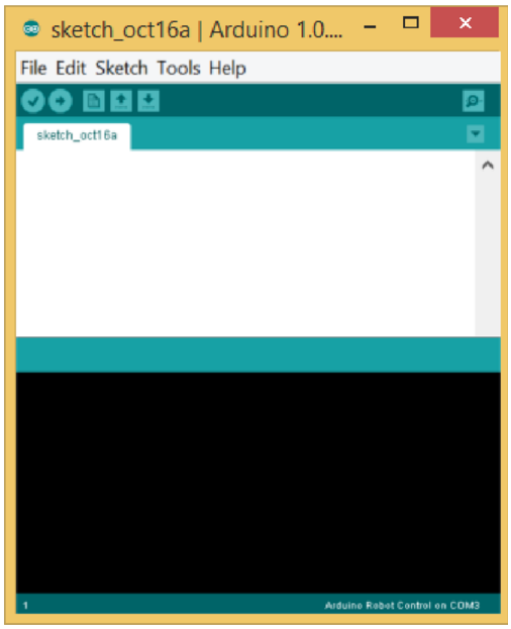
The Arduino uses a serial connection to the host computer. This is implemented over a USB interface. The first thing we need to do is connect the Arduino to the computer.

1. Log in to the computer using your username and password.
2. Plug one end of the USB cable into a socket on the front of your computer and the other end into the socket on the Arduino. You may see the computer load some drivers the first time that you do this.
3. If you are using a Macintosh or Linux PC you may have to do some extra work to get the serial connection to work. You can find help here:

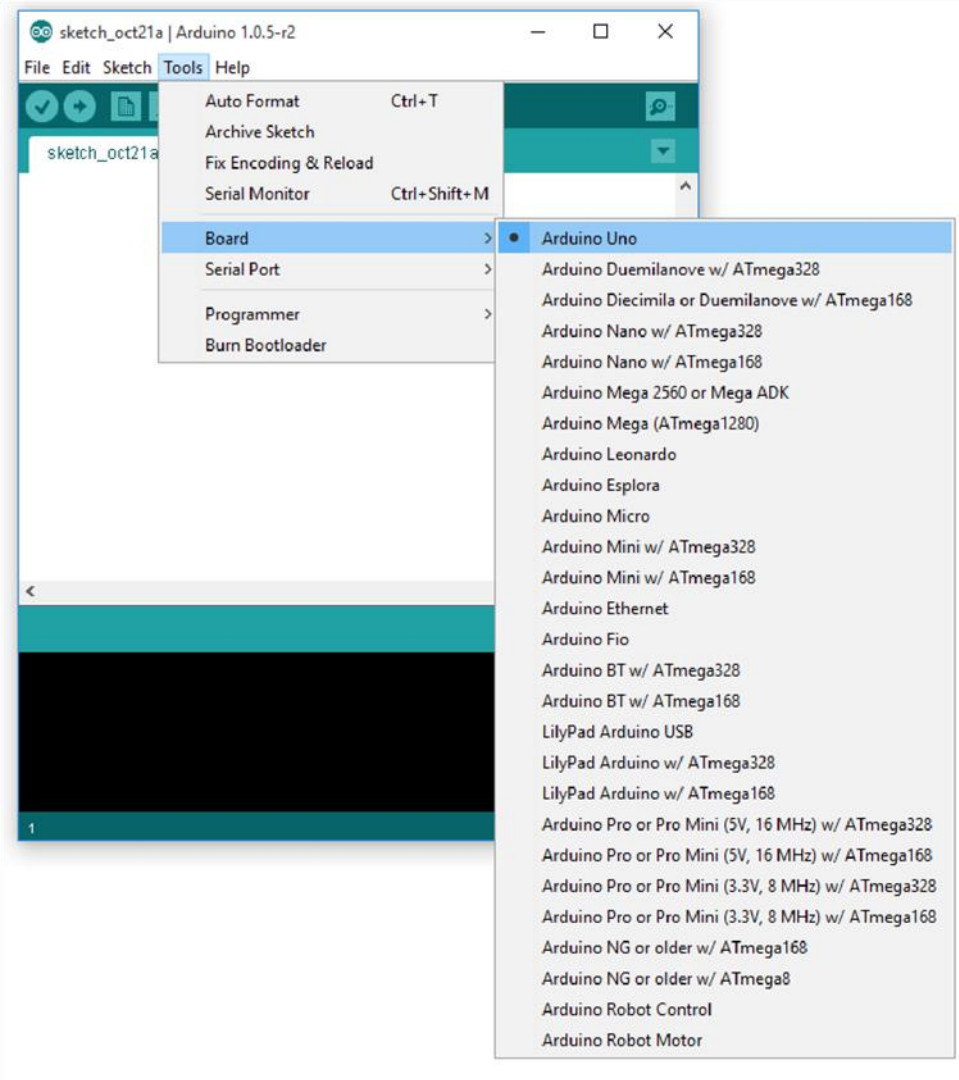
   `http://arduino.cc/en/guide/macOSX`
   `http://playground.arduino.cc/Linux/All`

