**4.1.2 Purpose and use of microcontrollers as opposed to microprocessors**

**Numbering System (In response to comment I feel its important for students to understand numbering System)**

Number System is a way to represent numbers in computer architecture. There are four different types of the number system, such as:

1. Binary number system (base 2)
2. [Octal number system (base 8)](https://byjus.com/maths/octal-number-system/)
3. [Decimal number system(base 10)](https://byjus.com/maths/decimal-number-system/)
4. [Hexadecimal number system (base 16)](https://byjus.com/maths/hexadecimal-number-system/).

Let's discuss what the binary number system.

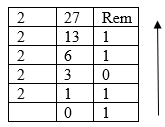
**Binary Number System:** According to digital electronics and mathematics, a binary number is defined as a number that is expressed in the binary system or base 2 numeral system. It describes numeric values by two separate symbols: 1 (one) and 0 (zero). The base-2 system is the positional notation with 2 as a radix.

The binary system is applied internally by almost all latest computers and computer-based devices because of its direct implementation in electronic circuits using logic gates. A single binary digit is called a “Bit”. A binary number consists of several bits. Examples are:

* 10101 is a five-bit binary number
* 101 is a three-bit binary number
* 100001 is a six-bit binary number

A series of 8 bits makes up a byte. Here is a simple illustration on how to convert decimal numbers to binary numbers. A binary number can thus be converted to a decimal number by understanding the position of each bit.

Example: Convert 27 to its binary notation. We divide the value 27 by 2, and the remainders are read from bottom to the top to get our answer.



2710=110112

We can use 8 bits to represent the above number as 000110112. If zeros are added in front of our result, they are not changing the value of our number. We can prove this by raising the result to powers of 2 and then adding the results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

128 x0 + (64 x 0) + (32 x 0) + (16 x1) + (8 x 1)+ (4 x 0) +(2 x 1) +( 1 x 1)

=16 +8 +2 +1

=27

4.1.4  **Differentiating between microcontrollers and microcomputers**

Microprocessors are much faster than microcontrollers. The clock speed of a microprocessor is above 1 GHz. While in the case of the Microcontroller, the clock speed is 200MHz or more, depending on the architecture.

Comparison between microcomputer and microprocessor

|  |  |
| --- | --- |
| **Microcontroller** | **Microprocessor** |
| Microcontrollers are used to execute a single task within an application. | Microprocessors are used for big applications. |
| Its designing and hardware cost is low. | Its designing and hardware cost is high. |
| Easy to replace. | Not so easy to replace. |
| It is built with CMOS technology, which requires less power to operate. | Its power consumption is high because it has to control the entire system. |
| It consists of CPU, RAM, ROM, I/O ports. | It doesn’t consist of RAM, ROM, I/O ports. It uses its pin |

## **4.1.5 Types of Microcontrollers**

Microcontrollers are divided into various categories based on memory, bits architecture and instruction set. In this chapter we will explain classification of microcontrollers according to memory, architecture and according to bits.

Classification according to bits (I see this has been inserted already)

1. 8-bit
2. 16-bit
3. 32-bit

**Classification according to memory**

Based on the memory configuration, the microcontroller is further divided into two categories.

* **External memory microcontroller-** Microcontrollers of this type do not have a program memory built into the chip. Thus, it is referred to as an external memory microcontroller. An example would be the Intel 8031 microcontroller.
* **Embedded memory microcontroller** - An embedded microcontroller is designed so that all the programs, data memory, counters, timers, interrupts, and I/O ports are integrated onto the chip. Typical microcontrollers include the Intel 8051.

**Classification according to Architecture**

The memory architecture of microcontroller are two types, they are namely:

* **Harvard memory architecture microcontroller**- A microcontroller that has a dissimilar memory address space for its program and data memory is said to have Harvard memory architecture.
* **Princeton memory architecture microcontroller**- Microcontrollers with Princeton memory architecture has a common memory address for program and data memory.

**Applications of Microcontrollers**

There are a variety of different devices that utilize microcontrollers, such as −

* Light sensing and controlling devices like LED.
* Temperature sensing and controlling devices like microwave oven, chimneys.
* Fire detection and safety devices like Fire alarm.
* Measuring devices like Voltmeter.

**Applications of Microprocessor**

Microprocessors are mainly used in devices like:

* Calculators
* Accounting system
* Complex industrial controllers
* Traffic light
* Defence systems
* Mobile phones
* Automobiles
* CD/DVD players, Washing machines, Cameras, Microwave oven
* Security alarms
* Watches, Mp3 players, Games machine

**4.1.7 Similarities and differences between mainstream microcontrollers and single- board microprocessor computer boards**

A microcontroller has a lot in common with an SBC. Both have a processor, storage, and input/output peripherals. But this is where the similarities end.

