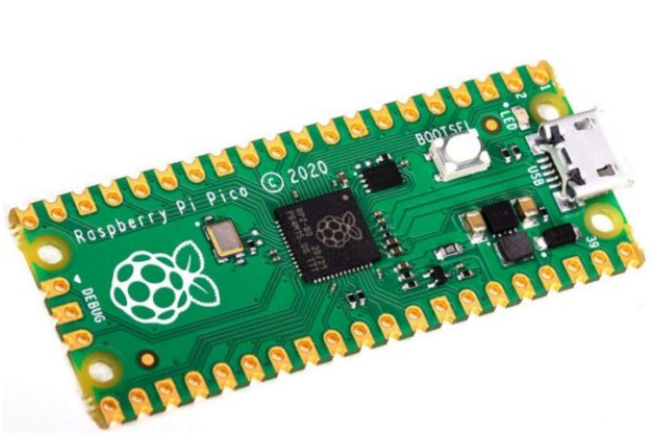
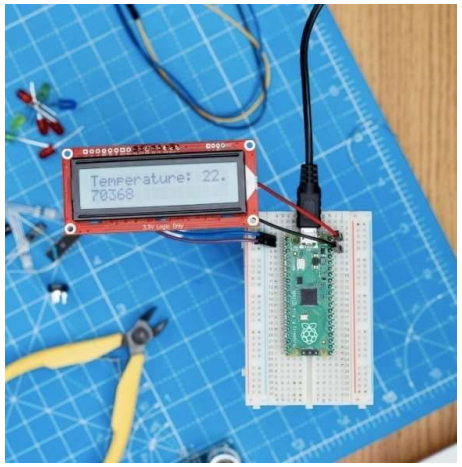
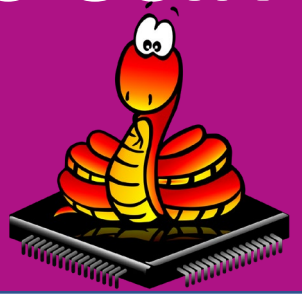


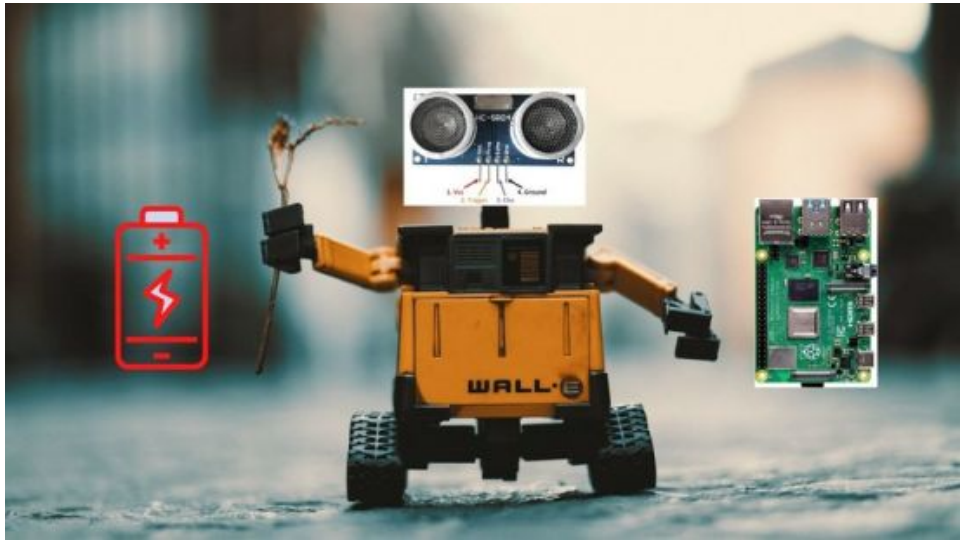


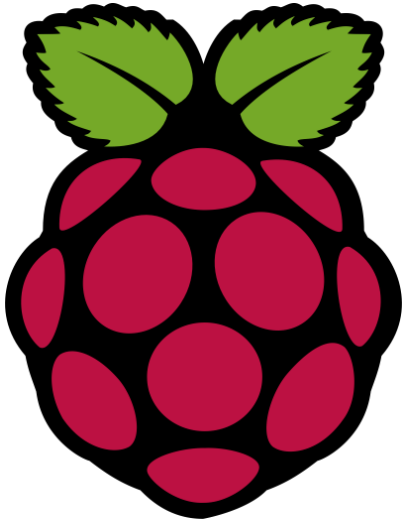
INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

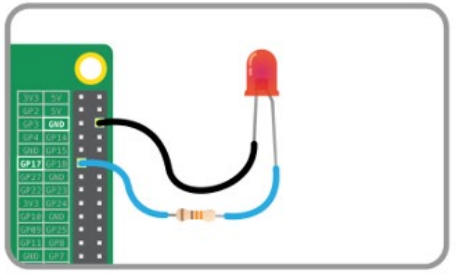
(CSE 4110)

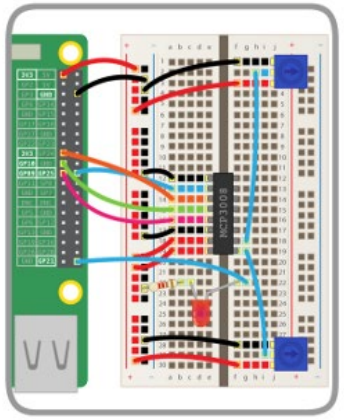
(LECTURE – 5)







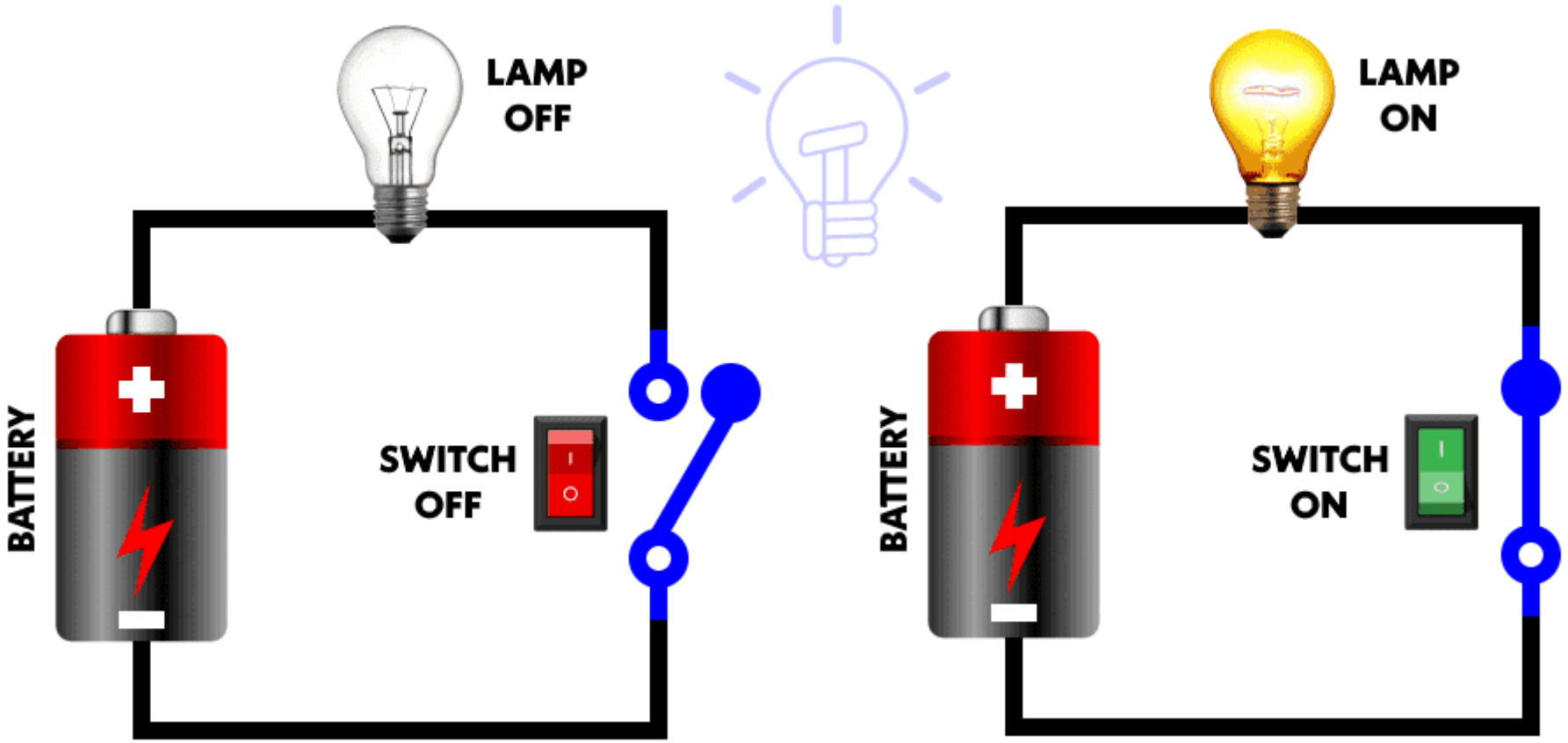




What is a Switch?

A switch is a device which is used to make or break an electric circuit automatically or manually.

Switches are a part of the control system and without it, control operation cannot be achieved. A switch can perform two functions, namely fully ON (by closing its contacts) or fully OFF (by opening its contacts).



Working of an Electrical Switch

The controlling operation of a switch can be defined by its “Pole” and “Throw“. A Pole represents the number of operations controlled by a single switch. The Throw indicates the number of contacts in a switch.

For example, the NO (Normally Open) and NC (Normally Closed) are Single Throw which is used to control the circuit by making/breaking contacts of the switch.

The intermediate and changeover switches are Double Throw which is used to control the two way operation of the circuit by closing/opening contacts of the switches which is used to divert the flow of current from one circuit to another.

A Single-Pole-Single Throw (SPST) switch is known for controlling a single circuit (e.g. ON/OFF) per operation. This way, the number of poles and throws are used to represent the controlling process of a switch.

Types of Switches

- **Mechanical** (activated physically, by moving, pressing, releasing, or touching its contacts.)
- **Electronic** (do not require any physical contact in order to control a circuit)
- **Electro-Mechanical**

The number of **poles** on a switch defines how many **separate circuits** the switch can control. So a switch with one pole, can only influence one single circuit. A four-pole switch can separately control four different circuits.

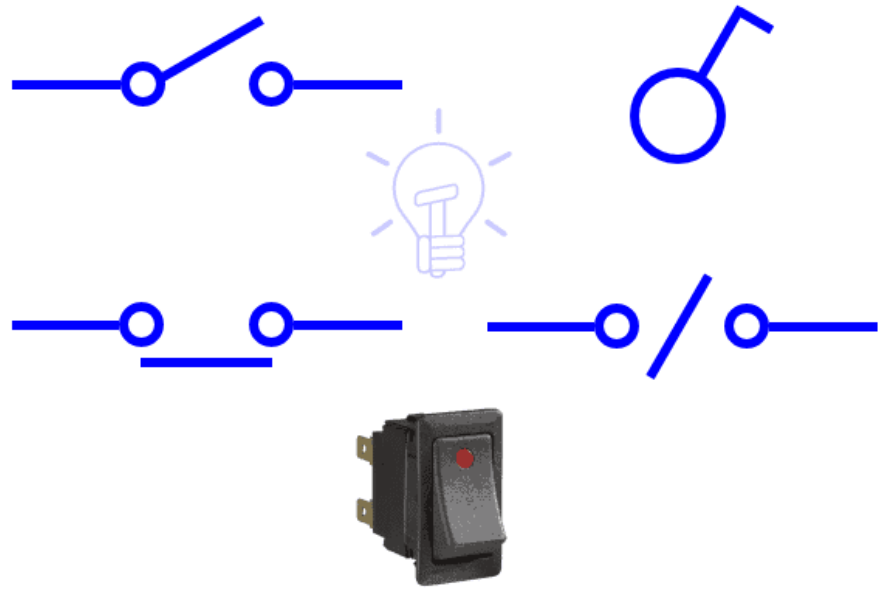
A switch's **throw**-count defines how many **positions** each of the switch's poles can be connected to. For example, if a switch has two throws, each circuit (pole) in the switch can be connected to one of two terminals.

Mechanical Switch

- ✓ **Momentary Switches** (like push buttons, for example) are used to make momentary contact (for a brief time or as long the button is pressed).
- ✓ **Latched Switches**, maintain the contact until it is forced to the other position.

SPST (Single Pole Single Throw)

- ➢ A **simple ON/OFF switch** commonly found in our homes for lighting circuits and small load appliances as well as computers and devices.
- ➢ It is also called a **"One Way" or "Single Way"** Switch
- ➢ It controls single operation in a circuit
- ➢ The contact of the SPST switch can be either **NO (Normally Open)** for OFF position or **NC (Normally Closed)** for ON position.



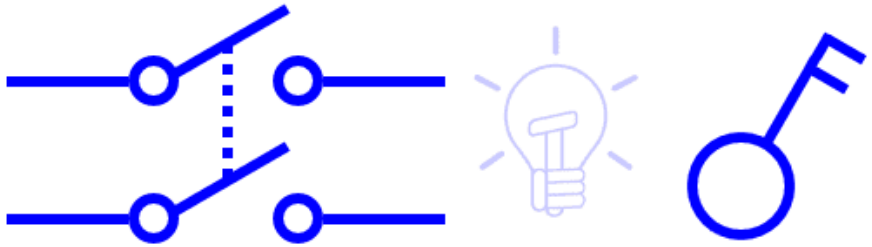
Mechanical Switch

SPDT (Single Pole Double Throw)

- SPDT switch has mainly three pins (terminals) and an extra as ground terminal. The input terminal is known and used as “common” which is also called a **Two-Way Switch**
- **Two circuits can be controlled** at the same time while using this switch.
- The remaining two terminals are called **travelers (output terminals).**

DPST (Double Pole, Single Throw)

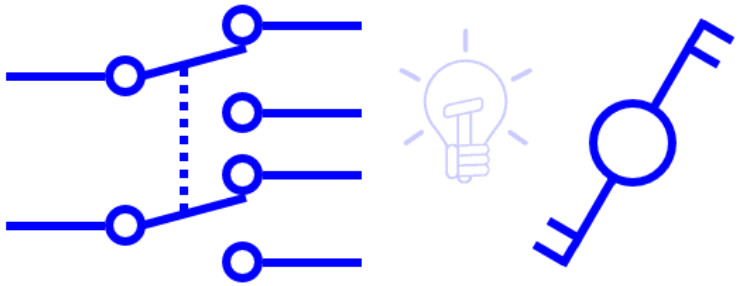
- DPST switch is basically **two SPST switches in one package** and can be operated by a **single lever**.
- This switch is mostly used where both ground and line need to be broken (or closed) at the same time, same as the operation of a 2-Pole breaker.
- It has **two poles** i.e. it can control two circuits (Hot and Neutral) and a Single Throw i.e. it can only make one operation e.g. ON or OFF.
- Double Pole, Single Throw switch has **four terminal pins i.e. 2 as Input and the rest of 2 as output.** DPST switches are used to **control a single circuit** while both the contacts are needed to be actuated.