You will be using the Arduino software on your machine to create and deploy programs. The software needs to be configured so that it knows the type of Arduino it is connected to and the serial connection it is using. To do this follow the procedure below.

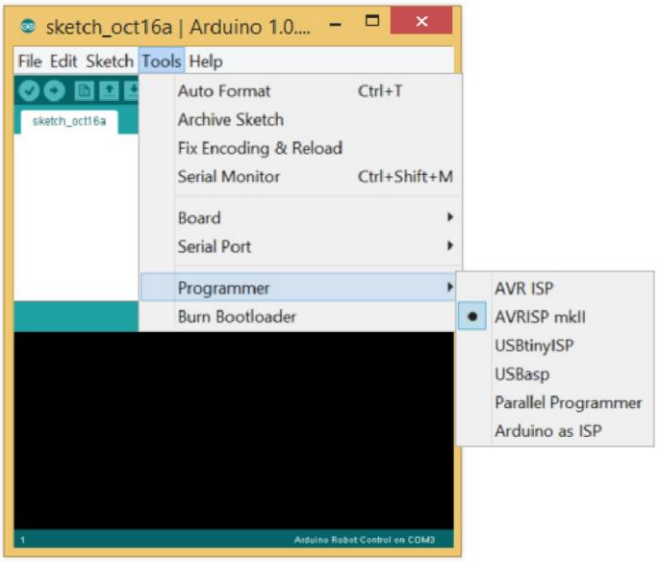
1. Open up the Arduino program editing software just type “Arduino” at the start button. You should see the following window



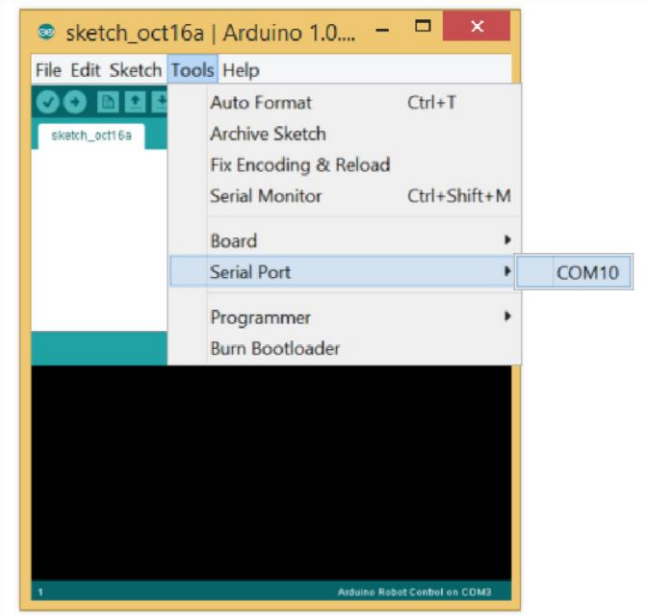
2. Deploy an empty program by opening the example program called BareMinimum. This can be found in the “01_Basics” folder accessible from the “File>Examples” menu.
3. Next tell the software which device we are using. Do this from the Tools menu as shown in the next image. Select the Arduino Uno as shown.



