CamJam EduKit 1 on BMC64

Mark Bush

Code and screen samples were typeset using the C64 Pro Mono font (available from style64.org).

Acknowledgements

CamJam EduKits

The CamJam EduKits have been created by the Cambridge Raspberry Jam people in partnership with The Pi Hut. They are very inexpensive kits which will help you get started on your journey of understanding how to use a Raspberry Pi to interact with the real world. The CamJam team have created very easy to follow worksheets which allow you to experiment with the components using Python. This booklet is based on those worksheets and allows you to perform the same activities using BMC64.

BMC64

BMC64 is an amazing 'bare metal' implementation of classic Commodore computers on the Raspberry Pi by Randy Rossi. This means that there is no operating system running on the machine—just the emulator. It can switch between many models such as C64, C128, and VIC20 among others.

Fritzing

Circuit diagrams and component images are produced using Fritzing.

Warning

When connecting components to your Raspberry Pi, always make sure to follow examples carefully. Never connect a pin providing power directly to a grounded pin—this will overload your Pi and likely burn it out. If you use GPIO devices for BMC64 (such as a PCB to connect to an original C64 keyboard, etc), you **must** disconnect them before trying the experiments in these worksheets or you could damage them and your Pi. Devices connected to USB ports are fine.

Saving Programs

If you don't already have a disk image on your SD card to save programs to, then you can easily create one. Go into the settings menu (F12), select the 'Drives' option and then 'Create empty Disk'. For now, create a 'D64' disk and call it something like "CamJam-EduKit1". Now select 'Drive 8' and then 'Attach Disk' and select the disk you just created.

To save a program, use the "SAVE" command:

```
SAVE "1 HELLO WORLD",8
```

The ",8" part tells the command that you are referring to the disk you selected above. You should always make sure that it saved by verifying it:

```
VERIFY "1 HELLO WORLD",8
```

If you get an error, try saving again. If the name is already in use, then you need to use a special form of the same command to ensure the file gets overwritten (if you really mean to replace the program on disk):

```
SAVE "@:1 HELLO WORLD",8
```

Without the "@:" part, the computer will refuse to replace a file that already exists, however it won't produce an error. This is why you should always verify that your save was successful!

Contents

1 Introduction 2

2 LEDs 5

3 Blinking LEDs 11

4 User Input 13

5 Button 15

6 Buzzer 19

7 Traffic Lights 23

8 Games 26

1 Introduction

Project	Setting up your Raspberry Pi.
Description	Set up your Raspberry Pi and run your first BASIC program to print "Hello World" to the screen. You will not be connecting any of the contents of the CamJam EduKit to the Raspberry Pi for this short exercise.

Notes and Assumptions

During these experiments, we will be using the Raspberry Pi GPIO pins as both inputs and outputs. It is very important that you do not have any other equipment connected to the GPIO pins as this could result in unexpected behaviour. In the worst case, it could damage your Raspberry Pi!

We will be using the BMC64 emulator of the Commodore 64. This is a 'bare metal' application, which means that it runs on the Raspberry Pi as a single application whithout any operating system.

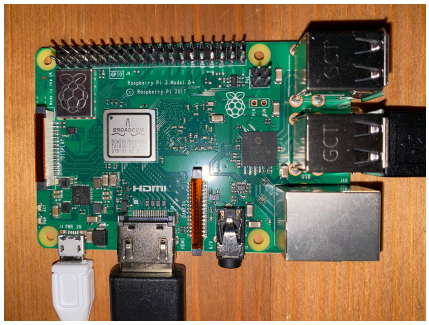
The Commodore 64 had a special port to allow people to connect electronics called the **User Port** and this has been made available in BMC64. It connects to some of the GPIO pins. The user port is only available for Raspberry Pis with 40 GPIO pins, however it is not currently available on the Raspberry Pi 4.

Equipment Required

- A Raspberry Pi (model 3 B+ recommended).
- An SD card with BMC64 installed. Instructions for setting up BMC64 can be found on the BMC64 website.
- Monitor and cable to connect to the HDMI output of your Pi.
- A keyboard.
- A Raspberry Pi power supply.
- The EduKit 1 kit, available from The Pi Hut.

Setting Up Your Raspberry Pi

Find your Raspberry Pi.



- Plug in the micro SD card.
- Plug the HDMI/video cable into the Raspberry Pi and the monitor.
- Plug the keyboard into a USB port.
- Ensure nothing is connected to any GPIO pins!
- Plug in the power supply.

When everything is connected, it should look like this. It is a good idea to ensure you are always using the latest version of BMC64. This will mostly just require copying kernel images from the distribution ZIP file onto your card. Sometimes, other configuration files are updated. If you have edited them, check what is new and just copy those parts into your edited version.

Once you have supplied power to your Pi, you should see the normal C64 startup screen as shown:

Go into the settings menu ('F12'), select the "Prefs" option, then the GPIO option. Use the right and left cursor keys to select option 4 ("Userport and Joys"). This is required to activate the connection between the BMC64 user port and the Raspberry Pi GPIO pins.

How to Type In Code

Program listings will be shown in one of two possible ways. Both will use the same font that BMC64 uses. One will show exactly how something will appear on the screen, like this:

This will let you see if what you have typed is exactly what it should be. This form of listing doesn't make it very easy to know what to type to get the special characters or how many spaces there might be in a particular string. For example, in the above line of code, it is not obvious that you need to press the 'CLR' key (usually SHIFT+'Home' on a USB keyboard) to get the reverse video heart symbol or why it is there at all.

To help with typing in special symbols, most listings will use descriptions for special keys, as shown here:

```
PRINT "{CLR}HELLO, WORLD"
```

When you need to press a special key, the name of the key will be shown in curly brackets, as above. Some of the special keys just require you to press the relevant key on your keyboard (such as the space bar for SPC, the 'Home' and 'Delete' keys for HOME and DEL, etc.). Some will require you to hold down another key at the same time (for example, CLR is usually typed by holding SHIFT and hitting the 'Home' key, F1–F8 are typed as they are on some keyboards and sometimes require that you hold down the 'function' key at the same time). The colour keys are obtained by holding down either the 'CTRL' key or the 'Commodore' key (usually the 'windows' key marked '⊞' or 'cmd' key marked '⌘' on a USB keyboard). Table 1 shows how to obtain each of the special characters. When you enter them in strings, they will appear as the characters in the 'Show' column.

Many of the keys on your keyboard will produce graphics characters when pressed while holding either SHIFT or Commodore keys. To indicate this, you'll see an underline when you need the shifted version (like 'N') and double angle brackets when you need to use the Commodore key (like '«N»'). When you need to enter more than one of the same character in a row, you'll see this in curly brackets with the number of times you need to enter the character. If a program line uses two screen lines, just keep typing—don't press RETURN in the middle. For example, a listing might be given as follows:

```
10 PRINT"{CLR}{ORG}{4 *} THE CARD SUITS ARE:
   ASZX"
```

However it will appear on your screen as:

Writing Code

You are now going to create your first small piece of BASIC code that will simply print "HELLO WORLD" to the screen. Type in the code exactly as shown:

```
10 REM PRINT HELLO WORLD!
20 PRINT "HELLO WORLD!"
```

Everything on the same line after 'REM' is a comment and will be ignored by BASIC. Save the program as "1 HELLO WORLD".