Mechanical Switch

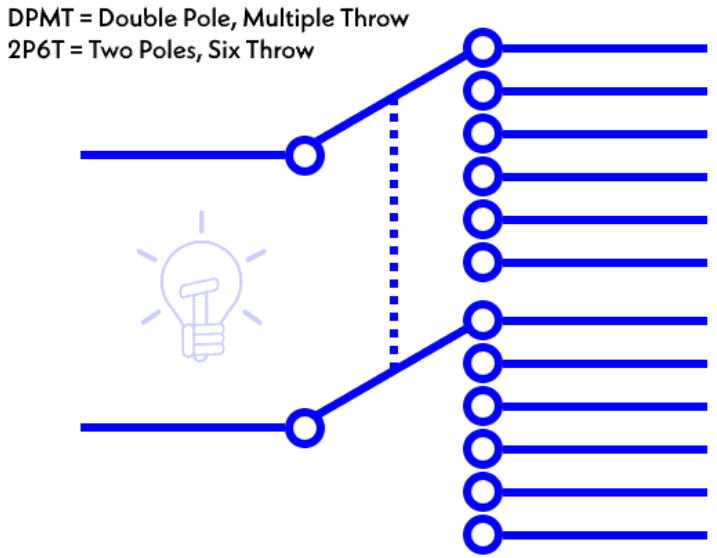
DPDT (Double Pole Double Throw)

- DPDT switch is equivalent to **two SPDT switches packaged in one unit.**
- This switch has **two common pins** and **four signal pins** (total of 6 terminals).
- Total four different combinations of signals can be applied to the input pins of this switch.
- DPDT switches are used to **control two different electric circuits at the same time**. While it has a common lever for both operations, It can be used for two different operation i.e. ON & OFF positions.



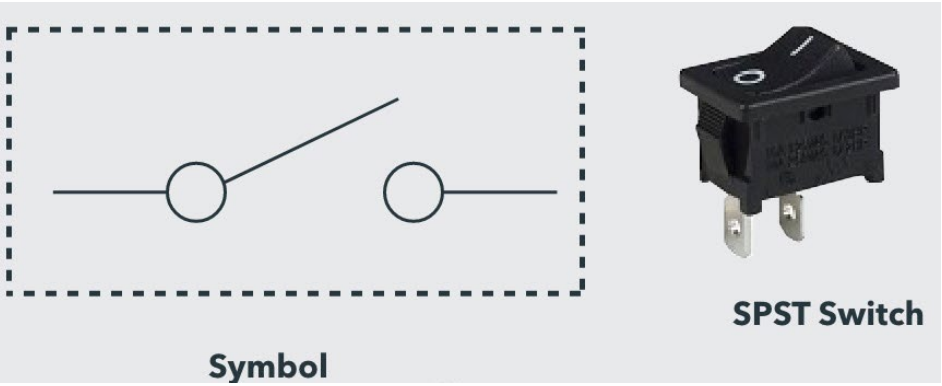
DPMT (Double Pole Multi Throw)

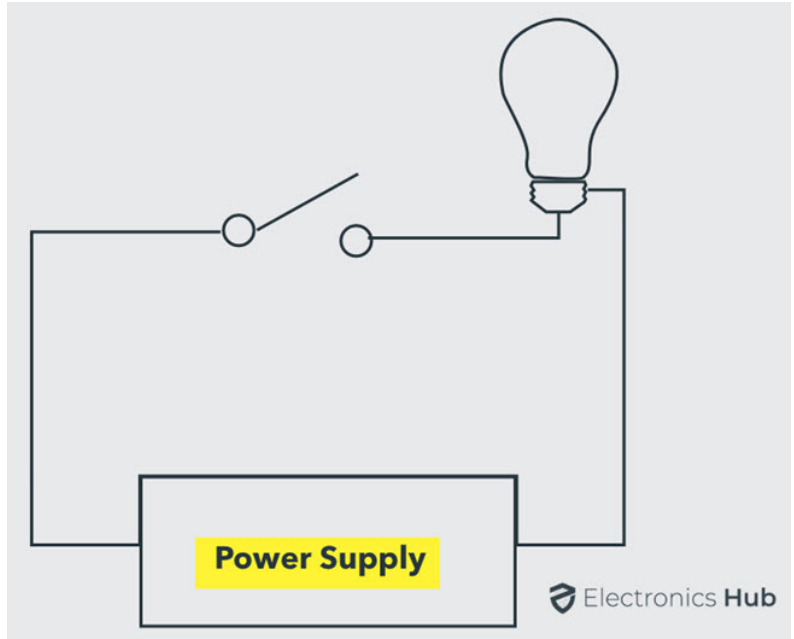
These kinds of switches consist of 2-poles and multiple throw i.e. they can be used to control two independent circuits. These types of switches with a common lever are used as changeover and selectors switches for multiway switching.



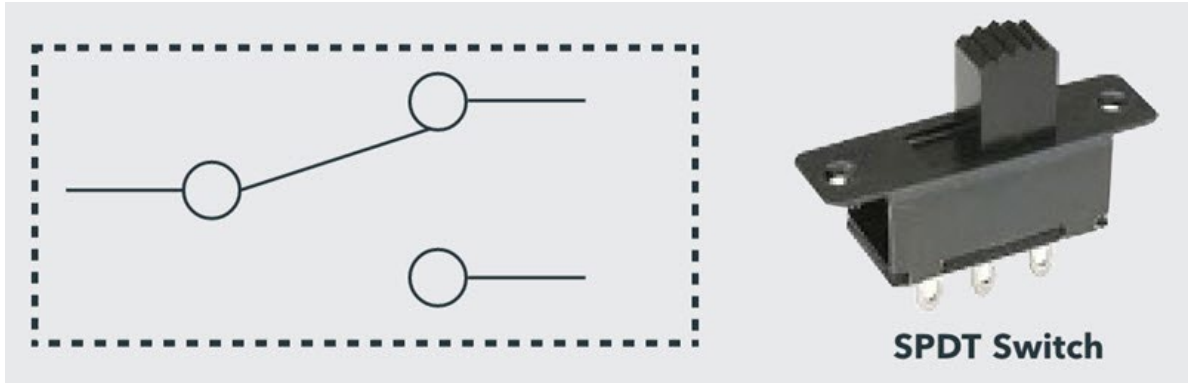
Mechanical Switch

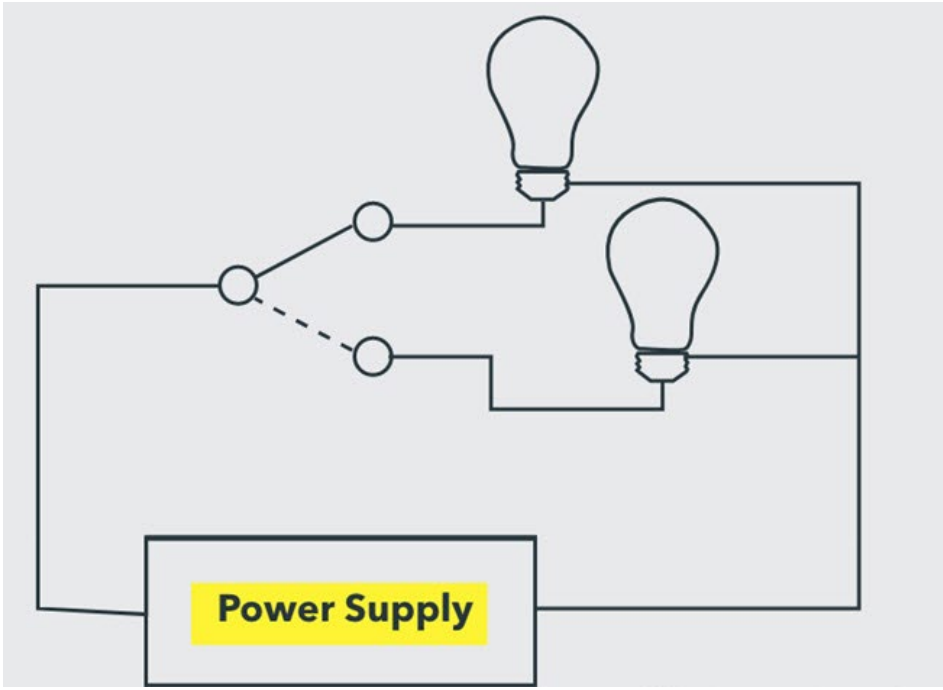
Single Pole Single Throw Switch (SPST)





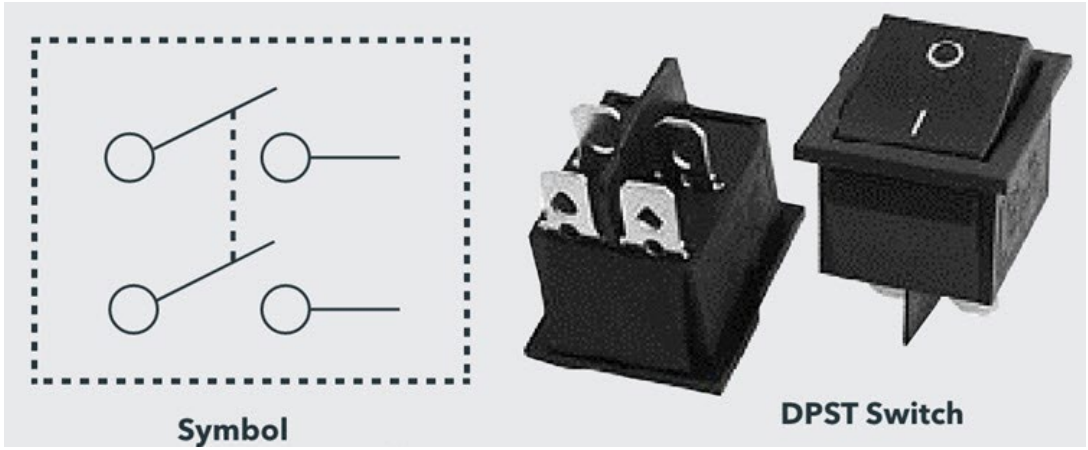
Single Pole Double Throw Switch (SPDT)

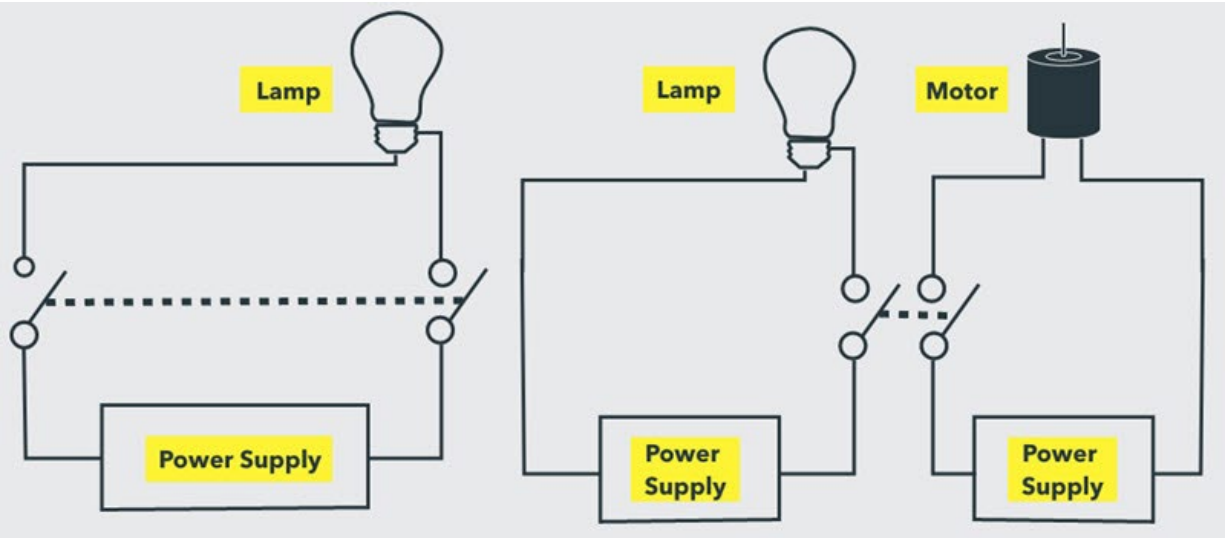




Mechanical Switch

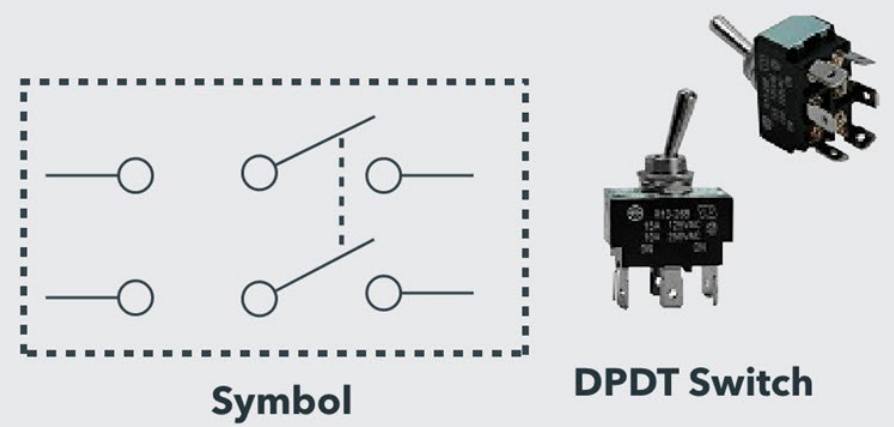
Double Pole Single Throw Switch (DPST)

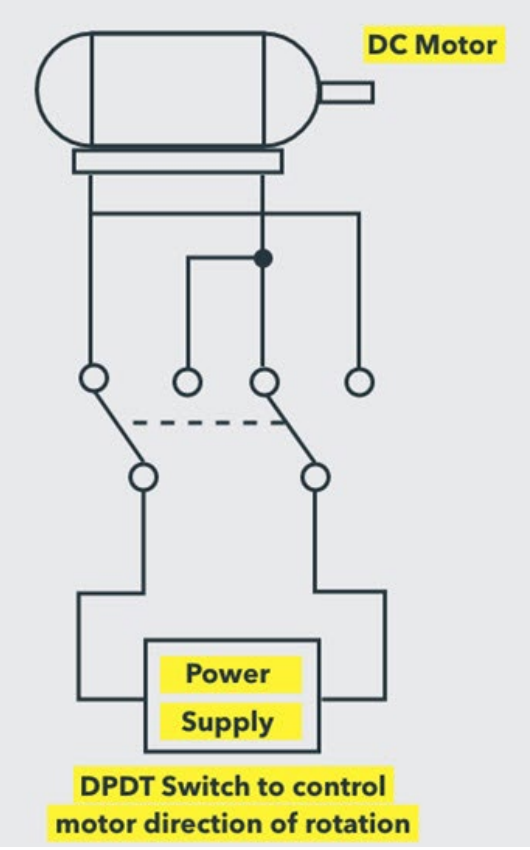




- This switch consists of four terminals: two input contacts and two output contacts.
- It behaves like a two separate SPST configurations, operating at the same time.
- It has only one ON position, but it can actuate the two contacts simultaneously, such that each input contact will be connected to its corresponding output contact.
- In OFF position both switches are at open state.
- This type of switches is used for controlling two different circuits at a time.
- Also, the contacts of this switch may be either normally open or normally closed configurations.