4. Next set the programmer that will be used, do this from the tools menu and select **AVRISP mkII** as shown below.



5. Finally select the serial port that the Arduino is connected to. Do this from the Tools menu. This will be different depending on the computer you use and the software will often pick this up automatically. However it can cause a problem particularly if you are using the port for connection to another program, for example Unity.



6. Test the program and configuration by uploading it to the Arduino. Do this by kicking the upload button at the top left of the screen. If successful you should see the message “Done Uploading”. There are usually LEDs on the Arduino board that light when it is communicating with the PC, these should flash during the upload process however the project doesn’t do anything so you shouldn’t see anything happen on the Arduino when it is complete.
7. Test the connection further by loading the “Blink” program from the samples folder. Deploy this to the Arduino and you should see one of the on board lights flashing on and off.

Creating a simple circuit

The LED connected to pin 13 on the Arduino is wired directly to the pin on the device circuit board. Now we are going to discover how this wiring works by connecting our own LED to another digital pin.

We are going to start by creating a simple circuit to light up an LED. The circuit is as follows:

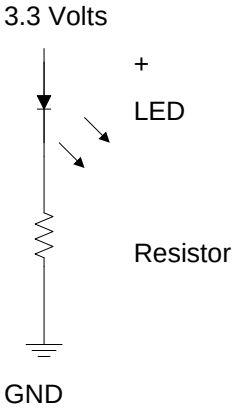


Figure 1: A very simple circuit

Computers work by storing patterns of 0 and 1 and then “twiddling” with them. In some computers the value of 1 is represented by a 5 volt signal, but in the Arduino this voltage is reduced to 3.3 volts to reduce power consumption. You might want to think about how this helps. The equation Power = Current x Voltage might be useful.

The current flows through the LED and causes it to light up. The resistor is there to reduce the flow of current so that that the LED receives just enough to light, but not so much that the LED overheats and fails. The 3.3 volt source is from the Arduino power supply. You might expect that the voltage you are going to use is 5 volts; after all, the Arduino is plugged into the USB port on your computer, and that produces 5 volts. However, it turns out that the actual Arduino computer runs on 3.3 volts to reduce the power consumption of the chip.

Breadboards and circuits

We are going to create our circuit using a breadboard. You can buy these in various shapes and sizes. They can be used to prototype a design and prove that it works, before you get the hardware actually manufactured. Alternatively, for very simple projects that you don’t want to mass produce, you can use them to produce the finished product. However it’s more common to transfer your circuit to Veroboard which is a more permanent soldered implementation.

Breadboards are great for building simple circuits. They provide sockets that you can push wires and component leads into to link them together. Figure 2 shows the breadboard that we are going to use (yours might be a different colour). If you wish

you can use the even larger one in the kit (although it is a bit hard to push the wires into), but the tiny one is a good start. The neat thing about a breadboard is that the sockets are connected underneath, so you can link things together without needing any (or at least many) wires.

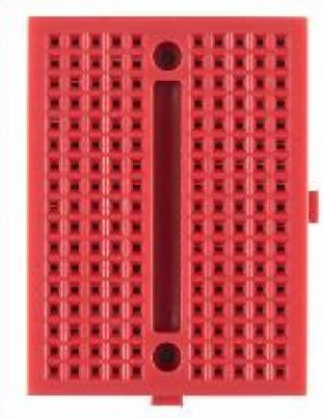


Figure 2: Our empty breadboard

If you look very carefully at the breadboard in Figure 2 you will find that actually it is made up of three parts. The top and bottom rows are separate. This means that the breadboard has a central area for your components and then two connectors along the top and the bottom for delivering power and providing a ground connection.