Running the Code

To run your code, type 'RUN' and hit the 'ENTER' or 'RETURN' key. You will see "HELLO WORLD!" printed on the screen:

Table 1: How to produce all the 'special' characters.

Ref	Meaning	Type	Show	Ref	Meaning	Type	Show
BLK	Black	CTRL-1		ORG	Orange	-1	
WHT	White	CTRL-2		BRN	Brown	-2	
RED	Red	CTRL-3		LRD	Light red	-3	
CYN	Cyan	CTRL-4		DGY	Dark grey	-4	
PUR	Purple	CTRL-5		MGY	Medium grey	-5	
GRN	Green	CTRL-6		LGN	Light green	-6	
BLU	Blue	CTRL-7		LBL	Light blue	-7	
YEL	Yellow	CTRL-8		LGY	Light grey	-8	
RIGHT	Cursor right	Cursor right		F1	Fn key 1	F1	
LEFT	Cursor left	Cursor left		F2	Fn key 2	F2	
DOWN	Cursor down	Cursor down		F3	Fn key 3	F3	
UP	Cursor up	Cursor up		F4	Fn key 4	F4	
RVS	Reverse video	CTRL-9		F5	Fn key 5	F5	
OFF	Normal video	CTRL-0		F6	Fn key 6	F6	
HOME	Go to top left	Home		F7	Fn key 7	F7	
CLR	Clear screen	SHIFT-Home		F8	Fn key 8	F8	
DEL	Delete	Delete		SPC	Space	Space	
INST	Insert	SHIFT-Delete					

2 LEDs

Project	Controlling LEDs with BASIC
Description	In this project, you will learn how to connect and control LEDs (Light Emitting Diode) with the Raspberry Pi.

Equipment Required

You will need:

- Your Raspberry Pi
- 400 Point Breadboard
- 1 x Red LED
- 1 x Yellow LED
- 1 x Green LED
- 3 x 330Ω Resistors
- 4 x M/F jumper wires

The Parts

In this first circuit, you will be connecting three LEDs to the GPIO header of your Raspberry Pi and using BASIC to turn the LEDs on and off.

It is important that you read this section, as you need to understand the Raspberry Pi GPIO pins, how the holes in the breadboard are connected together, and which leg of the LED is which.

Before you build the circuit, let us look at the parts you are going to use.

Raspberry Pi GPIO Pins

First, lets look at the Raspberry Pi's '**GPIO**' pins. GPIO stands for **General Putpose Input Output**. It is a way the Raspberry Pi can control and monitor the outside world when connected to electronic circuits. The Pi is able to control LEDs, turning them on and off, or motors, or many other things. It is also able to detect whether a switch has been pressed, or what the temperature is, or whether there is light. With this CamJam EduKit you will learn to control LEDs and a buzzer, and detect when a button has been pressed.

The disgram on the left shows the pin layout for all the Raspberry Pi models built for the last few years; they have 40 GPIO pins. The original Raspberry Pi models A and B only had 26 pins and will not work for these worksheets.

Some pins have different functions. There are pins that provide power at 5 volts and 3.3 volts, ground pins (0 volts), input/output pins and some pins that interface to external circuits in more complex ways (which are not currently supported by BMC64). You are going to use 8 of the GPIO pins and the ground pins in these worksheets. Note that all of the pins marked as ground (GND) are equivalent.

In BMC64, we access the GPIO pins through the User Port. For these worksheets, we will assume that you are operating in C64 mode. In this case, the user port is represented by memory location 56577. Each memory location can hold a number from 0 to 255. This is because the computer uses 8 **bits** to hold each value and each bit can be 0 or 1. Table 2 shows the numerical value of each bit and which GPIO pin it accesses.

Table 2: GPIO pins used by the user port.

Bit	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
GPIO Pin	21	20	26	16	8	25	24	4

The bits are numbered from the right (lowest valued) position, starting at bit number 0. To specify which bits we want turned on, we add the value for those bits. For example, if we want to access bits 1 and 0, we see that they have values 2 and 1. Now, $2 + 1 = 3$ which means we would need a value of 3 to represent those two bits.

Each pin in the user port can be either an input or an output. Location 56579 holds this direction information and is known as the **data direction register** or **DDR**. In the DDR, each bit that is 0 means the corresponding bit in the user port is an input and each bit that is 1 means the user port bit is an output. The default value in the DDR is 0, so all of the pins default to inputs. If the DDR had a value of 255, then all of the pins would be outputs and a value of 15 would mean the first four pins would be inputs and the last four pins would be outputs.

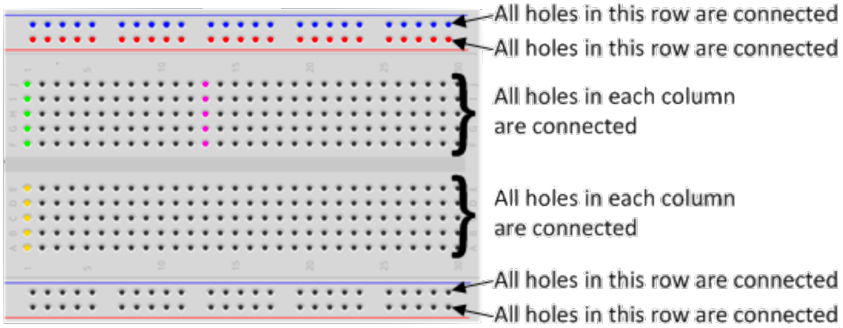
The BASIC commands you will use to read and write these values are **PEEK** and **POKE**. Since you will be using the user port and DDR locations a lot, it will be best to put them into variables. Let's use **UP** and **DDR** for these purposes. In this worksheet, you will be performing output, so you will need to be POKEing values into UP and DDR.

To set all of the user port pins to inputs, you will use **POKE DDR,0** and to set them all as outputs, you use **POKE DDR,255**. When pins are set as inputs, you can read the state of them using **PEEK(UP)** and if you want to set pins to be on, then you will use **POKE UP,X** (where X will represent which pins are on and which are off).

In this way, using the user port, we will be able to access and control up to eight devices at the same time.

The Breadboard

The breadboard is a way of connecting electronic components to each other without having to solder them together. They are often used to test a circuit design before creating a Printed Circuit Board (PCB).

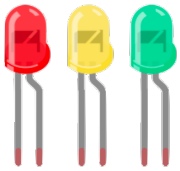


The holes on the breadboard are connected in a pattern.

With the breadboard in the CamJam EduKit, the top row of holes are all connected together—marked with blue dots. And so are the second row of holes—marked with red dots. The same goes for the two rows of holes at the bottom of the breadboard.