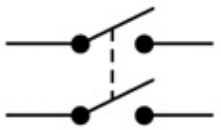
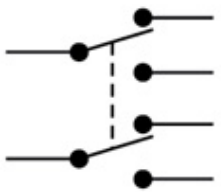
Double Pole Double Throw Switch (DPDT)

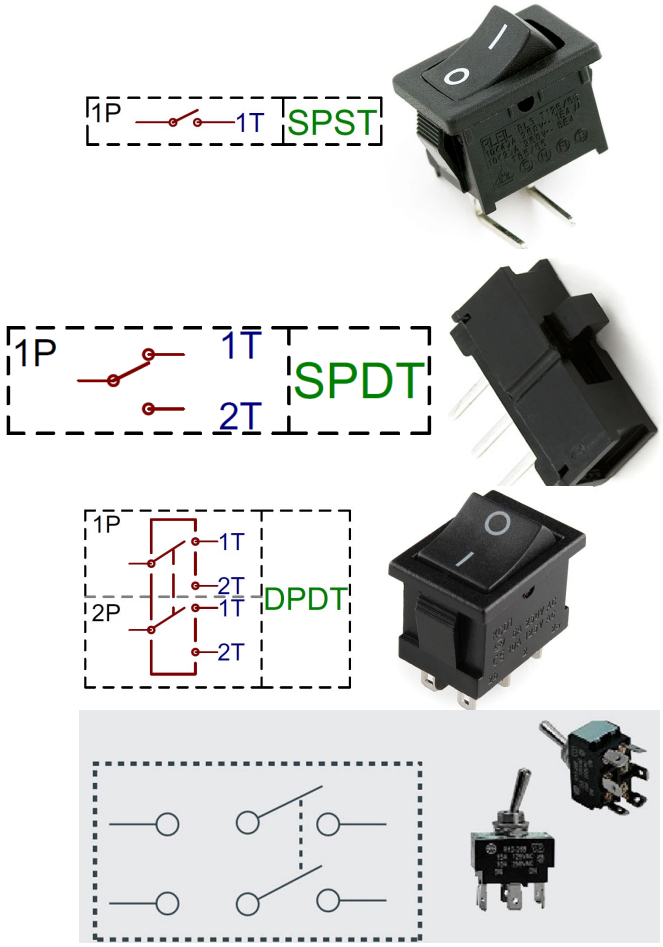




- This is a dual ON/OFF switch consisting of two ON positions.
- It has six terminals, two are input contacts and remaining four are the output contacts.
- It behaves like a two separate SPDT configuration, operating at the same time.
- Two input contacts are connected to the one set of output contacts in one position and in another position, input contacts are connected to the other set of output contacts.

Mechanical Switche

Switch Type	Desig.	Symbol	Mechanism
Single-Pole, Single-Throw	SPST		OFF-ON OFF-(ON)
			OFF-(ON)
Single-Pole, Double-Throw	SPDT		ON-NONE-ON ON-OFF-ON (ON)-OFF-(ON) ON-OFF-(ON)
Double-Pole, Single-Throw	DPST		OFF-ON OFF-(ON)
Double-Pole, Double-Throw	DPDT		ON-NONE-ON ON-OFF-ON (ON)-OFF-(ON) ON-OFF-(ON)



SPST, *also known as* ***1P1T***
Single pole, single throw

DPST *also known as* ***2P1T***
Double pole, single throw

SPDT *also known as* ***1P2T***
Single pole, double throw

3PST *also known as* ***3P1T***
Three pole, single throw

Mechanical Switch

Toggle Switch

- Toggle switches are **latched type** of switches which are actuated by **a lever angled in one or more directions.**
- This switch is **stable** in state and remains in that state unless or until the lever is pushed in another direction.
- Most of all household applications (such as lighting control switches) have toggle switch and it can fall into any category as mentioned above e.g. SPST, DPDT, DPST, DPDT etc.
- They are available for high current applications up to 30+ amperes and can be used for small current switching operations.

Slide Switch

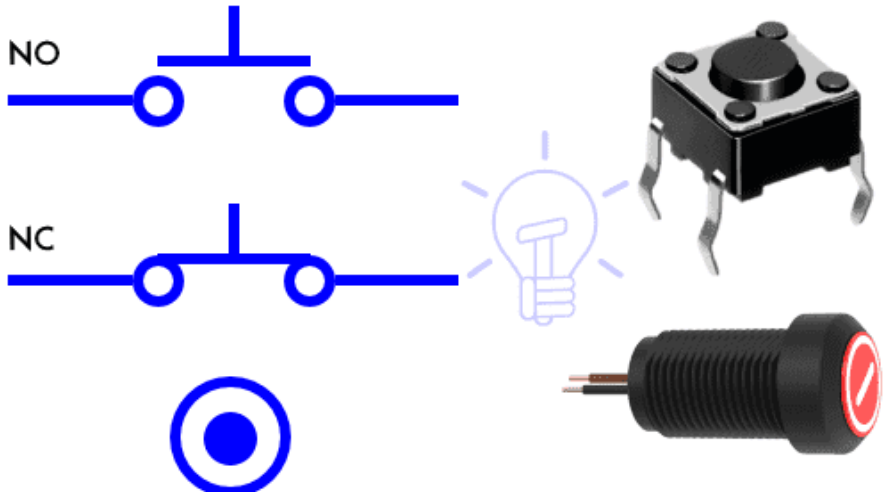
- A slide switch uses a slider as actuator which slides back and forth to make and break the contacts.

SPDT (single pole double throw) slide switch

Mechanical Switch

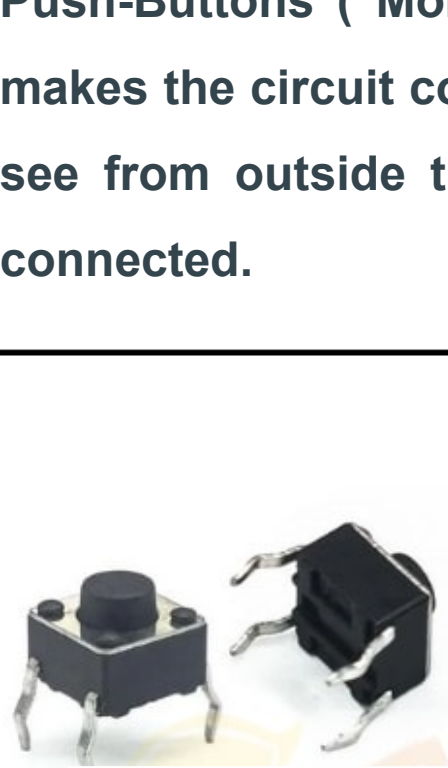
Push Buttons Switch

- push buttons are momentary switches which are operated by pressings (put a pressure by pushing) it for a while. The spring mechanism for the actuator inside it is used to close or open the circuit for OFF and ON operation.
- When a pushbutton switch is pressed, the movable contacts attached to the button make sure to connect the static (stationary or stable) contacts in series to make the circuit. When the pressure is released, contacts of the pins are detached and the circuit operation returns back to the first position either ON or OFF.
- They are generally NO, NC or double acting which is used to control two different circuits. Examples of push-button switches are drills, blowers and doorbells etc.



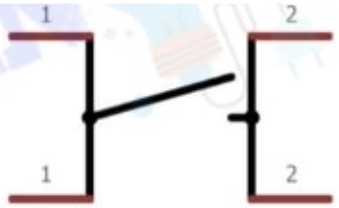
Push Button Switch

Push-Buttons (Momentary Switch) are normally-open tactile switches. When we pressed the button it makes the circuit connected and when released the button it makes the circuit connection breaks. We can see from outside this switch is consists of four terminals. but both side of the terminal is internally connected.

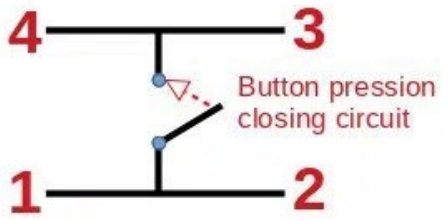


Push Button Switch

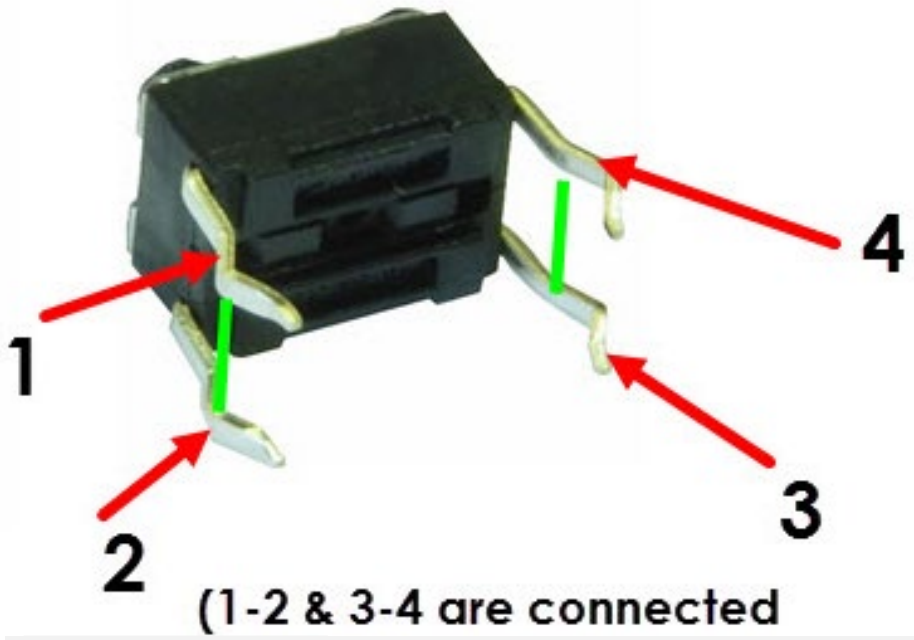
Internal Connection



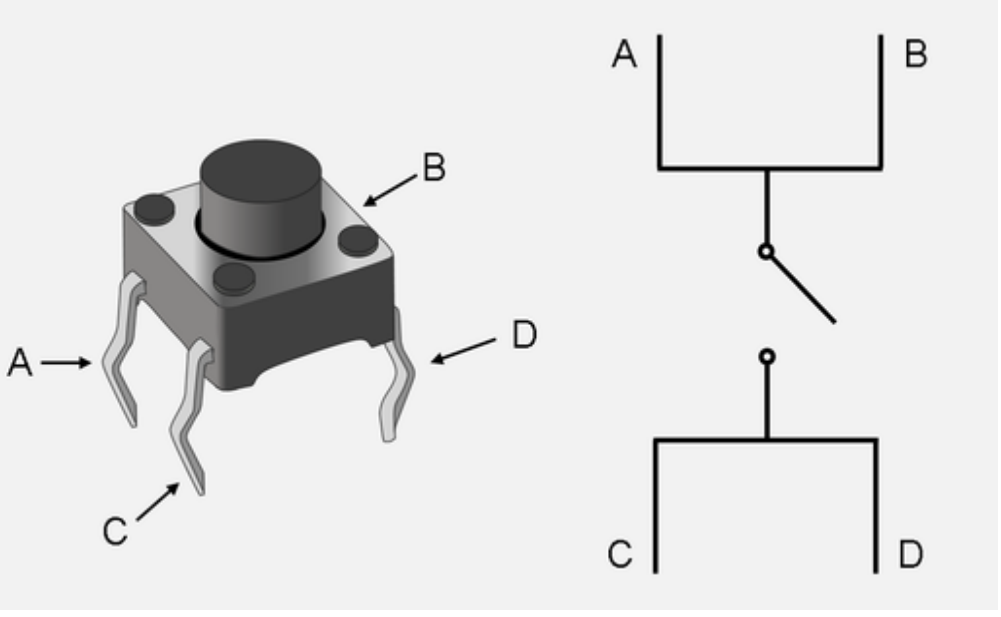
Symbol



Push Button Switch

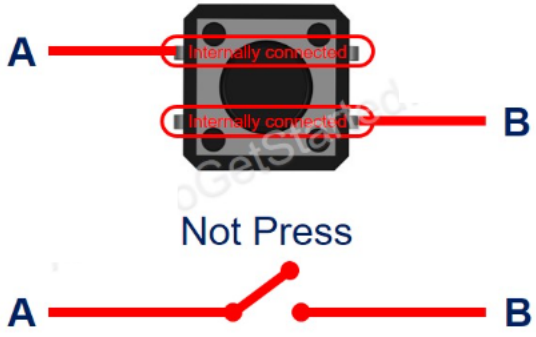


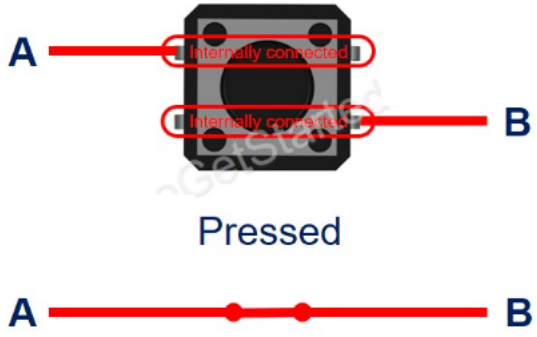
(1-2 & 3-4 are connected



How It Works

- When the button is NOT pressed, pin A is NOT connected to pin B
- When the button is pressed, pin A is connected to pin B





However, these pins are internally connected in pairs. Therefore, we only need to use two of the four pins, which are NOT internally connected.

There are four ways (actually two ways because of symmetry) to connect to button (see image)

We can use only two pins of a button, why does it have four pins?

⇒ To make it stand firmly in PCB (board) to resist the pressing force.

GPIO of Raspberry Pi as Input

- If the Raspberry Pi wants to read the value from an external device, the corresponding GPIO pin must be declared as an Input Pin.