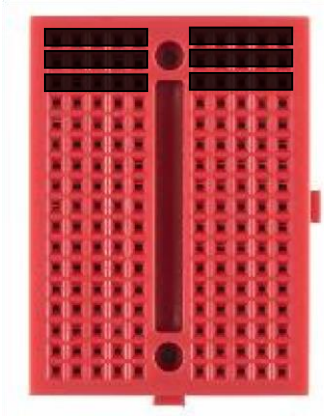


Figure 3: Breadboard connections

Figure 3 shows how the sockets are connected together on the breadboard that we are going to use.

I've only drawn in the top three rows, but the rest down the board are connected the same way. Note that there is no connection across the "river" in the middle, so we can put components across this bit and then use the sockets either side to connect to them.

Making an LED of Our Own

The first thing we are going to do does not involve any programming at all. We are simply going to make a circuit that will cause an LED to light up. We can start by considering how we are going to insert the components in the breadboard end. To do this we will need to use:

- The breadboard (obviously)
- The red LED
- A 220 Ω resistor (from the helpfully labelled pack)
- A red jumper wire
- A black jumper wire

We are going to implement the circuit that lights the LED. Then we are going to get the Arduino to control the LED.

Create a circuit using the arrangement show above. Make sure that the components all line up and push the LED into the two holes and make sure that the rows with the plugs on line up correctly.

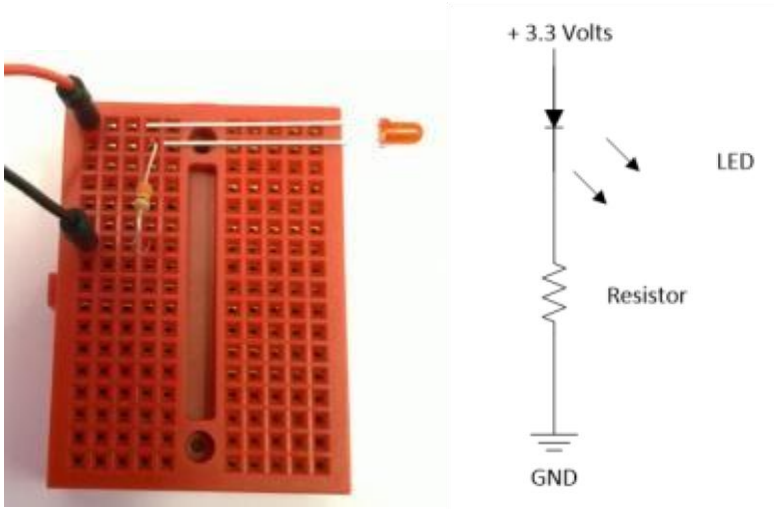


Figure 4: A circuit on a breadboard

Figure 4 shows how we take a circuit and implement it on a breadboard. The resistor makes the connection between the power socket and the LED bridges the gap between the two vertical rows. I've laid the LED on its side so you can see which socket the long side goes into.

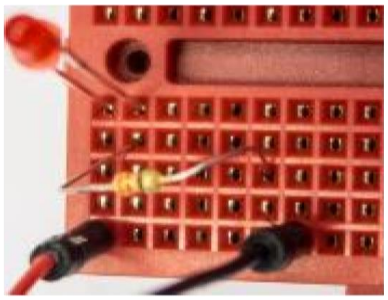


Figure 5: Something wrong her

Figure 5 shows a LED that is would not light up. Can you see what is wrong here?

Now we have connected the breadboard end we need to consider how to connect to the Arduino that is going to provide the power.

Figure 6: Connecting to the Arduino

Figure 6 shows the connections to be used. We are going to use the 3.3 volt and the ground sockets. The pins on the Arduino are all labelled with their function. However, the labels are printed on the board and the connectors are quite long, so make sure that you line up the label correctly.

Put the red connector into the socket labelled 3.3 volts and the black connector into a socket labelled GND. There are three of these on the board, you can use any one of them you prefer.

Once you have made the connections, you can plug the USB cable into your PC and power up the Arduino. At this point the power sockets on the device should become live and the LED should light up.

If this doesn’t happen make sure that you have lined up all the connections on the breadboard correctly and that the LED is plugged in the right way round.