In the middle, the columns of holes are connected together with a break in the middle. So, for example, all the green holes marked are connected together, but they are not connected to the yellow holes, nor the purple ones. Therefore, any wire you poke into the green holes will be connected to other wires poked into the other green holes.

The LEDs

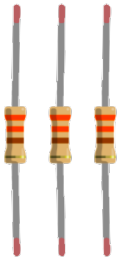


Three LEDs are supplied in the EduKit—one red, one yellow, and one green. LED stands for Light Emitting Diode. An LED glows when electricity is passed through it.

When you pick up an LED, you will notice that one leg is longer that the other. The longer leg (known as the 'anode') is always connected to the positive supply of the power supply. The shorter leg (known as the 'cathode') is connected to the negative side of the power supply (known as 'ground').

LEDs will only work if power is supplied the correct way round (i.e. if the 'polarity' is correct). You will not break the LEDs if you connect them the wrong way round—they will just not light. If you find that they do not light in your circuit, if may be because they have been connected the wrong way round.

The Resistors



Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of 'current' that is allowed to flow. The measure of resistance is called the Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

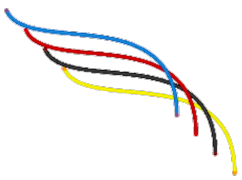
The EduKit is supplied with two sets of resistors. There are three 330Ω resistors and one 4.7kΩ (or 4700Ω) resistor. In the LED circuit, you will be using the three 330Ω resistors. You can identify the 330Ω resistors by the colour bands along the body. The colour coding will depend on how many bands there are on the resistors supplied:

- If there are four colour bands, they will be Orange, Orange, Brown, and then Gold.
- If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.

You have to use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi can only supply a small current on the GPIO pins. The LEDs will want to draw more and, if allowed to, they will burn out the Raspberry Pi. Therefore, putting resistors in the circuit will ensure that only this small current will flow and the Pi will not be damaged.

It does not matter which way round you connect the resistors. Current can flow in both directions through them.

The Jumper Wires



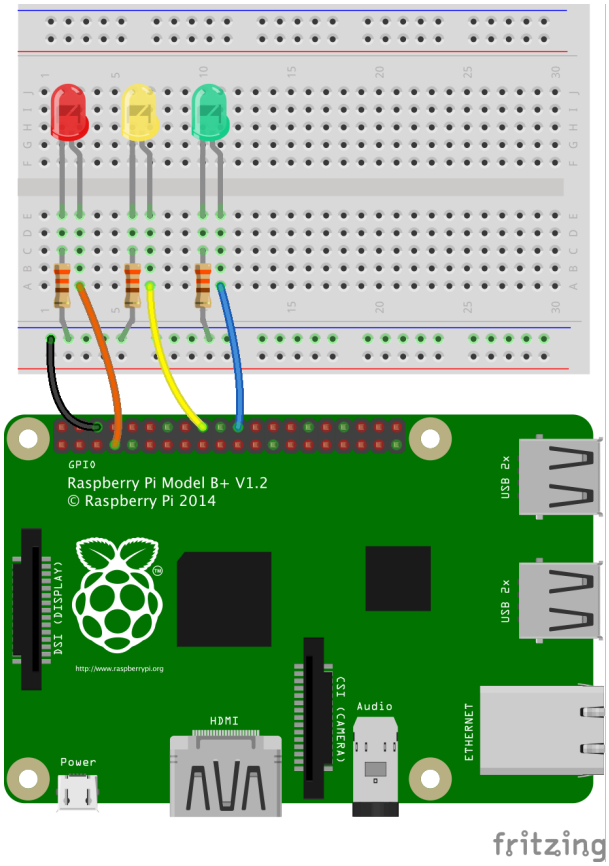
Jumper wires are used on breadboards to 'jump' from one connection to another. The ones you will be using in this circuit have different connectors on each end. The end with the 'pin' will go into the breadboard, and is known as the 'male' end. The end with the piece of plastic with a hole in it will go onto the Raspberry Pi's GPIO pins. This is the 'female' end.

The jumper wires supplied in the EduKit will vary in colour and are unlikely to match the colours used in the diagrams.

Building the Circuit

While you can build the circuit with the Pi turned on, it is best to turn it off at this stage.

You will be using one of the 'ground' (GND) pins to act like the 'negative' or 0 volt end of a battery.



The 'positive' ends of the battery will be provided by three of the other GPIO pins, one for each of the three LEDs. You will be using the pins marked 4, 24, and 25 for the Red, Yellow, and Green LEDs respectively.

When they are 'taken high', which means they output 3.3 volts, the LEDs will light.

Now take a look at the circuit diagram on the left.

The power for each LED will be provided by the Pi, from GPIO pins 4, 24, and 25. You can control them from BASIC, meaning you can make the GPIO pins supply either 0 volts (off) or 3.3 volts (on).

There are, in fact, three separate circuits in the diagram. Each one consists of the power supply (the Pi), an LED that lights when power is applied, and a resistor to limit the current that can flow through the circuit.

Each circuit is going to share a 'common ground rail'. In other words, you will be connecting all of the circuits to the same 'ground' (0 volts) pin of the Raspberry Pi. You are going to use the second row up from the bottom of the breadboard. Remember that the holes on the two top and two bottom rows are connected together? So, connect one of the jumper wires from the third pin from the left on the top row of the Pi to the second row up of the breadboard, as shown in the diagram (the black wire).

Next, push three LEDs legs into the breadboard, with the long leg on the right as shown in the circuit diagram.

Then connect the three 330Ω resistors between the 'common ground rail' and the left leg of the LEDs. You will need to bend the legs of each of the resistors to fit, but please make sure that the wires of each leg do not touch one another.

Lastly, using three jumper wires, complete the circuit by connecting pins 4, 24, and 25 to the right-hand leg of each LED. These are shown here with orange, yellow, and blue wires.

You are now ready to write some code to switch the LEDs on.

Code

Follow the instructions in Section 1 to turn on your Pi. Type in the following code:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS
50 R=1:Y=2:G=4:A=R+Y+G
60 REM SET PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 PRINT "LEDS ON"
110 POKE UP,A
120 PRINT "WAIT FOR ONE SECOND"
130 FOR I=1 TO 720:NEXT I
```

```
140 PRINT "LEDS OFF"
150 POKE UP,0
```

Once you have typed all the code and checked it, save it to disk as "2 LED".
So, what is happening in the code? Let's go through it a section at a time:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
```

This introduces the variables UP, to refer to the user port, and DDR, to refer to the data direction register. This is so that we don't have to type the locations multiple times. It will also mean that we are less likely to make mistakes. If you decide to try these worksheets using a different emulator model (eg. VIC20), then it will likely have different locations for these two. Putting them into variables means that you will have to make fewer changes to be able to run the code.

```
40 REM PIN VALUES FOR LEDS
50 R=1:Y=2:G=4:A=R+Y+G
```

This is telling BASIC that you are going to use pins 0, 1, and 2 (with values 1, 2, and 4) of the user port. To reference all of them together, remember that we just add them.

```
60 REM SET PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
```

We have connected the three LEDs to pins 0, 1, and 2 of the user port so we need to tell BASIC that we are going to use them for outputting information, which means you are going to be able to turn the pins 'on' and 'off'. We have also set all pins on the user port to 0 (off) so that the LEDs will start in the off state.

```
100 PRINT "LEDS ON"
```

This line prints some information to the screen so you can see what the program is trying to do.

```
110 POKE UP,A
```

This turns the three LEDs 'on'. What this actually means is that these three pins are made to provide power of 3.3 volts to the three GPIO pins.

```
120 PRINT "WAIT FOR ONE SECOND"
130 FOR I=1 TO 720:NEXT I
```

These two lines print a message to the screen, followed by an empty loop which will pause BASIC for about 1 second.

```
140 PRINT "LEDS OFF"
150 POKE UP,0
```

This prints out a message again, then turns all three LEDs off.