There are fewer resources on microcontrollers than on SBCs. Instead of MBs or GBs of storage on SBCs, microcontrollers may have KBs. Similarly, their processing capabilities are also limited. In most cases, a microcontroller also needs breakout boards to make programming them easier. An Arduino Uno is a suitable example of a microcontroller with a breakout board. It allows you to connect the microcontroller, in this case, an Atmel ATmega328P, with input/output devices, power, and peripherals.

An SBC is a self-contained computer capable of running an OS, whereas a microcontroller is just a chip with limited resources. Additionally, unlike SBCs, microcontrollers are designed to run only one program at a time. SBCs like Raspberry Pie run Linux, whereas Arduino Uno does not.

Since single-board computers and microcontrollers share a lot in common, telling them apart can be difficult. Just remember that microcontrollers are meant to perform simple, repetitive tasks, like turning on and off a button. Since these tasks aren't resource-intensive, microcontrollers aren't as capable as SBCs.

**4.1.8 Components of a microcontroller board and a microprocessor board**

**Components of a microcontroller**

**Sections missing after the diagram**

**Oscillator circuit:**

This is a circuit containing capacity and inductance such that a single voltage impulse would give rise to a damped alternating current. Oscillators generate the heartbeat of every microcontroller and produce clock signals essential for synchronizing internal operations. In microcontrollers, the clock signals can be generated using either mechanical resonant devices or electrical phase shift circuits

**Special functioning block**

Some microcontrollers used only for some special applications (e.g., space systems and robotics) these controllers containing additional ports to perform such special operations. This considered as special functioning block. This block has extra ports to carry out some operations.

**4.2.8 Breadboard connection to GPIO**

A series circuit is a circuit in which resistors or loads are connected end to end so that the circuit will have only one path through which electric current flows.

**6.1.3**

**Output**

Digital output

The Raspberry Pi produces a digital signal when it draws a digital signal from the computer. This is known as a digital output. Using an LED, a digital output can be seen. In the event that the LED glows, then the voltage is HIGH, and in the event that it doesn't glow, the voltage is LOW.

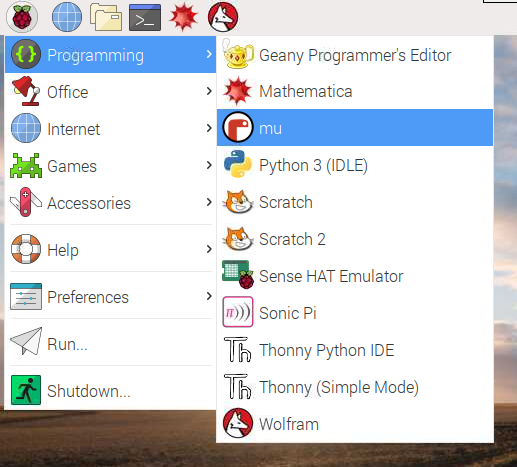
**6.1.6 Test and calibrate the kit**

There might be some differences in the calibration process for different soldering irons when it comes to calibrating them. We will, however, cover the basic steps involved in calibrating a soldering iron in this unit.

Install a set of tip sensors on the tip thermometer. The sensors should be looped onto the thermometer board. Then, match the blue and red sides of the sensor wire with the blue and red dots on your thermometer board.

**6.1.7 Write code in a simulation environment to achieve a particular task (including modifying and debugging)**

**Starting Mu IDE**



**[MISSING ACTUAL CODE TO ACHIEVE SPECIFIC TASK/INSTRUCTION HOW TO ACTUALLY WRITE CODE]**

**6.1.8 Deploy the code in Mu**

The main area in *mu* is where you will be writing your code. Enter the code supplied below to print numbers from 1 to 10

**Program to generate numbers from 1 to 10 using a loop.**

#using a while loop

#set the starting point

start=1

#add the terminating condition

while start <11:

#display the numbers

print(f'Number {start} is :' , start)