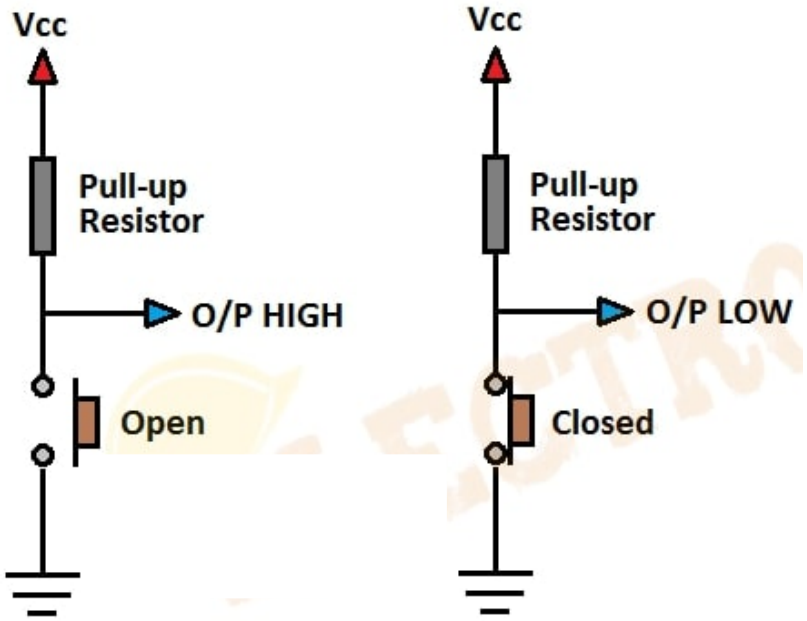
- But when a GPIO Pin of the Raspberry Pi is declared as Input, it must be ‘tied’ to High or Low or else it is called as a Floating Input Pin. A Floating Input is a pin which is defined as input and left as it is.

- Any Digital Input Pin is very sensitive and catches even the slightest changes and will pick up the stray capacitances from your finger, breadboard, air etc.

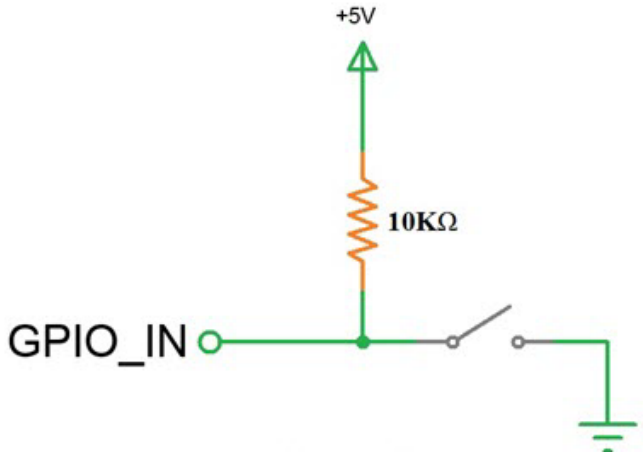
- In order to avoid this, a Digital Input Pin must be tied to VCC or GND with the help of Pull-up or Pull-down resistors.

Pull-Up and Pull-Down Resistor

Pull-up Resistors



Pull-up Resistor Circuit



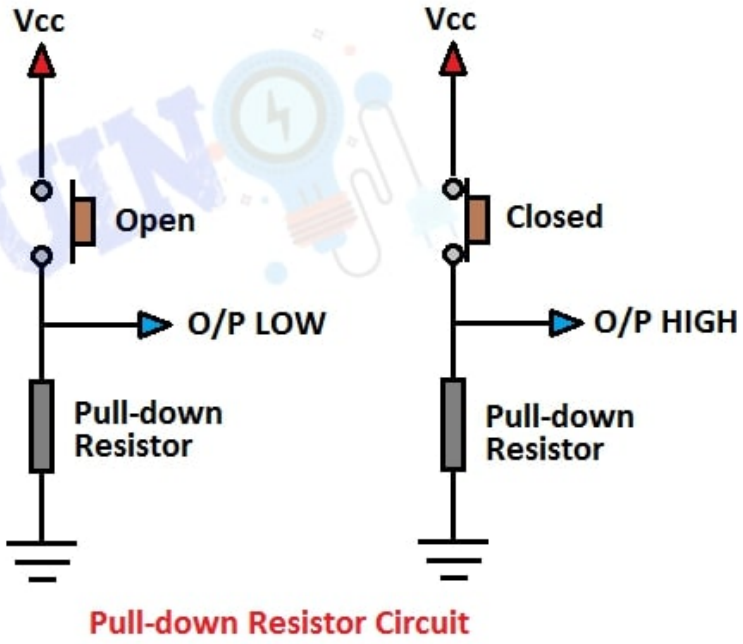
If we connect the push button switch directly to Raspberry Pi to get digital input, It means switch one pin is connected to Ground or 5v Vcc and another pin connected to Raspberry Pi digital pin. In this case, the Raspberry Pi is read **unstable input** from the push button.

So, we need to connect a “pull-up” or “pull-down” resistors circuit to **stabilizes the input**, when using the switch.

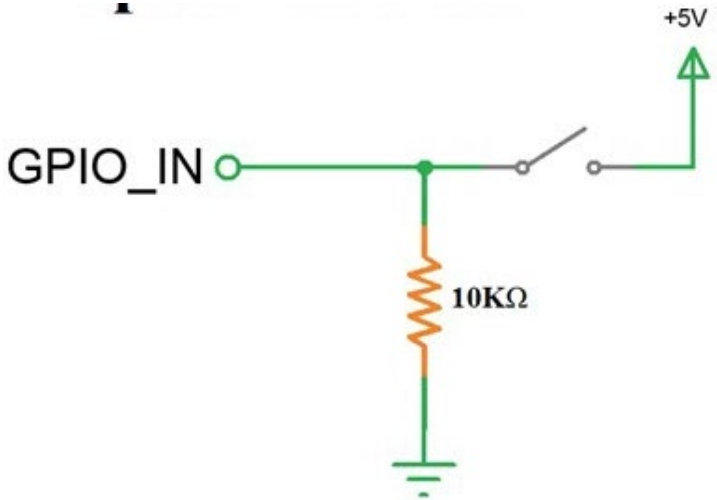
If the push button one pin is connected to the Vcc through a resistor and the other pin is connected to the ground, this circuit known as the **pull-up resistor circuit**. In this case, the push button output is **High (1)** when the button is **open**, and the output of the push button is **Low (0)** when the **button is pressed.**

Pull-Up and Pull-Down Resistor

Pull - down Resistor



Pull-down Resistor Circuit



If the push button one pin is connected to the ground through a resistor and the other pin is connected to the Vcc, this circuit known as the pull-down resistor circuit. In this case, the push button output is Low(0) when the button is open, and the output of the push button is High(1) when the button is pressed.

The Raspberry pi pico already has internal pull-up and pull-down resistors built-in which are automatically triggered when a digital input is read by the Raspberry pi pico meaning that we don't need to add any extra resistors to our circuit.

When to use Pull-Up and Pull-Down Resistor?

When should and should NOT we use a pull-down/pull-up resistor for an input pin?

- If the sensor has either closed (connected to **VCC** or **GND**) or open (NOT connected to anything) states, you need a pull-up or pull-down resistor to make these states become two states: **LOW** and **HIGH**. For example, push-button, switch, magnetic contact switch (door sensor)...
- If the sensor has two defined voltage levels (**LOW** and **HIGH**), you do NOT need a pull-up or pull-down resistor. For example, motion sensor, touch sensor ...

Trouble-shooting Pull-Up and Pull-Down Resistor

There are two common troubles that beginners usually get into:

1. Floating Input Problem

- ✓ **Symptom:** The reading value from the input pin is not matched with the button's pressing state
- ✓ **Cause :** Input pin is NOT used pull-up and pull-down resistors.
- ✓ **Solution:** Use Pull-up and Pull-down resistor.

2. Chattering Phenomenon

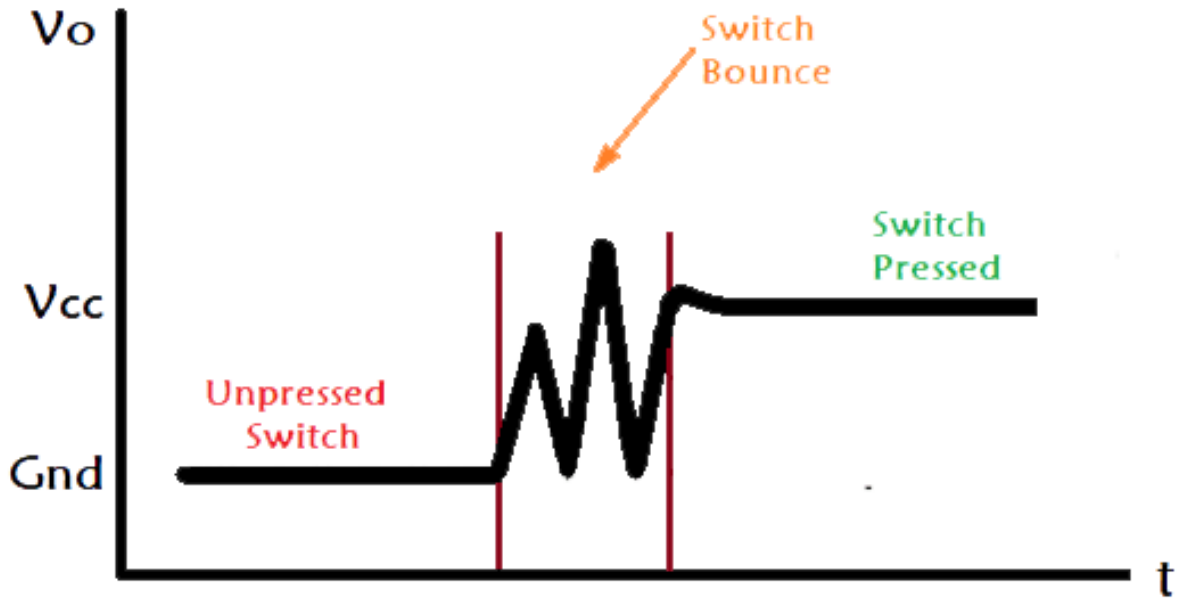
It should be considered in only some application that needs to detect exactly number of the pressing.

- ✓ **Symptom:** Button is pressed once, but Raspberry Pi code detects several times.
- ✓ **Cause :** Due to mechanical and physical issues, the state of button (or switch) is quickly toggled between **LOW** and **HIGH** several times.
- ✓ **Solution:** Debounce

Trouble-shooting Pull-Up and Pull-Down Resistor

What is Switch Bouncing?

When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply. But they don't connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made. The same thing happens while releasing the button. This results the false triggering or multiple triggering like the button is pressed multiple times. Its like falling a bouncing ball from a height and it keeps bouncing on the surface, until it comes at rest.



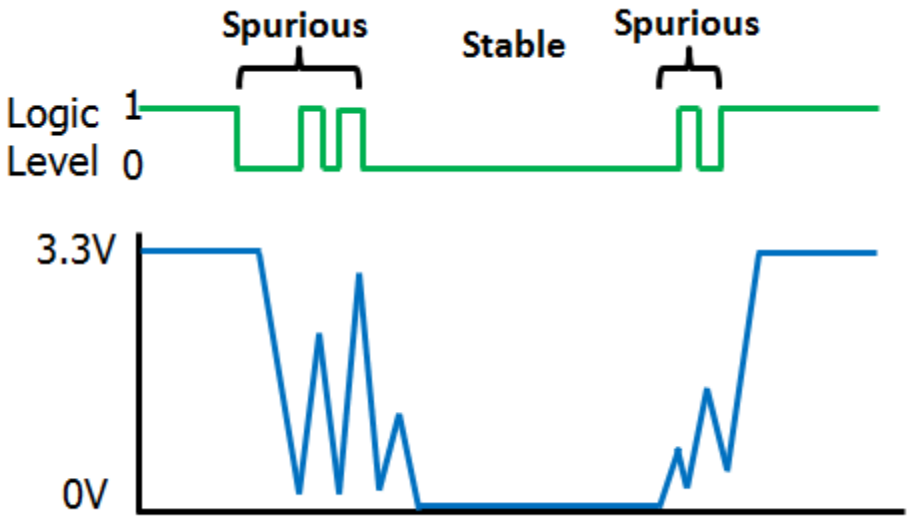
Switch bouncing is the non-ideal behavior of any switch which generates multiple transitions of a single input. Switch bouncing is not a major problem when we deal with the power circuits, but it cause problems while we are dealing with the logic or digital circuits. Hence, to remove the bouncing from the circuit Switch Debouncing Circuit is used.

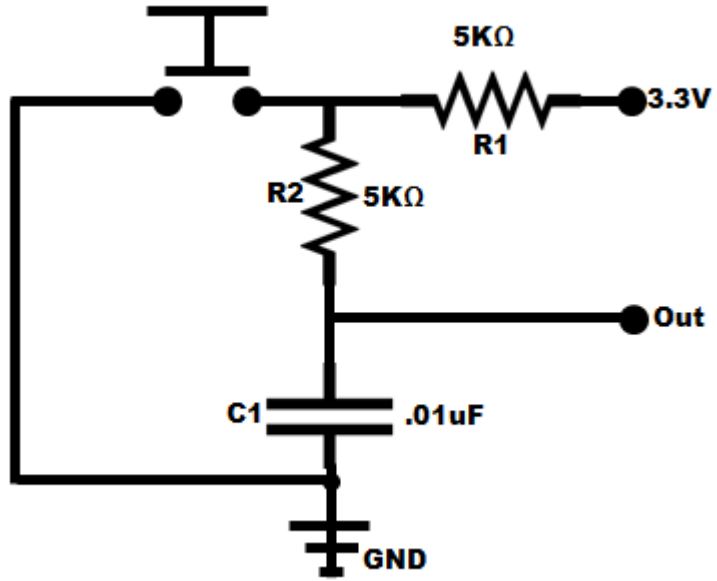
Trouble-shooting Pull-Up and Pull-Down Resistor

Switch De-Bouncing

There are three commonly used methods to prevent the circuit from switch bouncing.

- **Hardware Debouncing** (In the hardware debouncing technique we use an S-R flip flop to prevent the circuit from switch bounces. This is the best debouncing method among all.)
- **RC Debouncing** (The R-C is defined by its name only, the circuit used a RC network for the protection from switch bounce. The capacitor in the circuit filter the instant changes in the switching signal. When the switch is in open state the voltage across the capacitor remain zero. Initially, when the switch is open the capacitor charge through the R1 and R2 resistor.)
- **Switch Debouncing IC** (Some of the debouncing ICs are MAX6816, MC14490, and LS118.)