Running the Code

You are now ready to run the code. Run it by typing in **RUN** followed by the **ENTER** or **RETURN** key. You should see your LEDs light for one second, then turn off again.

If you find that the code does not run correctly there may be an error in the code you have typed. You should check and re-edit the code, save again and re-run it. Also check that you have configured the GPIO option to enable the user port in the settings.

> **Note**
> Do not disassemble this circuit as it will be used in the following worksheets.

3 Blinking LEDs

Project	Making LEDs blink with BASIC.
Description	In this project, you will learn how to make LEDs blink.

This worksheet uses the same LED circuit that was built in Section 2.

Equipment Required

The circuit built in CamJam EduKit Section 2.

Code—Blink Twice

Type in the following code:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS
50 R=1:Y=2:G=4:A=R+Y+G
60 REM SET PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 REM TURN LEDS ON
110 POKE UP,A
120 FOR I=1 TO 720:NEXT I:REM PAUSE FOR 1 SECOND
130 REM TURN LEDS OFF
140 POKE UP,0
150 FOR I=1 TO 720:NEXT I:REM PAUSE FOR 1 SECOND
160 REM TURN LEDS ON
170 POKE UP,A
180 FOR I=1 TO 720:NEXT I:REM PAUSE FOR 1 SECOND
190 REM TURN LEDS OFF
200 POKE UP,0
```

Once complete, save the file as “3 BLINK”.

Running the Code

Run the code. You will see the three LEDs turn on then turn off twice, with one second in between each change.

If your code does not run and an error is reported, edit the code again.

Code—Blink Forever

Our next pieve of code will flash the lights on and off forever (or until you press “RUN/STOP”).

Type in the following code:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS
50 R=1:Y=2:G=4:A=R+Y+G
60 REM SET PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 POKE UP,A:REM TURN LEDS ON
110 FOR I=1 TO 720:NEXT I:REM PAUSE FOR 1 SECOND
```

```
120 POKE UP,0:REM TURN LEDS OFF
130 FOR I=1 TO 720:NEXT I:REM PAUSE FOR 1 SECOND
140 GOTO 100:REM LOOP FOREVER
```

Save the program as "3 BLINK FOREVER".

Running the Code

Run the code. You will see the three LEDs turn on and off forever, or until you press "RUN/STOP".

If the LEDs are still on when you stop the program, just execute the following command:

```
POKE UP,0
```

If there is an error in the code, change the code and re-save it. Re-run to check that you have corrected the error.

4 User Input

Project	Interact with the user and input your choice.
Description	In this project, you will control the red, yellow, or green LED depending on your choice.

Equipment Required

The circuit built in CamJam EduKit Section 2.

Code

You are going to use the same circuit again, but this time you are going to control the LEDs with user input. This worksheet will introduce user input as well as using variables to store information that will be used in later code.

Explanations have been placed within the code. These are called 'comments' and in BASIC they are the text following 'REM'. Nothing after REM will be run, and can be left out if you want, although best practice is to use comments to remind you what you intended your code to do.