We will use this technique to add other components to the circuit and connect the Arduino to them. Just remember how the sockets are connected. If it helps, you can think of the river that runs down the middle of the breadboard as being at the bottom of a valley, with the rows of sockets all connected together as they run down towards it.

Controlling a LED from your program

At this point in the exercise you should have already downloaded the Blink program into your device and it should be merrily flashing your LED. We are going to connect

our red LED to another pin on the Arduino and then change the program so that it controls that LED.

Figure 7: The Blink program

Figure 7 shows the Blink program inside the Arduino environment. You should have this up on your screen at the moment, and the Blink program should be running. Take a look at the statement:

int led = 13;

The program will look in this variable if it ever needs to know which of the digital ports is connected to the LED.

Change the value of led to 12 and upload your program. Note that the LED on the Arduino no longer flashes. The effect of changing the value 13 to 12 is that the digitl output 12 is used rather than 13 in the original program. As the LED is connected to 13 the LED runs fine. However there is nothing wrong with the program so it runs fine.

Note: One of the most frustrating things with electronic interfacing is that when a solution doesn't work you have to debug your code and your electronics.

Move the red cable connection from the 3.3 v pin across the Arduino to digital pin 12. You should see the LED start to flash.

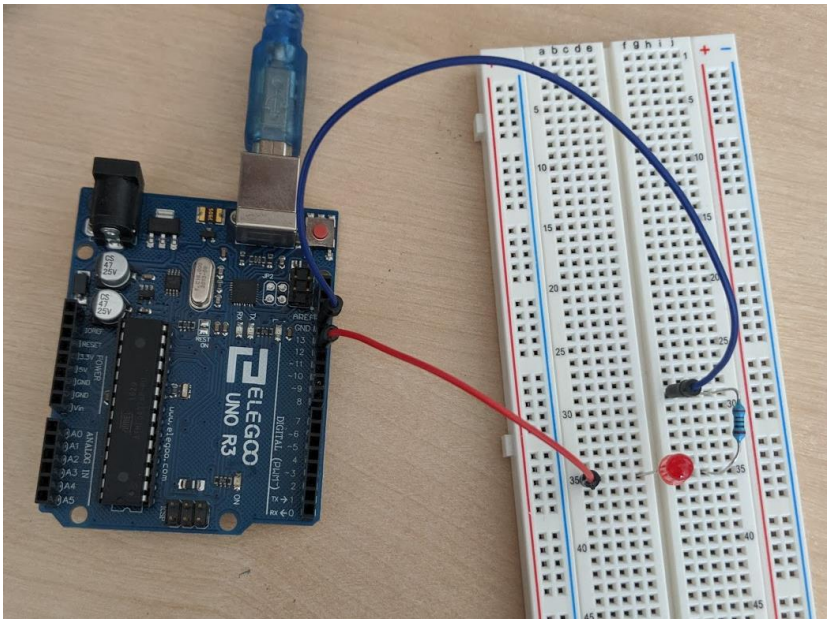


Figure 8: The complete connections

If your LED isn't flashing, check your wiring against the arrangement in Figure 8.

Working with the LED Flashing program

The program that is controlling the LED looks like this:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
  This example code is in the public domain.
 */
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 12;

// the setup routine runs once when you press reset:
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop()
{
digitalWrite(led, HIGH);          // turn the LED on (HIGH is the voltage level)
delay(1000);                      // wait for a second
digitalWrite(led, LOW);           // turn the LED off by making the voltage LOW
```

```
delay(1000);                    // wait for a second
}
```

The top part is just a comment that tells us about the program. It says what the program does. This is followed by the creation of our led variable. Note that the comments no longe agree with what the program does so we probably need to fix that.

The led setup is followed by the creation of a method called setup. A method is a set of C statements that have been given a name. The setup method is used when the hardware needs to be set up. This is just after the Arduino is switched on.