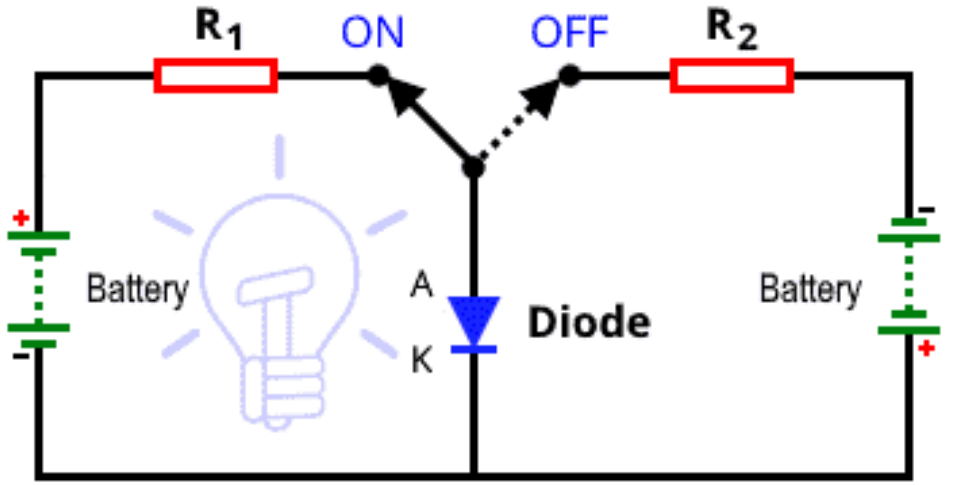


Electrical & Electronic Switches

- Electrical and mostly electronic switches are **solid-state devices** based on semiconductor materials with **fast response, accurate operation and small in size** as compared to mechanical and electromechanical switches. The solid state switches are based on the basic components such as diodes, SCR, MOSFET, GTO, IGBT, transistors, and relays etc.
- Electronic switches have **no physical contacts or moving parts** and can be automatically operated by electric signals or programmed circuits like microcontroller or microprocessor. They are precise in operation for stability and reliability of the system without noise of switching operation.
- They are used in many modern applications such as variable frequency drive (VFD) drives for motors, HVAC & in industrial, automation, automotive, aerospace, robotic and many more commercial applications.

Electrical & Electronic Switches

Diode as a Switch



- A basic PN junction diode can be used as a switch. The diode in forward bias acts as a closed switch while it acts as an open switch in case of reverse bias.

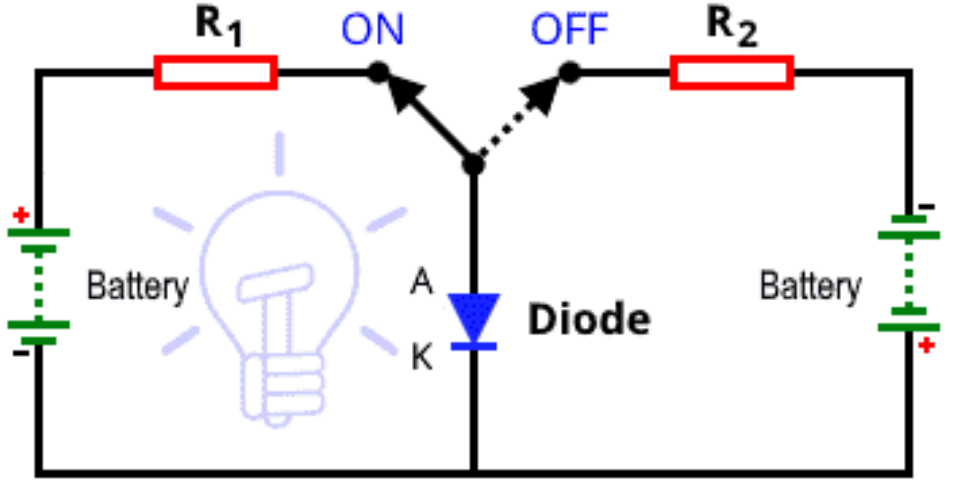
switching operations of a diode.

- Forward Bias: If positive signal applied to the Anode terminal and cathode is negative, the diode is forward biased, hence it acts as a closed switch.
- Reverse Bias: If the negative signal applied to the Anode and cathode is positive, the diode is in reverse biased, thus it acts as an open switch.

Electrical & Electronic Switches

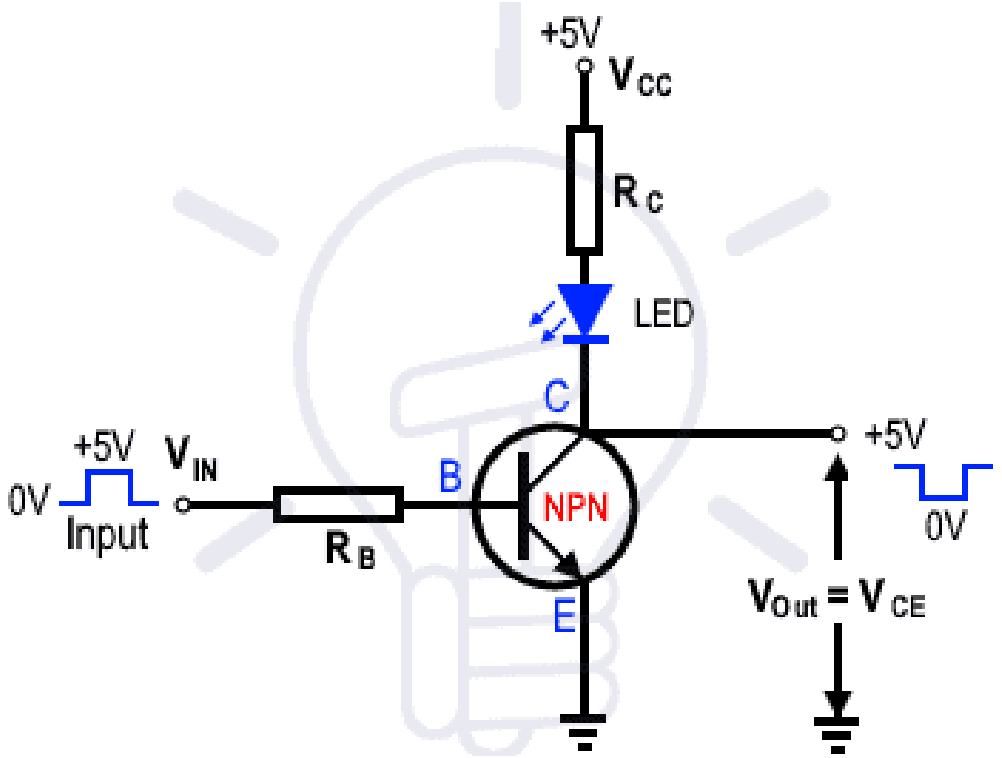
BJT Transistors as a switch:

- Bipolar Junction Transistors (BJT) can be used as normal switches as they are able to block or pass the flowing of electric current in different modes of operations.



NPN Transistor as a Switch

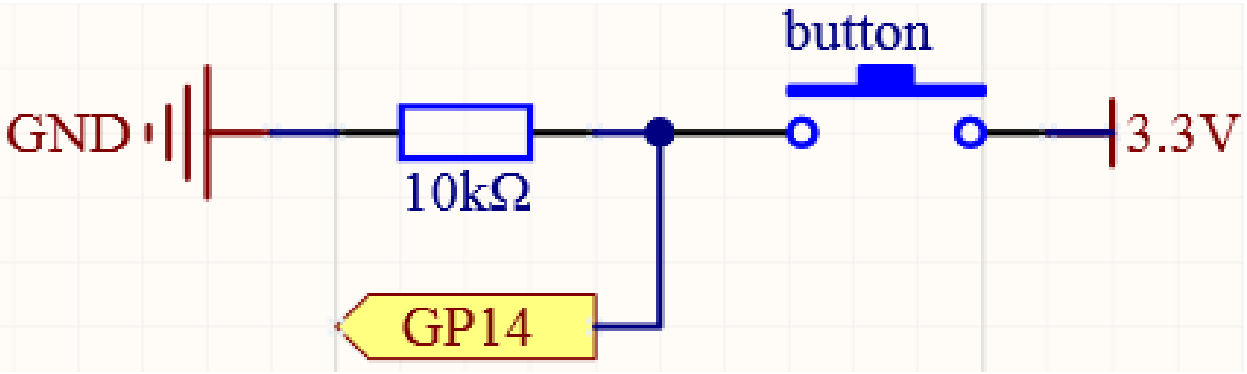
- When the input to the inverter is high "+5V/+3.3V", the NPN transistor is saturated and its output is low "≈0V". When the input to the inverter is low, the transistor is cut-off and its output is high.
- **In the saturation region, it is "ON" like a closed switch**
- **In the cut-off region, it is "OFF" like an open switch.**



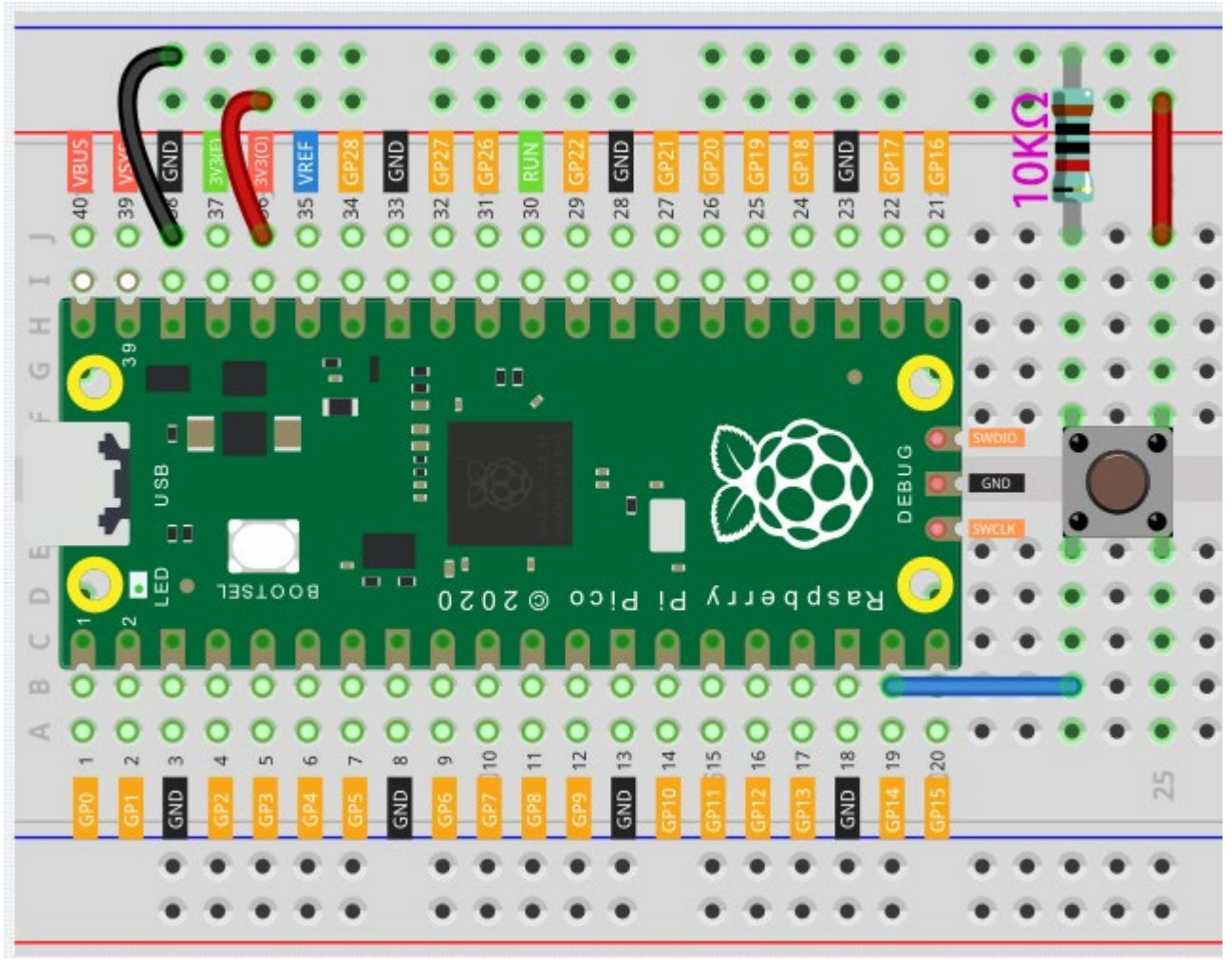
Reading Button Value

Buttons require **pull-up resistors or pull-down resistors**. If there is no pull-up or pull-down resistor, the main controller may receive a 'noisy' signal which can trigger even when you're not pushing the button.

Pull-down Working Mode



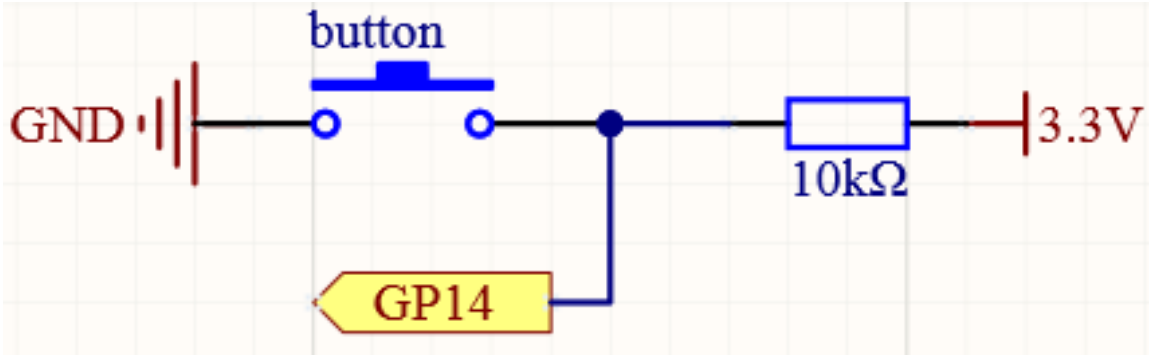

```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 1:
        print("You pressed the button!")
        utime.sleep(1)
```