Type in the following:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS
50 R=1:Y=2:G=4:A=R+Y+G
60 REM SET PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 PRINT CHR$(147);:REM CLEARS THE SCREEN
110 REM ASK THE USER WHICH COLOUR LED TO BLINK
120 PRINT "WHICH LED WOULD YOU LIKE TO BLINK?"
130 PRINT "1: RED?"
140 PRINT "2: YELLOW?"
150 PRINT "3: GREEN?"
160 INPUT "WHAT IS YOUR OPTION";L
170 REM ASK THE USER HOW MANY TIMES THEY WANT THE
    LED TO BLINK
180 INPUT "BLINK HOW MANY TIMES";C
190 REM SET THE VARIABLE P TO BE THE LED PIN
    VALUE
200 IF L=1 THEN PRINT "YOU PICKED THE RED LED"
    :P=R
210 IF L=2 THEN PRINT "YOU PICKED THE YELLOW LED"
    :P=Y
220 IF L=3 THEN PRINT "YOU PICKED THE GREEN LED"
    :P=G
230 REM IF WE DON'T HAVE A VALID CHOICE, END
240 IF P=0 THEN END
250 REM REPEAT C TIMES
260 FOR I=1 TO C
270 POKE UP,P:REM TURN THE CHOSEN LED ON
280 FOR D=1 TO 720:NEXT D:REM SLEEP 1 SECOND
290 POKE UP,0:REM TURN THE CHOSEN LED OFF
300 FOR D=1 TO 1440:NEXT D:REM SLEEP 2 SECONDS
310 NEXT I
```

Save the program as "4 USER INPUT".

Running the Code

Run the code. The screen will clear, and you will be prompted for which LED you want to turn on or off. Enter 1, 2, or 3. You will then be prompted for how many times you want the LED to flash. The LED you chose will then flash the number of times you requested.

Note

Do not disassemble the circuit as it will be used in the following worksheets.

5 Button

Project	Push button for physical input
Description	In this project, you will learn how to wire and program a push button for physical input.

Equipment Required

The circuit build in Section 2 plus the following:

- push button
- 1 x M/F jumper wire
- 1 x M/M jumper wire

Additional Parts

You will be adding a switch to the LED circuit that you made in Section 2. Here are the additional components. You may skip this section if you already know about these components.

Push Button

A push button will complete a circuit when the button is pressed. What this means is that a current will not flow across the button until it is pressed. When it is released, the circuit will be 'broken'.

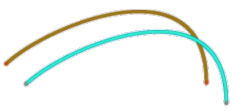
Normally, if you try to detect the state of an input pin when there is no current, it will randomly be either on or off due to interference of other components. BMC64 solves this by internally 'pulling up' the input to 3.3 volts through a large resistor. When read, the input pin will be seen as 'on'. The resistor ensures that the current is very low so that your Pi is not damaged!

BMC64 handles this automatically for you whenever a pin is specified as an input. This means your circuit usually only needs to be connected to ground. Input pins are 'pulled high' internally when there is no external connection and outputs are set high internally when you set the user port to 1.

This is also why your LEDs may be on when you first connect them. The user port is in input mode and the pins are 'high' when the computer is turned on. They will only glow faintly because of the large resistor. This is not a problem. This is also why the programs in these worksheets start by turning all the pins off in the user port, once the output pins have been specified in the data direction register.

When you press the switch, another circuit is made which will make the current 'flow to ground', which means that the input pin will be seen as being 'off'. This is how you will detect the switch.

Jumpers



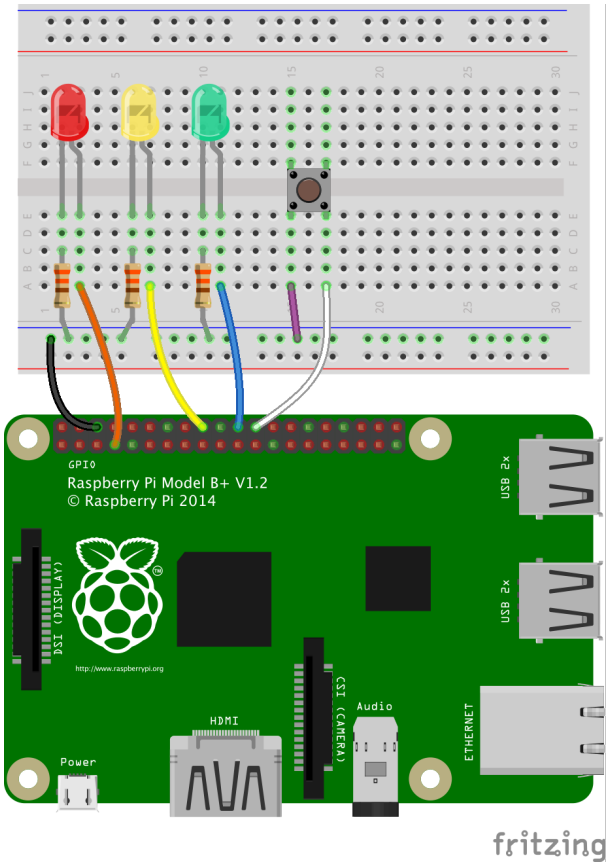
You will be adding one more Male/Female jumper wire and one Male/Male (M/M) jumper, which has a pin on each end. The M/M jumper will be used for joining two rows or columns on the breadboard together.

The jumper wires supplied in the EduKit will vary in colour and are unlikely to match the colours used in the diagrams.

Building the Circuit

Before you connect additional components to your circuit, you should turn off your Pi. Leave the LED circuit from the previous worksheets in place.

Add the button to the breadboard with the pins straddling the split in the middle of the board. It will only fit one way round. Push down hard to ensure that the pins are inserted fully into the breadboard.



Connect one of the jumpers (shown in white here) from one side of the button to the input pin you are going to use on the Raspberry Pi. You are using pin 8.

Remember that GPIO 8 is bit 3 of the user port and has a value of 8. We will need this value later to be able to read the input state of the switch.

Finally, connect the 'ground rail' to the other side of the switch, shown here by the purple wire.

So, how does the switch work? When the switch is not being pressed, the pin will be internally 'pulled high' to 3.3v and read as 'on'.

When the switch is pressed, the current takes another route. It goes through the switch and back to ground (0v). This splits enough of the current off that the voltage across the white wire is much lower and no longer high enough for the input GPIO pin to be seen as 'on'.

To be able to tell when the button has been pressed, we need to be able to isolate the bit of the user port we are interested in. This is because when we read the user port, we will read back any output bits that are set (if we have turned any LEDs on) and we will also read 1 for each input bit that is not grounded.

We can use the AND operator to isolate an individual bit from a value. If we take the user port value and AND with the bit value we are interested in, then we will extract just that bit. The result will be 0 if there is no input (the pin is low, which means the button is pressed) or the bit value if the pin is high (which means the button is not pressed).

Code

Type in the following code:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS AND BUTTON
50 R=1:Y=2:G=4:B=8:A=R+Y+G
60 REM SET ONLY PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
```

```
100 PRINT "--------------"
110 PRINT "BUTTON + GPIO"
120 PRINT "--------------"
130 P=PEEK(UP) AND B:REM ONLY GET BUTTON VALUE
140 REM IF BUTTON IS PRESSED THEN P WILL BE 0
150 IF P=0 THEN GOSUB 200
160 IF P<>0 THEN GOSUB 300
170 GOTO 130
200 PRINT "BUTTON PRESSED"
210 PRINT P
220 FOR D=1 TO 720:NEXT D:REM SLEEP 1 SECOND
230 RETURN
300 PRINT CHR$(147);:REM CLEAR THE SCREEN
310 PRINT "WAITING FOR YOU TO PRESS THE BUTTON"
320 FOR D=1 TO 360:NEXT D:REM SLEEP 0.5 SECONDS
330 RETURN
```

Save the program as "5 BLINK".

Concepts

What is happening in the code? Let us go through some of the important concepts before looking at the code:

- A 'variable' is a name that contains a value. That value can be changed within your code at any time. It is often easier to use a variable to contain a number because it is easier to remember the name.
- A "GOSUB" statement tells BASIC to go and run some code from somewhere else until a "RETURN" statement is reached. When execution reaches "RETURN", BASIC will go back to the statement after the original "GOSUB".

Explanation of the Code

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS AND BUTTON
50 R=1:Y=2:G=4:B=8:A=R+Y+G
60 REM SET ONLY PINS 0,1,2 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
```

This is the same setup code we have used before except we have set a 'variable' B to hold the value of the pin that we are using for input from the button.

```
130 P=PEEK(UP) AND B:REM ONLY GET BUTTON VALUE
140 REM IF BUTTON IS PRESSED THEN P WILL BE 0
```

'PEEK' will read the value of the entire user port. Using AND, we can pick out just the bit representing the button input. When the pin detects a voltage nearing 3.3v, it is read as 'on' and this bit will be 1. Since this bit has a 'value' of 8, the result will be 8. If the voltage nears 0v, then it is 'off' and the result will be 0.

```
150 IF P=0 THEN GOSUB 200
```

The 'IF' statement checks the value of the pin to see if it is 0 (so the button is being pressed). If so, then the code in lines 200 to 230 will execute.

```
160 IF P<>0 THEN GOSUB 300
```

This 'IF' statement checks the value of the pin to see if it is **not** 0 (so the button is not being pressed). If so, then the code in lines 300 to 330 will execute.