```
// the setup routine runs once when you press reset: void setup() {
  // initialize the digital pin as an output.   pinMode(led, OUTPUT);
}
```

This version of the setup method calls another method, pinMode, to tell the Arduino system that the led in is going to be an output. The pinMode method is given two pieces of information. The number of the pin that is being controlled (in this case the value in led) and the mode for that pin. The value OUTPUT is actually built into the Arduino software. There is a corresponding value which can be used to set a pin as an input. I will leave you to work out what this is.

The second method in our program is called loop. This is called repeatedly when the Arduino is switched on.

```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
 }
```

You can probably work out what the statements in the loop method do without much further explanation. The text at the right of each line, after the // characters, is comment text. The digitalWrite method writes a value to a particular pin and the delay method pauses for a particular number of thousandths of a second.

Change the delay value to 500 and upload your program. This should make the light flash at a different frequency.

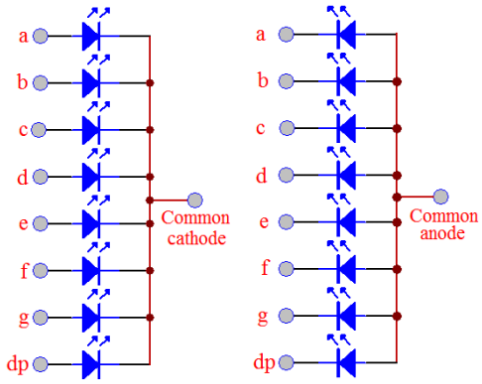
Task: Add a second LED (the yellow one) and connect it to pin 11. Change the variables so that you have a redLed variable with 12 in it and a yellowLed variable with 11 in it. Make both the LEDs flash together.

Remember that each LED that you add will need its own 220 Ω resistor and connection to ground. Remember also that you need to make sure the LED is connected the right way round, with the long lead going towards the positive signal

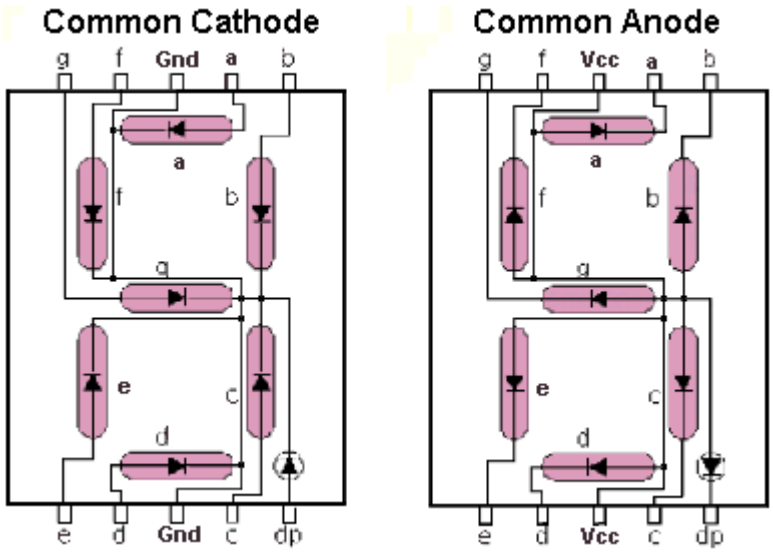
(the output pin). Change your program so that when one LED is lit the other is off, so that the light seems to “bounce” between them.

Interfacing a 7 Segment Display.

Seven segment displays are used mostly to display numbers. They are most often used in clocks and stopwatches. Individual even segment displays as shown below have 10 pins however they are not all standard. There are versions configured as common cathode and common anode. The difference is that the common cathode has all the ground ends of the LEDs connected together and you turn on different elements by connecting the other end to positive feed. The common cathode is the reverse, i.e. you connect the pin to 3.3 v via a resister and turn them on by connecting the other end to ground. This is shown in the diagram below.



The good news is that the pins for each segment are in the same position for both configuration. This is shown below.



The seven segment display supplied in the lab kits are 5161A5 these are common Cathode

Tasks.

1. Wire up a circuit capable of displaying numbers on the seven segment LED. Remember you need a bias resister in series with LEDs.
2. Write a program to control the elements on the LED.
3. Make your program count up to 9 at the rate of 1 per second.