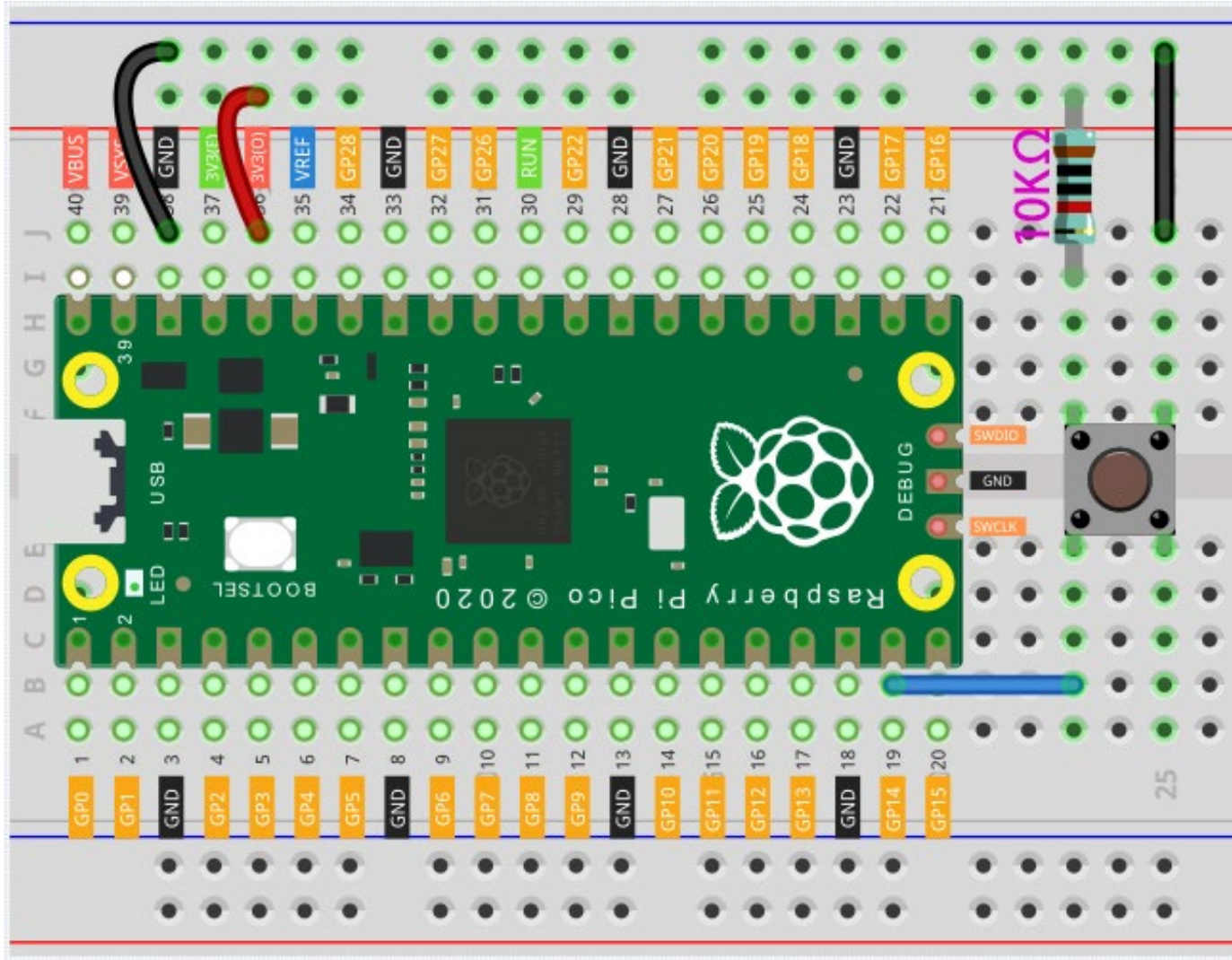


Reading Button Value

Buttons require **pull-up resistors or pull-down resistors**. If there is no pull-up or pull-down resistor, the main controller may receive a 'noisy' (floating) signal which can trigger even when you're not pushing the button.

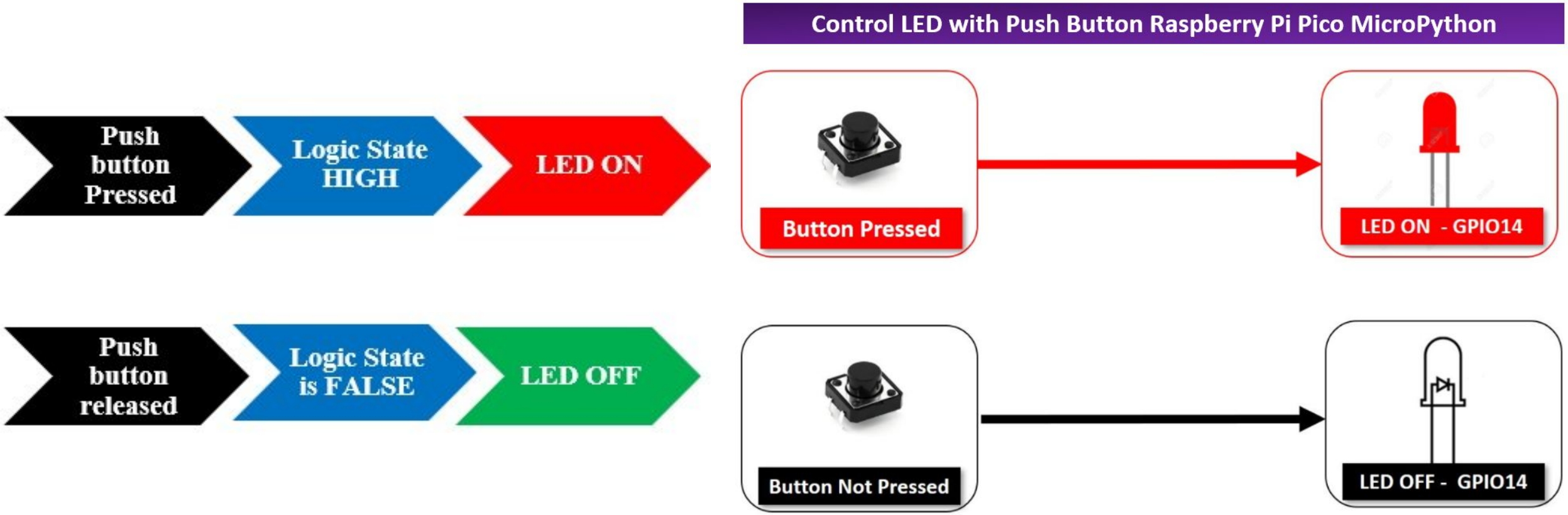
Pull-up Working Mode






```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 0:
        # When the button is pressed, GPIO will be connected to GND.
        print("You pressed the button!")
        utime.sleep(1)
```

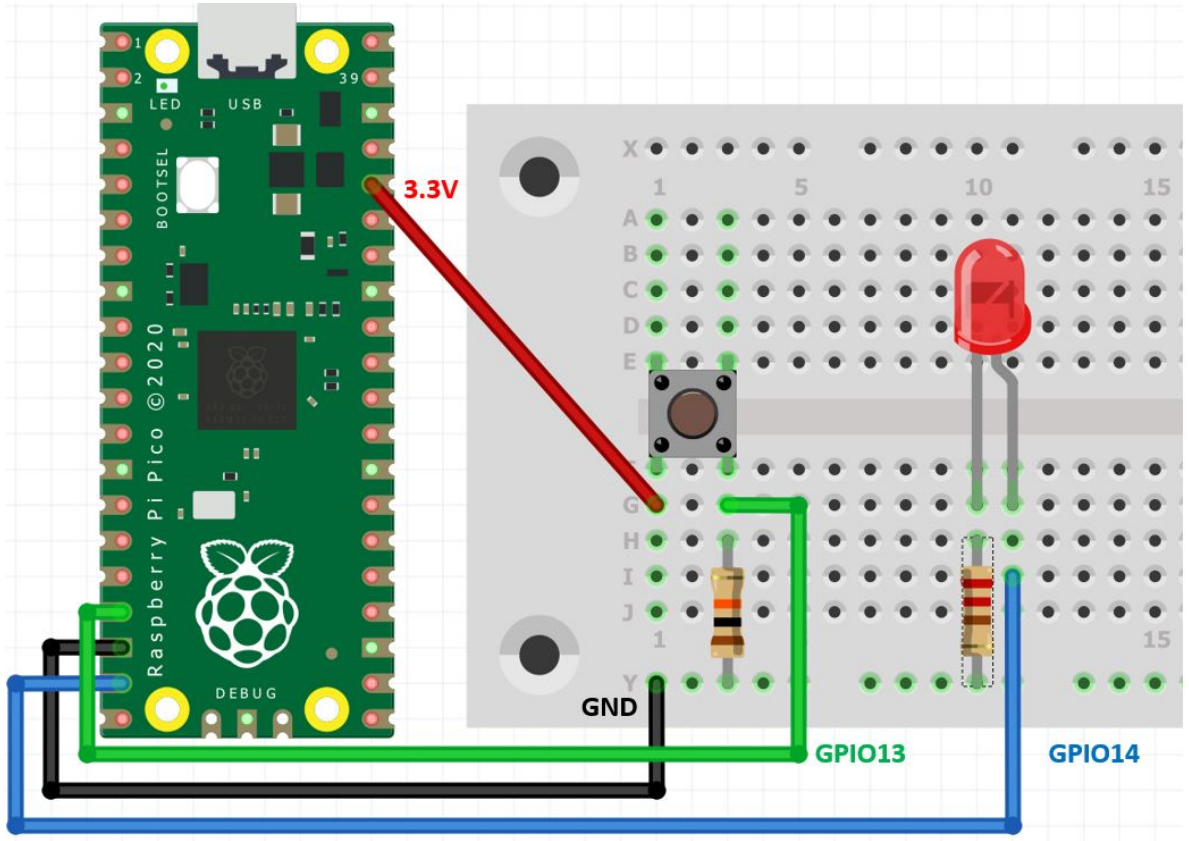
Interface Push Button with Raspberry Pi Pico and Control LED



Objective:

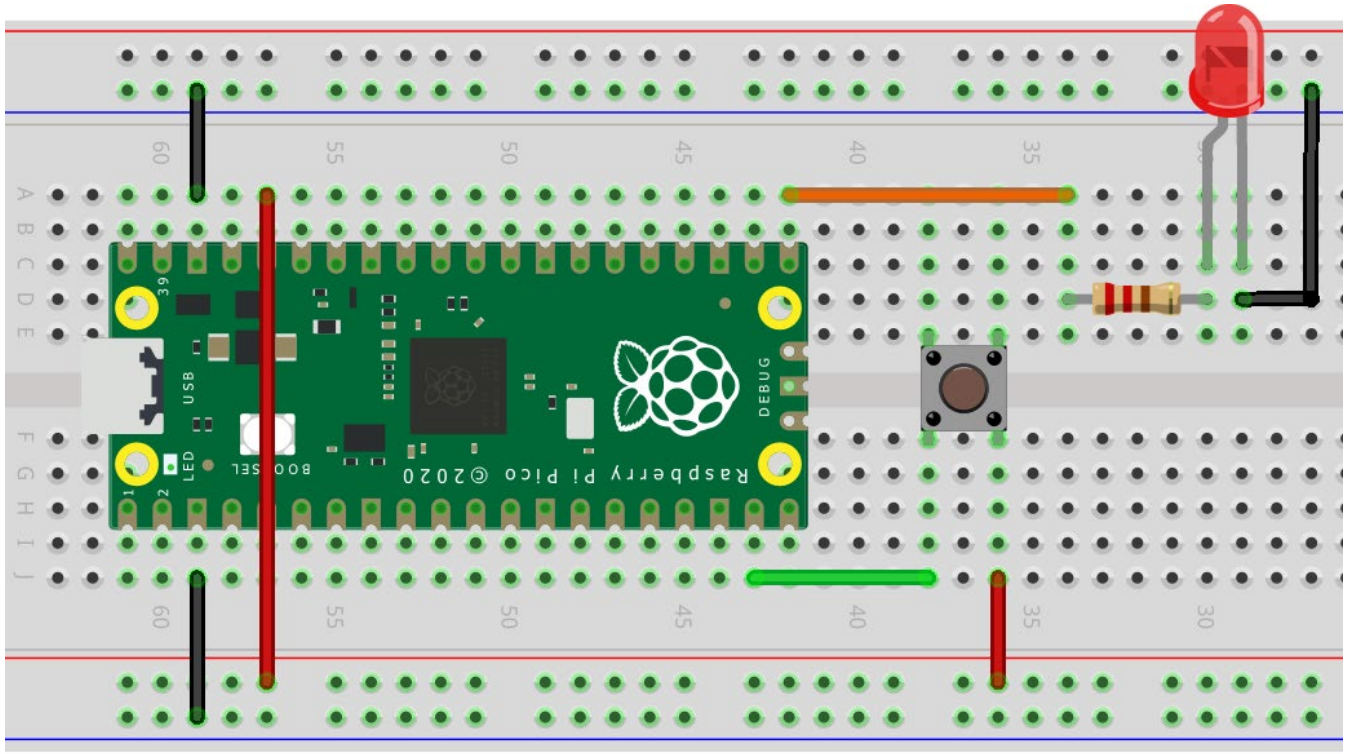
If you press the push button, the LED will glow and if you leave the push button, a LED will remain off.

Interface Push Button with Raspberry Pi Pico and Control LED



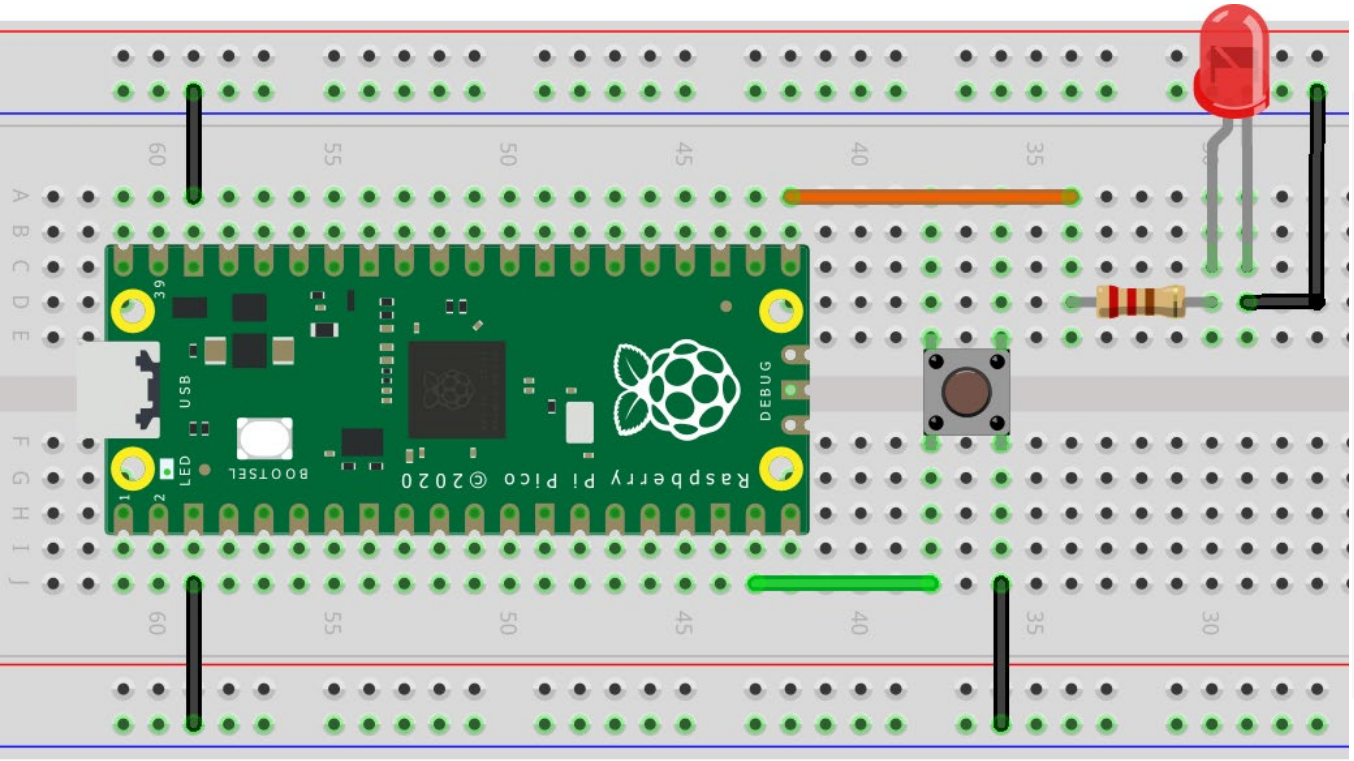

```
from machine import Pin
from time import sleep


led = Pin(14, Pin.OUT)     # 14 number in is Output
push_button = Pin(13, Pin.IN)  # 13 number pin is input

while True:

  logic_state = push_button.value()
  if logic_state == True:      # if push_button pressed
      led.value(1)              # led will turn ON
  else:                        # if push_button not pressed
      led.value(0)              # led will turn OFF
```