```
170 GOTO 130
```

Once the status of the button has been printed, we go back and repeat the process again. This 'loop' will run forever, or until 'RUN/STOP' is pressed.

Running the Code

Run the code. The program will wait for the button to be pressed and report the status to the screen. This will continue until you press 'RUN/STOP'.

6 Buzzer

Project	Morse code SOS using a Buzzer
Description	In this project, you will learn how to wire and program a buzzer, and use it to produce Morse code. You will be using 'subroutines'.

Equipment Required

The cuircuit built in Sectino 5, plus the following:

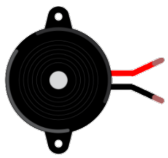
- Buzzer
- 1 x M/F jumper wire
- 1 x M/M jumper wire

Additional Parts

You will be adding a buzzer to the LED and switch circuit that you made in Section 5. Let us look at the additional component.

Do not skip this section, as you will need to know how to connect the buzzer.

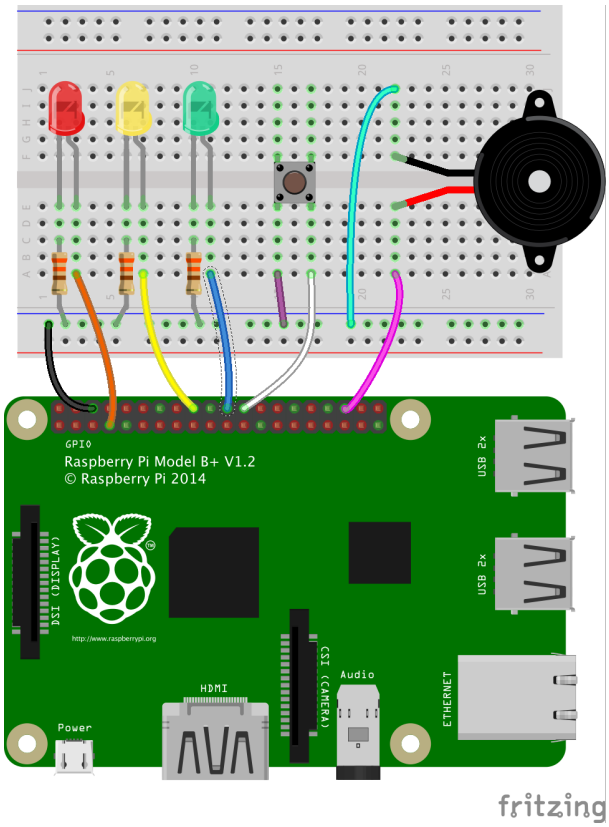
Buzzer



The buzzer supplied in the EduKit is an 'active' buzzer, which means that it only needs an electric current to make a noise. In this case, you are using the Raspberry Pi to supply that current.

The buzzer has positive and negative legs. The longer leg is positive (shown in red in the diagram), the shorter leg is negative (shown in black on the diagram).

Building the Circuit



Before you connect additional components to your circuit, you should turn off your Pi.

Leave the LED and switch circuit from Section 5 in place.

Place the buzzer on the breadboard straddling the middle divide. The longer leg should be connected via a jumper wire to GPIO 16.

The other leg should be connected to the ground rail.

GPIO 16 will be an output pin and, when it is set on, the buzzer will sound.

Remember that we will need to ensure that this pin can be used as an output. This pin is bit 4 in the user port and has a value of 16. You will need to set this bit in the DDR to 1 to allow output.

Concepts

You are going to be using 'subroutines' in the code below. These are pieces of code that you may want to run more than once, but by using subroutines, you only have to write them once. You then 'call' that subroutine from within your code each time you want to run it.

A subroutine is any section of code which ends with a 'RETURN' statement:

```
200 PRINT "HELLO WORLD!"
210 RETURN
```

To use this subroutine, you must 'call' it by using a 'GOSUB' statement with the line number that starts the subroutine:

```
50 GOSUB 200
```

Now, every time BASIC sees 'GOSUB 200' in your code, it will print "HELLO WORLD!".

Code

Type in the following code below exactly as seen:

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR BUZZER
50 Z=16
60 REM SET BUZZER PIN FOR OUTPUT
70 POKE DDR,Z
```

```
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 PRINT CHR$(147);:REM CLEARS THE SCREEN
110 PRINT "MORSE CODE"
120 REM PROMPT THE USER FOR INPUT
130 INPUT "HOW MANY TIMES FOR SOS TO LOOP";C
140 FOR L=1 TO C
150 GOSUB 1500:REM S
160 GOSUB 1300:REM LETTER SPACE
170 GOSUB 1600:REM O
180 GOSUB 1300:REM LETTER SPACE
190 GOSUB 1500:REM S
200 GOSUB 1400:REM WORD SPACE
210 NEXT L
220 END
1000 REM A DELAY OF D TENTHS OF A SECOND
1010 T=D*72
1020 FOR I=1 TO T:NEXT I
1030 RETURN
1100 REM A SINGLE MORSE DOT
1110 POKE UP,Z
1120 D=1:GOSUB 1000
1130 POKE UP,0
1140 D=1:GOSUB 1000
1150 RETURN
1200 REM A SINGLE MORSE DASH
1210 POKE UP,Z
1220 D=3:GOSUB 1000
1230 POKE UP,0
1240 D=1:GOSUB 1000
1250 RETURN
1300 REM THE SPACE BETWEEN LETTERS
1310 D=2:GOSUB 1000
1320 RETURN
1400 REM THE SPACE BETWEEN WORDS
1410 D=6:GOSUB 1000
1420 RETURN
1500 REM THE MORSE FOR S
1510 GOSUB 1100:REM DOT
1520 GOSUB 1100:REM DOT
1530 GOSUB 1100:REM DOT
1540 RETURN
1600 REM THE MORSE FOR O
1610 GOSUB 1200:REM DASH
1620 GOSUB 1200:REM DASH
1630 GOSUB 1200:REM DASH
1640 RETURN
```

Save the program as “6 MORSE CODE”.

Running the Code

Run the program. You will be prompted for the number of times you want to repeat ‘SOS’.

Challenge

Using the above code as your template, write another program that will allow you to sound any Morse code you choose. Use the following rules:

- The length of a dot is one unit.
- The length of a dash is three units.
- The space between the parts of each letter is one unit.

- The space between letters is three units.
- The space between words is seven units.
- The letter and number codes are:

A	• –	G	– – •	M	– –	S	• • •	Y	– • – –	4	• • • • –
B	– • • •	H	• • • •	N	– •	T	–	Z	– – • •	5	• • • • •
C	– • – •	I	• •	O	– – –	U	• • –	0	– – – – –	6	– • • • •
D	– • •	J	• – – –	P	• – – •	V	• • • –	1	• – – – –	7	– – • • •
E	•	K	– • –	Q	– – • –	W	• – –	2	• • – – –	8	– – – • •
F	• • – •	L	• – • •	R	• – •	X	– • • –	3	• • • – –	9	– – – – •