#increment the counter

start=start+1

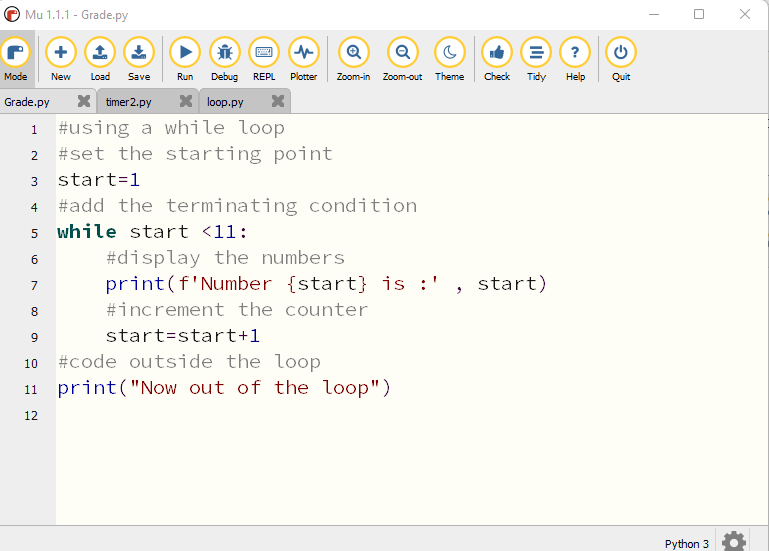
#code outside the loop

print("Now out of the loop")

Replace ***Figure 6.15: Writing “Hello world”***

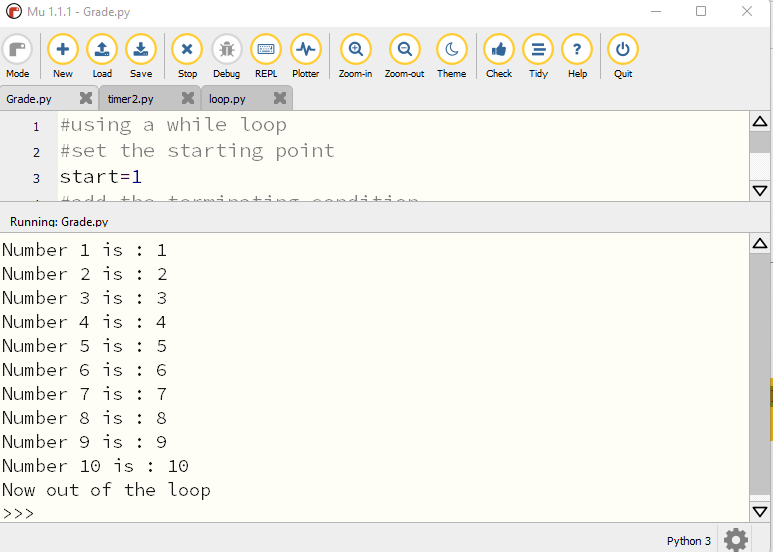


Click the Save icon to save your program on the local disk or external storage device. Enter the file name *code* and click Save in the window. Click the Run icon to run the program and check your output, as shown on the screen below.





***Figure 6.16: Selecting the option to run a program***



***Replace Figure 6.18: Selecting the option to run a program***

**6.1.9 Test the code effectiveness based on specifications**

Mu editor has a debug option that allows an analysis of the code step-by-step when running the program. On the right-hand side panel, the developer can see step by step output when variables change values by clicking the step over button.

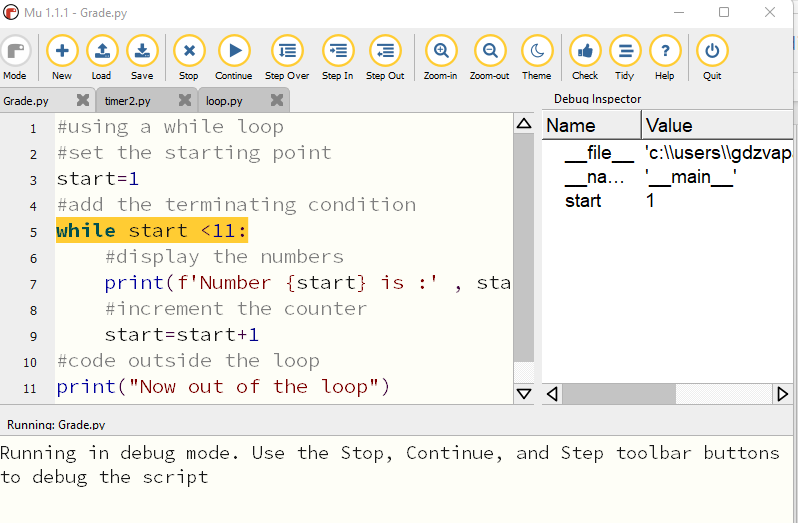


Figure 6.19: Debugging the Python Code

**6.2.3 Understanding the components of the design**

As illustrated in in Figure 6.19 (20 after additional diagram), you will need to connect the components listed below

* Raspberry Pi
* PushButton
* 3 Resistors (330 Ohm)
* 3 Led Lights (Red, Orange and Green)
* Buzzer
* Breadboard (Optional)