With the internal pulldown resistor of the digital input




```
from machine import Pin

pin_button = Pin(14, mode=Pin.IN, pull=Pin.PULL_DOWN)
pin_led    = Pin(16, mode=Pin.OUT)

while True:
    if pin_button.value() == 1:
        pin_led.on()
    else:
        pin_led.off()
```

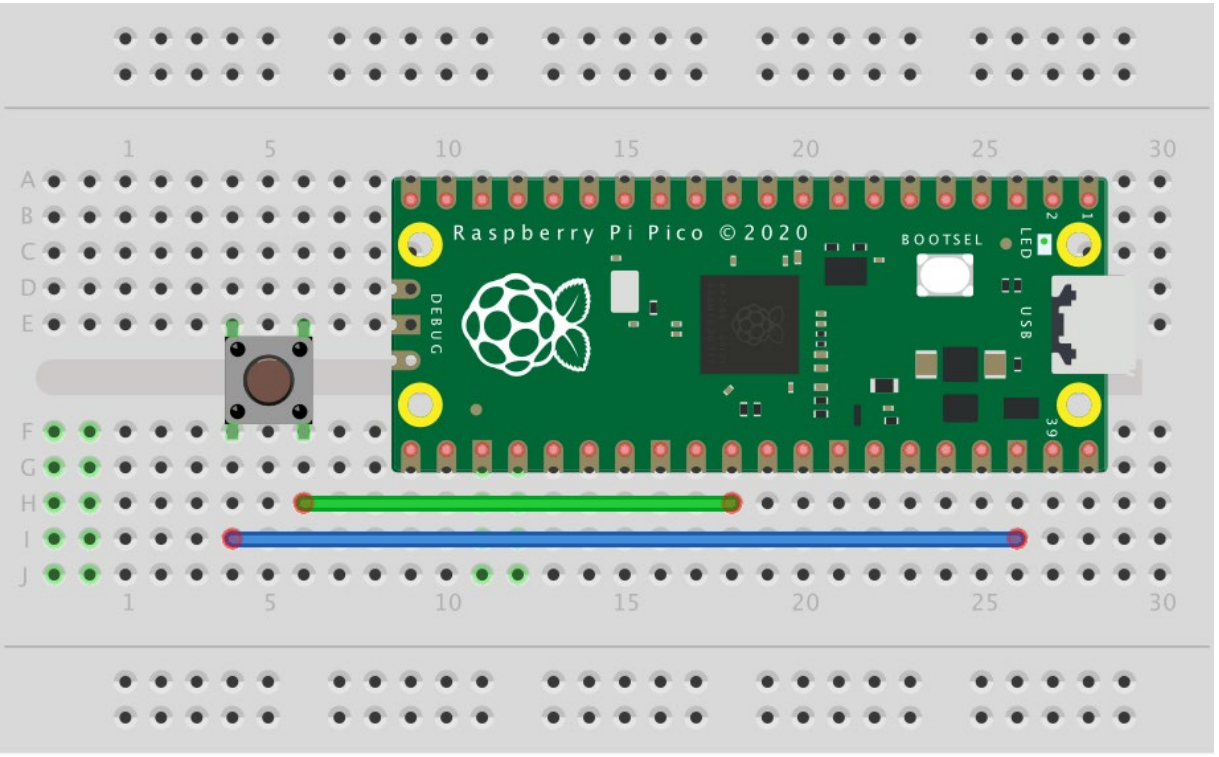
With the internal pullup resistor of the digital input




```
from machine import Pin

pin_button = Pin(14, mode=Pin.IN, pull=Pin.PULL_UP)
pin_led    = Pin(16, mode=Pin.OUT)

while True:
    if not pin_button.value():
        pin_led.on()
    else:
        pin_led.off()
```

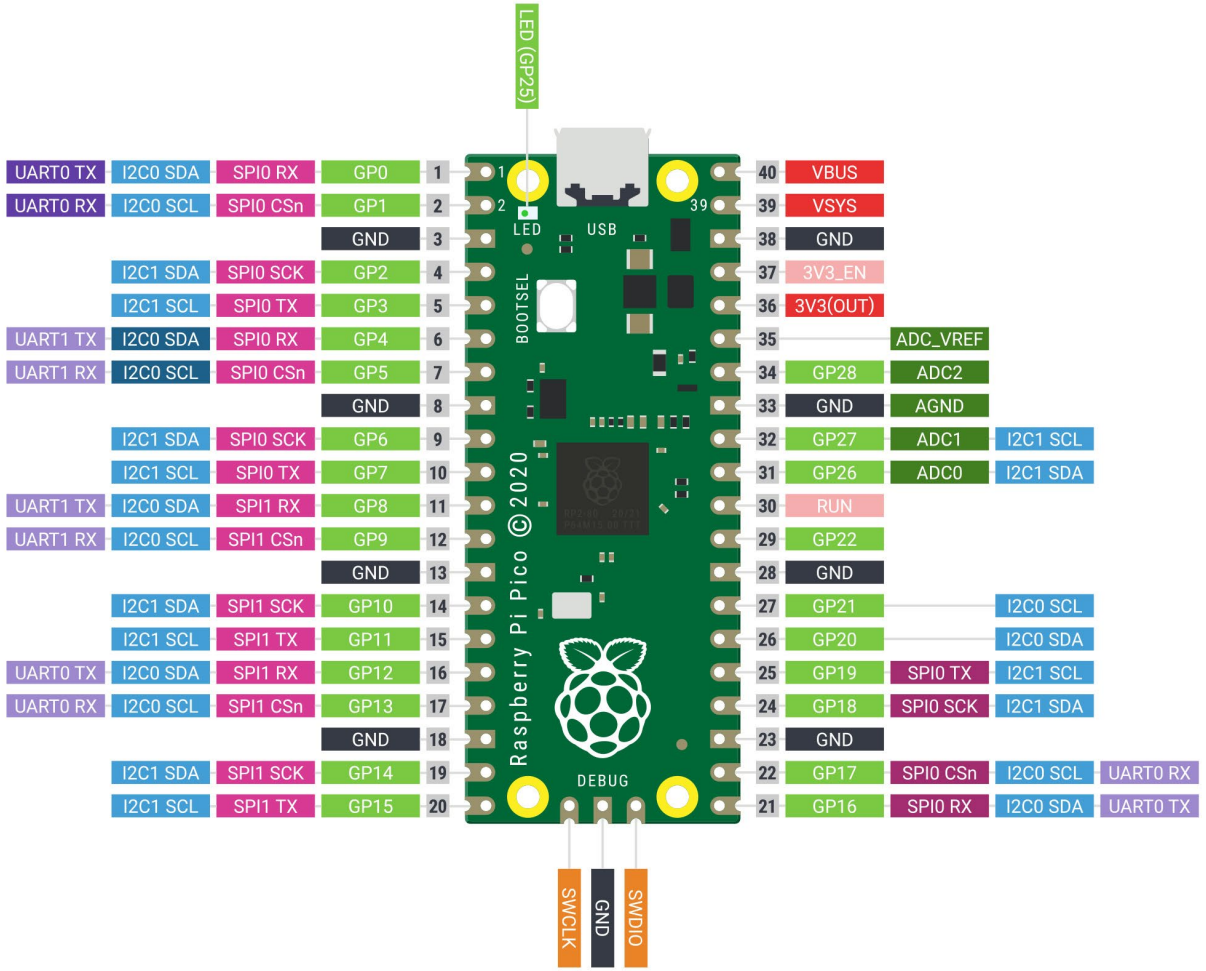
Push Button as reset button in Raspberry Pi Pico

When you press and hold the BOOTSEL button and connect your Pico to your computer, it mounts as a mass storage volume. You can then just drag and drop a UF2 file onto the board. Sometimes it is not convenient to keep unplugging the micro USB cable every time you want to upload a UF2 onto the Pico. Having a reset button on your Raspberry Pi Pico resolves this problem.



- Place the Pico on the breadboard.
- Place the button on the breadboard.
- Connect a jumper from one button pin to a GND pin on the Pico, we used pin 38.
- Connect a jumper wire from the other button pin to RUN, pin 30 on the Pico.

Push Button as reset button in Raspberry Pi Pico



Now, rather than unplugging and replugging the USB cable when you want to load code:

- Push and hold the reset button.
- Push the BOOTSEL button.
- Release the reset button.
- Release the BOOTSEL button.

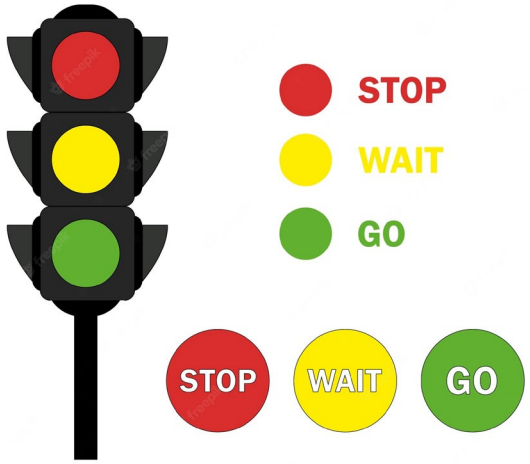
If your board is in BOOTSEL mode and you want to restart code already loaded:

- Briefly push the reset button.

If we press the button on the breadboard it will pull the RUN pin down to 0V, forcing the Pico to reset. Press the Stop button to reconnect to the MicroPython shell. If your code ever locks up then this trick will help get you back to work.

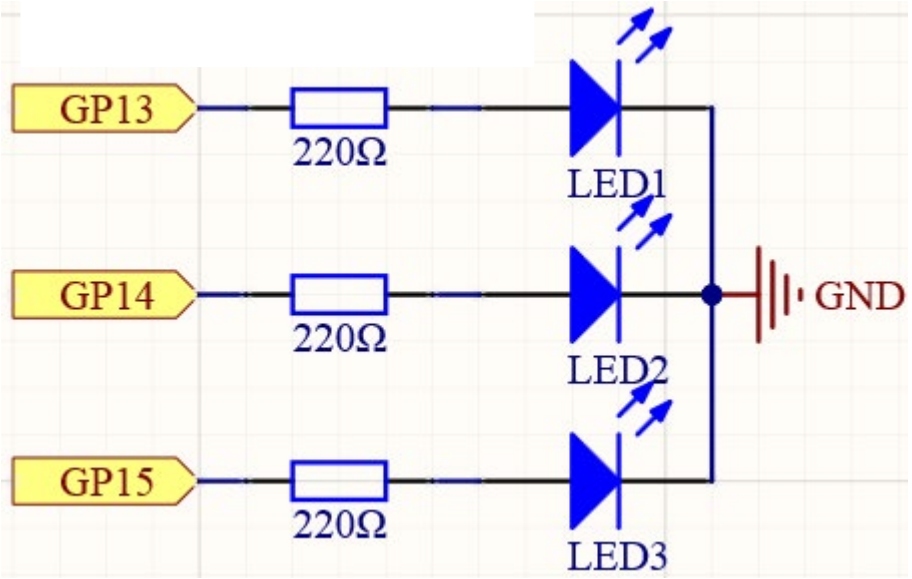
A basic three-light traffic light system

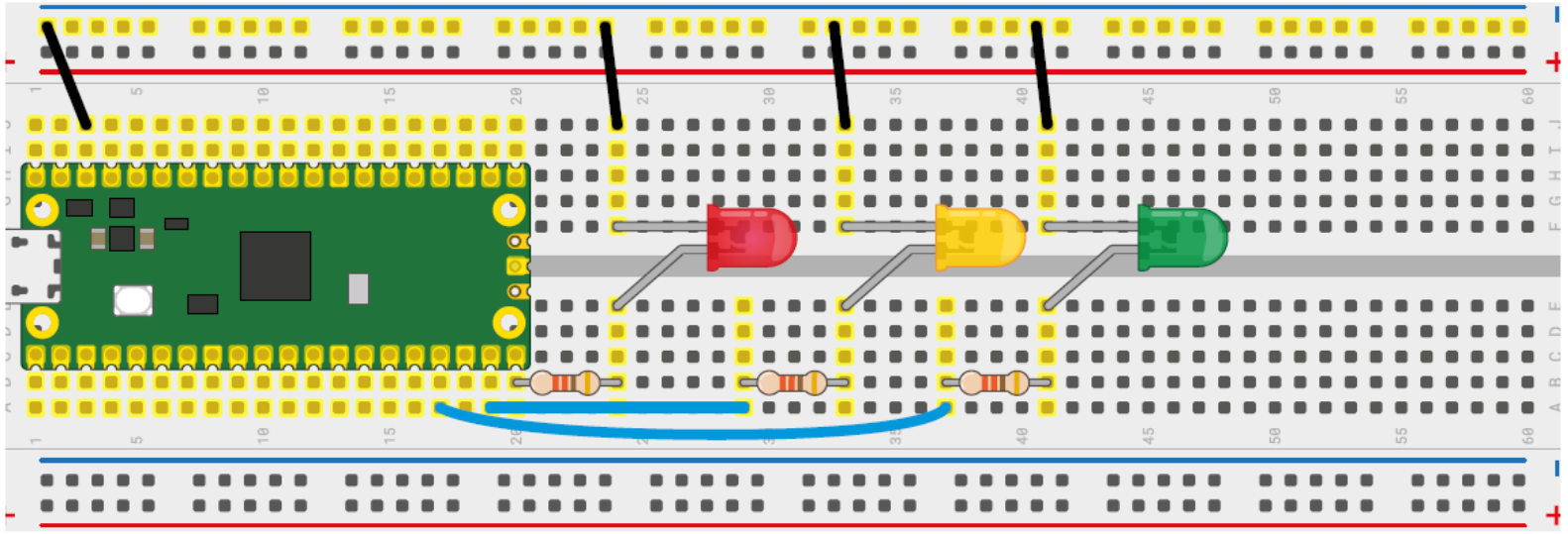
Traffic lights are used to direct traffic operation and are generally composed of red, green, and yellow lights.



Step-by-step, turning the LEDs on and off.