Advanced Challenge

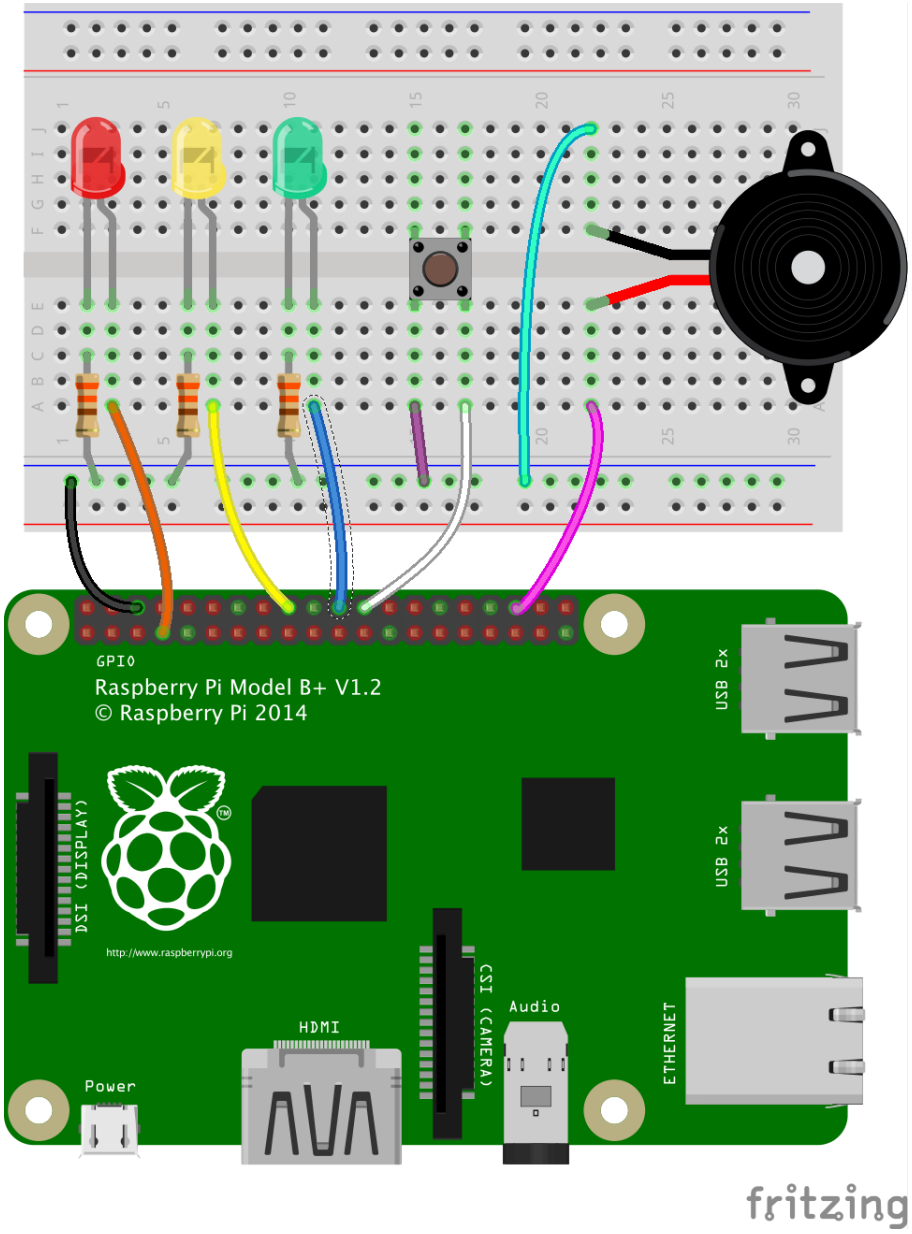
Using what you have learned so far, especially from Section 5, make your own Morse code machine by making the buzzer sound when you press the button.

7 Traffic Lights

Project	Traffic Light Simulator
Description	In this project, you will program a traffic light simulator using the circuit from Section 6.

Equipment Required

The circuit built in Section 6.



Exercise

In this worksheet, you are going to program your Raspberry Pi with EduKit to act like a standard UK Pelican Crossing traffic light. All UK lights work in the same way so that drivers know what to expect when they approach them.

Using the techniques learnt in the previous worksheets, your aim is to make the kit act like the traffic lights at a Pelican Crossing, reacting to the button press to allow pedestrians to cross. The LEDs will act as signals to the vehicles; the buzzer will act as a signal to the pedestrians.

The standard sequence is as follows:

Action	Signal to Vehicles	Signal to Pedestrians	Timings
	Steady Green	Red Standing Figure	
Pedestrian presses the button	Steady Amber (Cars should stop if they can)	Red Standing Figure	3 seconds
	Steady Red (Cars must stop)	Red Standing Figure	1 second
Pedestrians can start walking	Steady Red	Green Walking Figure and Beeping	4 to 7 seconds
Pedestrians should not start to cross	Steady Red	Flashing Green Figure and no sound	2 seconds
	Flashing Amber (Cars can start to go again if the crossing is clear)	Flashing Green Figure	6 seconds
	Flashing Amber	Red Standing Figure	1 second
	Steady Green (Cars can proceed)	Red Standing Figure	At least 20 seconds

On the last step, the button can be pressed within the 20 seconds, but the lights should not change immediately. After the 20 seconds, the process can start again.

Using knowledge from the previous CamJam worksheets, and the outline code, write your traffic light code!

Code Hints

The 'TI' variable gives the number of *jiffies* since your computer was turned on. A jiffy is 1/60th of a second. You can use it to time events or ensure that a certain amount of time has gone by.

Remember that 'PEEK(UP) AND B' will extract just the bit referred to by B from the user port. This can be used to see if that input has been triggered. The input will be 0 if the action happened and B if no action happened.

Outline Code

There are comments where you need to fill in some code.

```
10 REM LED
20 REM SETUP LOCATIONS
30 UP=56577:DDR=56579
40 REM PIN VALUES FOR LEDS AND BUTTON
50 R=1:Y=2:G=4:B=8:Z=16:A=R+Y+G+Z
60 REM SET ONLY PINS 0,1,2, AND 4 FOR OUTPUT
70 POKE DDR,A
80 REM CLEAR USER PORT (ALL PINS OFF)
90 POKE UP,0
100 PRINT CHR$(147);:REM CLEAR SCREEN
110 PRINT "TRAFFIC LIGHTS"
120 GOSUB 1100:REM INITIAL STATE
130 NP=B:REM BUTTON NOT PRESSED
140 S=TI:REM RECORD CURRENT TIME
150 FOR I=1 TO 72:NEXT I:REM 0.1 SECOND PAUSE
160 NP=PEEK(UP) AND B:REM CHECK IF NOT PRESSED
170 REM NP=0: PRESSED, NP=B: NOT PRESSED
180 IF NP=0 THEN GOSUB 1900:REM START SEQUENCE
190 IF NP<>0 THEN 150:REM LOOP WHILE NOT PRESSED
```

```
200 GOTO 130:REM LOOP FOREVER
1000 REM A DELAY OF D SECONDS
1010 T=D*720
1020 FOR I=1 TO T:NEXT I
1030 RETURN
1100 REM SETUP INITIAL STATE: GREEN ON, REST OFF
1102 REM PEDESTRIAN LIGHTS: RED ON, GREEN OFF
1190 RETURN
1200 REM GREEN OFF, AMBER ON FOR 3 SECONDS
1202 REM PEDESTRIAN LIGHTS: RED STILL ON
1290 RETURN
1300 REM AMBER OFF, RED ON FOR 1 SECOND
1302 REM PEDESTRIAN LIGHTS: RED STILL ON
1390 RETURN
1400 REM BUZZER FOR 4 SECONDS
1401 REM 0.5s ON, 0.5s OFF
1402 REM PEDESTRIAN LIGHTS: RED OFF, GREEN ON
1490 RETURN
1500 REM BUZZER OFF FOR 2 SECONDS
1502 REM PEDESTRIAN LIGHTS: GREEN FLASHING
1590 RETURN
1600 REM AMBER FLASHING FOR 6 SECONDS
1602 REM PEDESTRIAN LIGHTS: GREEN FLASHING
1690 RETURN
1700 REM AMBER FLASHING FOR 1 MORE SECOND
1702 REM PEDESTRIAN LIGHTS: GREEN OFF, RED ON
1790 RETURN
1800 REM TRAFFIC LIGHT SEQUENCE
1810 REM CALL SUBROUTINES IN CORRECT ORDER
1890 RETURN
1900 REM PREPARE TO START LIGHT SEQUENCE
1910 N=TI:REM CURRENT TIME
1920 REM KEEP CHECKING UNTIL 20 SECONDS REACHED
1930 IF (N-S)/60 <= 20 THEN 1910
1940 GOSUB 1800:REM RUN TRAFFIC LIGHT SEQUENCE
1950 RETURN
```