**Connections**

GPIO25 Red Led

GPIO8 Orange Led

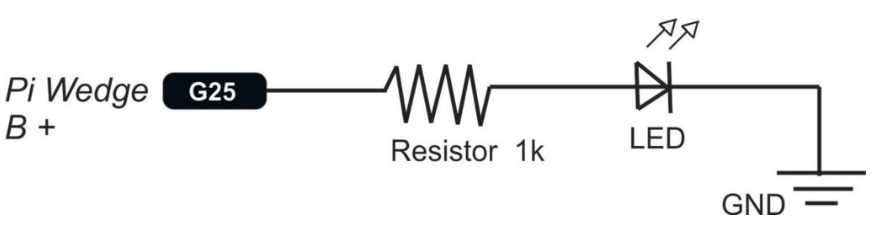
GPIO7 Green Led

GPIO15 Buzzer

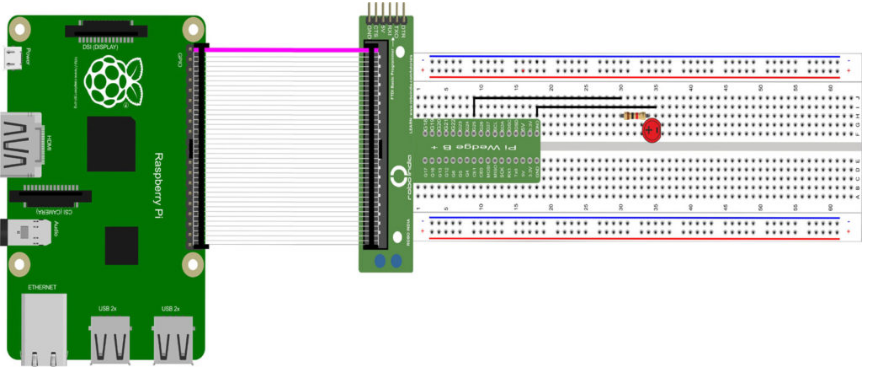
GPIO20 PushButton

**6.2.6 Design the applicable circuit on paper**

Fixure



**6.2.7 Design the applicable circuit schematic using an appropriate software prototyping and design tool**



**Using Mu**

Open *mu* and choose the mode you want to use (as set out in section 6.1.7). Use Python 3 if you are creating a new Python script. You can switch an LED on and off by typing commands directly into the REPL screen which you can activate by clicking on the REPL button in the menu bar. In layman's terms, REPL stands for "Read, Evaluate, Print, Loop", which is a concise description of what the panel does for you. It reads instructions that you type in Python, evaluates the meaning, prints any results it has for you, and then loops back to wait for your next Python command. You can try using REPL on your own. For now, we are going to use the code section for Mu to enter our code.

First import the GPIO Zero and time library and tell the Pi which GPIO pin you are using – in this case pin 17.

Here is the code

from gpiozero import LED

from time import sleep

led=LED(17)

while True :

led.on()

sleep(1)

led.off()

sleep(1)

**Note:**

The code inside the loop is indented. Run your program and you will see the LED turning on and off.

**Manually controlling the LED**

from gpiozero import LED,Button

from time import sleep

led=LED(17)

button=Button(2)

button.wait\_for\_press()

led.on()

sleep(3)

led.off()

**Making a switch**

Modify your code as follows:

from gpiozero import LED,Button

from time import sleep

led=LED(17)

button=Button(2)

while True:

button.wait\_for\_press()

led.toggle()

sleep(0.5)

**6.2.9 Develop, code and debug the source code for the operations of the artefact**

from gpiozero import Button, TrafficLights, Buzzer

from time import sleep

buzzer = Buzzer(15)

button = Button(21)

lights = TrafficLights(25, 8, 7)

while True:

button.wait\_for\_press()

buzzer.on()

light.green.on()

sleep(1)

lights.amber.on()

sleep(1)

lights.red.on()

sleep(1)

lights.off()

buzzer.off()

**6.2.10 Discuss the composition of the code**

By now you understand the following libraries:

* Gpiozero
* Time

Instead of using the LED module like before when we were lighting the LED, we implemented Traffic lights. You have three LEDs: red, amber, and green. Perfect for traffic lights! There TrafficLights provides built-in interface for traffic lights in GPIO Zero.

Our code also included a pedestrian crossing button. As soon as the button is pressed, the lights should turn red (not immediately), giving pedestrians time to cross before returning to green.

For visually impaired pedestrians, we also installed a buzzer that beeps quickly when it is safe to cross.

**6.2.11 Move the applicable code to the controller**

**6.2.12 Finish and present the design and artefact**

My interpretation is students must do the project here and present a real robot.