Save your program as “7 TRAFFIC LIGHTS”.

Running the Code

Run the code. If errors are reported, check your code again. Press the button while the green LED is lit and see what happens.

Challenge

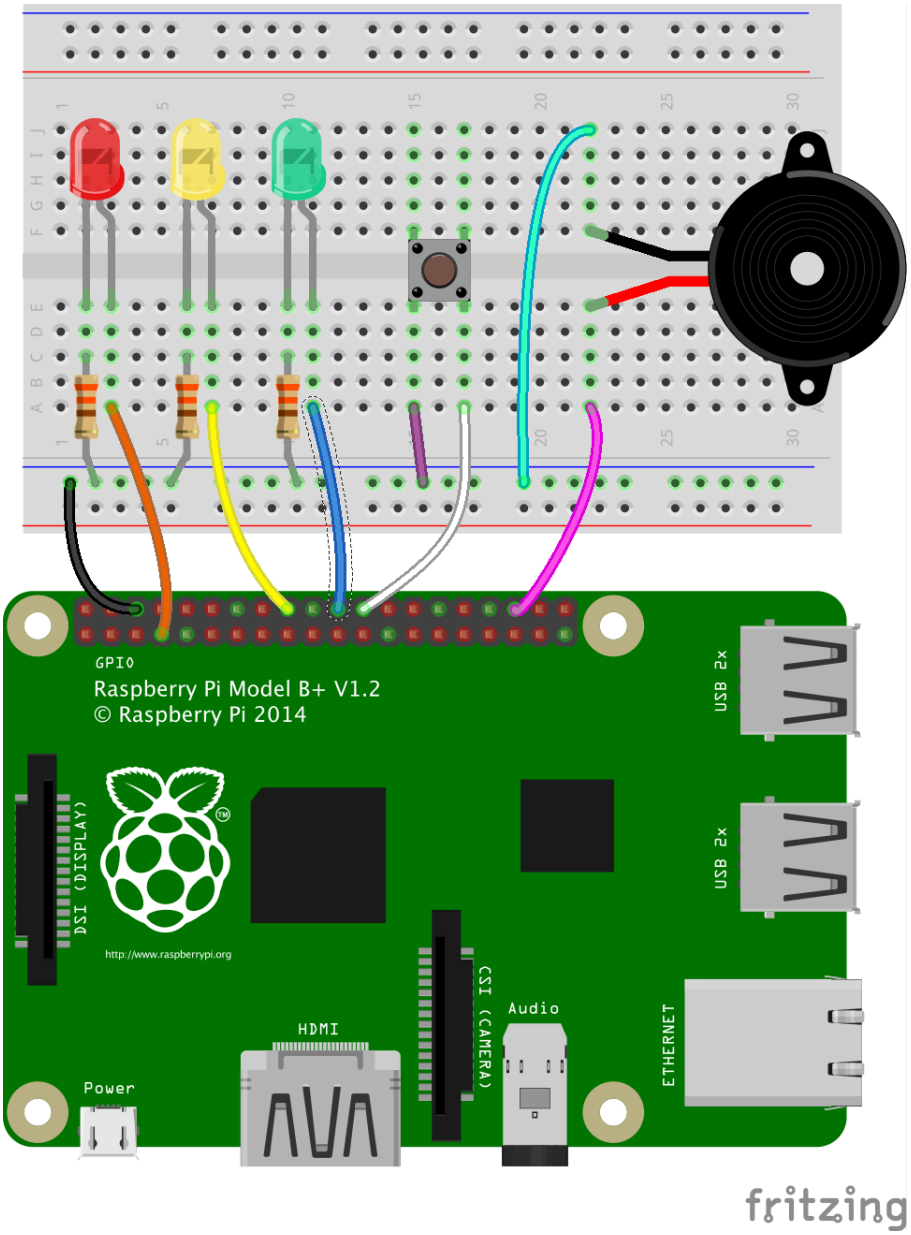
Use a second CamJam EduKit, or additional red and green LEDs and 330Ω resistors, and use them as the Red and Green pedestrian figures.

8 Games

Project	Games!
Description	Computers and electronics are not just about having fun—you can play games as well!

Equipment Required

The circuit built in Section 6.



Exercises

In this worksheet, you're not going to be given any code. You are just going to be given some ideas for games you can make and play using the CamJam EduKit, and some hints that you might need.

Hints

Random numbers can be generated with 'RND(1)'. This will generate a random number between 0 and 1. To generate a number between 1 and X, you can use 'INT(RND(1)*X+1)' as in the following code:

```
10 PRINT "RANDOM NUMBER BETEWEEN 1 AND 10: ";
20 PRINT INT(RND(1)*10+1)
```

Print out instructions for the user to the screen using the 'PRINT' command:

```
10 PRINT "TELL THE USER WHAT TO DO"
```

Game 1—Reaction Timer

Light the LEDs one at a time, going from Red to Green with one second in between. After the green has been lit, wait a random length of time (say, between 1 and 5 seconds) before sounding the buzzer. Time how long it takes from the buzzer sounding to the player pressing the button.

Game 2—Eat the Orange

Randomly choose which LED to light. Light it for 0.2 seconds. If the player presses the button while the Amber LED is lit, they get a point. Time how long it takes for them to get 10 points.

Game 3—Segment of Orange

Change the colour of the LEDs from red, to amber, to green, to amber, and back to red every 0.1s. If the player presses the button when the amber LED is lit, they get a point. Only give them 10 chances to hit the button when the amber is lit.