- a red light to tell the traffic to stop
- an amber or yellow light to tell the traffic the light is about to change,
- a green LED to tell the traffic it can go again





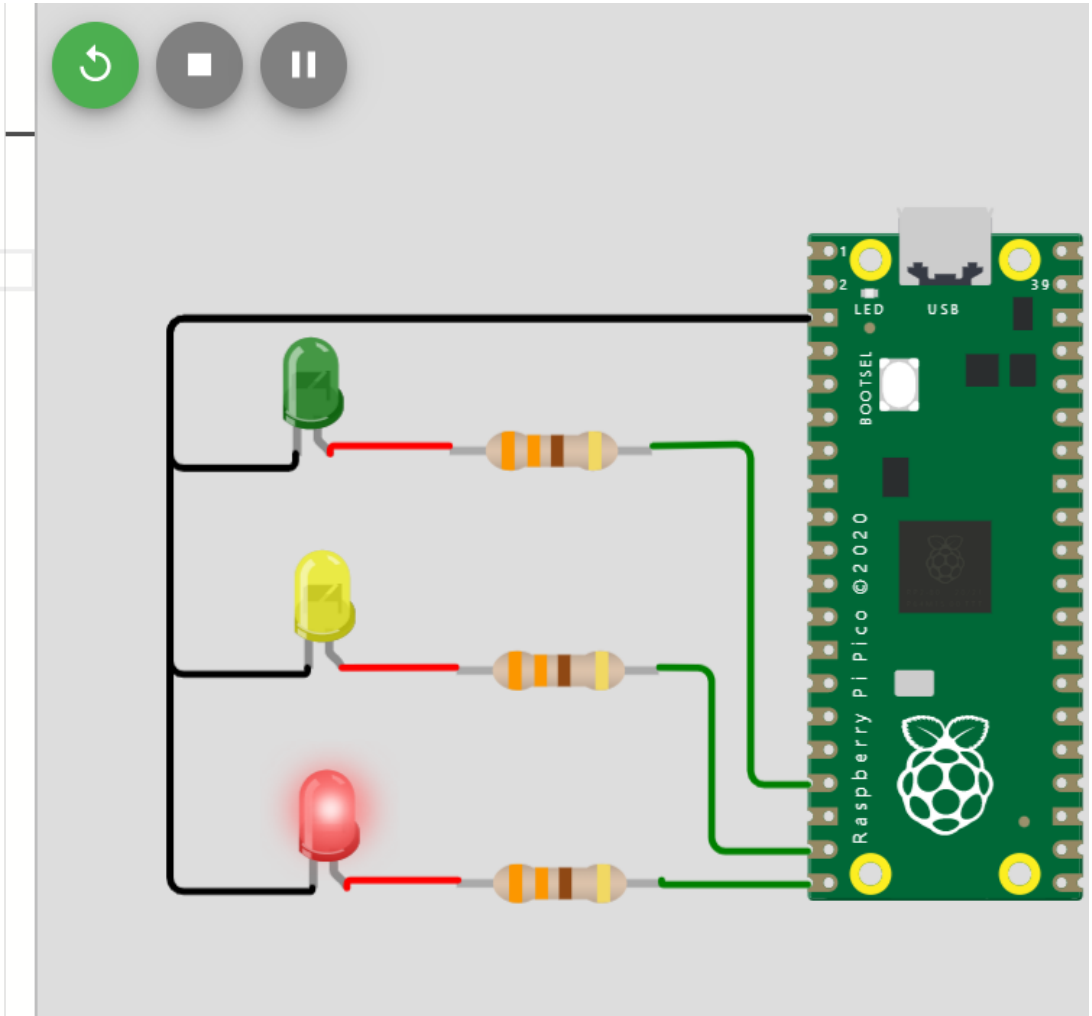
A simple traffic light Control system without proper control

Mini pedestrian crossing system

Objective:

The traffic light will switch in the order of red for 5 seconds, yellow for 2 seconds, green for 5 seconds, and yellow for 2 seconds.

```
1   from machine import Pin
2   from utime import sleep
3
4   led_red = Pin(15, Pin.OUT)
5   led_yellow = Pin(14, Pin.OUT)
6   led_green = Pin(13, Pin.OUT)
7
8   while True:
9
10      led_red.value(1)
11      sleep(5)
12      led_red.value(0)
13
14      led_yellow.value(1)
15      sleep(2)
16      led_yellow.value(0)
17
18      led_green.value(1)
19      sleep(5)
20      led_green.value(0)
21
22      led_yellow.value(1)
23      sleep(2)
24      led_yellow.value(0)
```



Puffin Crossing (Mini pedestrian crossing system with proper control)

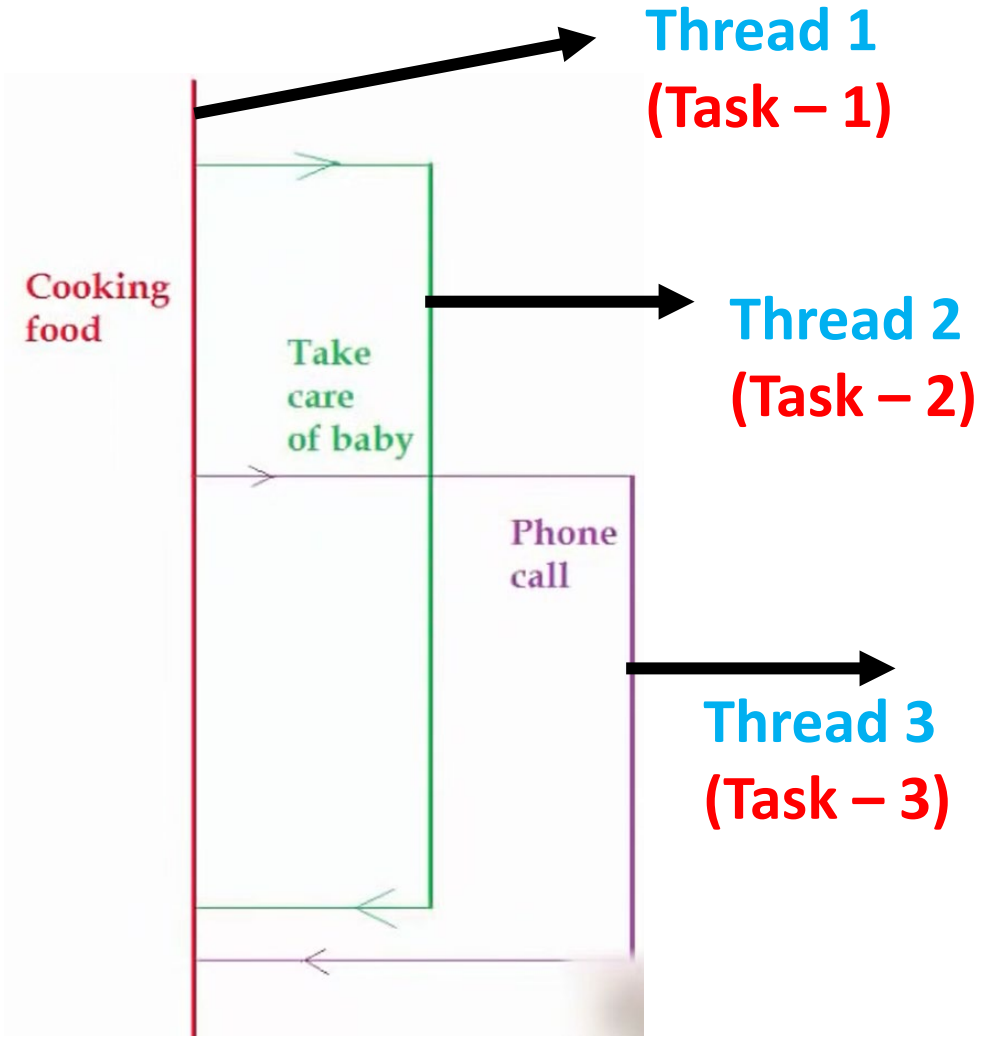
Through a push-button switch, the pedestrian can ask the lights to let them cross the road; and a buzzer, so the pedestrian knows when it's their turn to cross. If we (pedestrians) press the button, the red LED will be extended to 15 seconds, which will give us more time to cross the road.

Puffin Crossing (Mini pedestrian crossing system with proper control)

In puffin crossing our program should able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: **_thread.**

Multithreading Concept

Busy MOM



Multiple action without threading

```
1   import time
2
3
4   def calc_square(numbers):
5       print("calculate square numbers")
6       for n in numbers:
7           time.sleep(1)
8           print('square:',n*n)
9
10  def calc_cube(numbers):
11      print("calculate cube of numbers")
12      for n in numbers:
13          time.sleep(1)
14          print('cube:',n*n*n)
15
16  arr = [2,3,8,9]
17
18  t = time.time()
19
20  calc_square(arr)
21  calc_cube(arr)
22
23  print("done in : ",time.time()-t)
24  print("Hah... I am done with all my work now!")
```

For a given list of numbers print square and cube of every numbers

For example:

Input: [2,3,8,9]

Output: square list - [4, 9, 64, 81]
cube list - [8, 27, 512, 729]

Execute this in THONNY and check the processing time

```
calculate square numbers
square: 4
square: 9
square: 64
square: 81
calculate cube of numbers
cube: 8
cube: 27
cube: 512
cube: 729
done in :  8.065548181533813
Hah... I am done with all my work now!
```

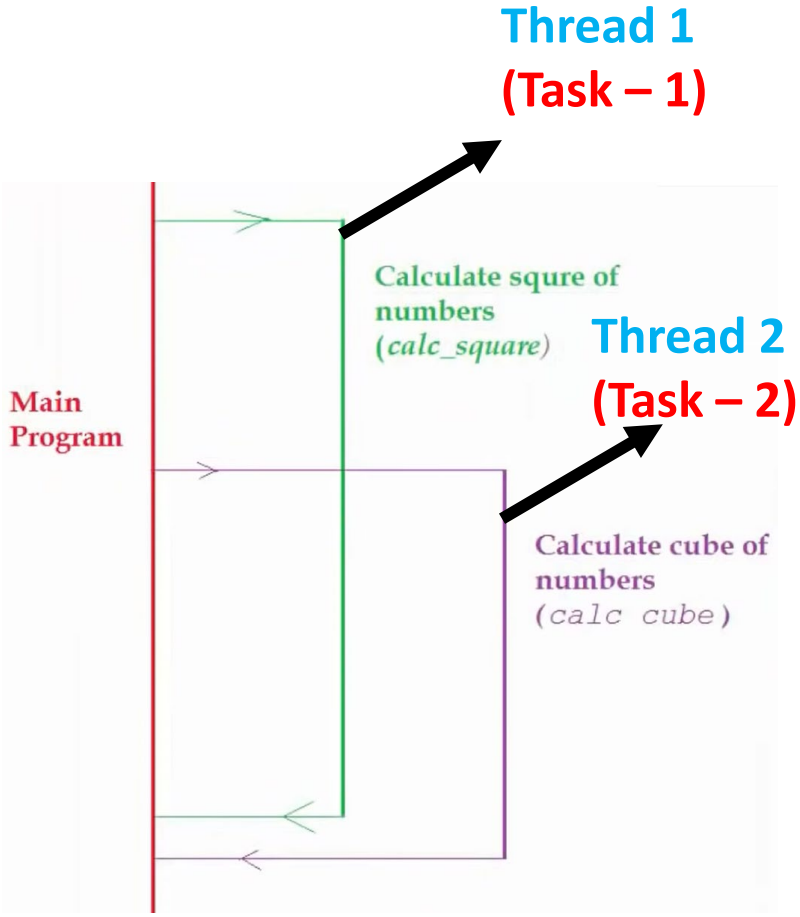
Multiple action with threading (MULTITHREADING)

```
1  import time
2  import threading
3
4  def calc_square(numbers):
5      print("calculate square numbers")
6      for n in numbers:
7          time.sleep(1)
8          print('square:',n*n)
9
10 def calc_cube(numbers):
11     print("calculate cube of numbers")
12     for n in numbers:
13         time.sleep(1)
14         print('cube:',n*n*n)
15
16 arr = [2,3,8,9]
17
18 t = time.time()
19
20 t1= threading.Thread(target=calc_square, args=(arr,))
21 t2= threading.Thread(target=calc_cube, args=(arr,))
22
23 t1.start()
24 t2.start()
25
26 t1.join()
27 t2.join()
28
29 print("done in : ",time.time()-t)
30 print("Hah... I am done with all my work now!")
```

```
calculate square numbers
calculate cube of numbers
square:cube:  48

cube:square:  279

square:cube:  64512

square:cube:  81729

done in :  4.064005613327026
Hah... I am done with all my work
now!
```

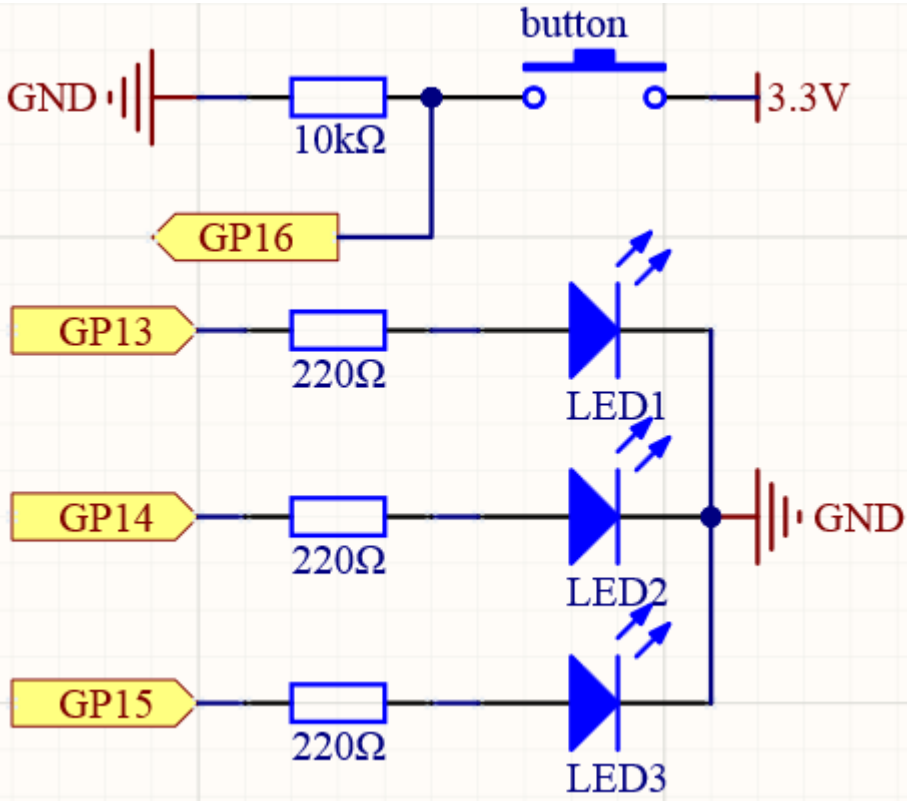
The RP2040 microcontroller which powers our Pico has two processing cores

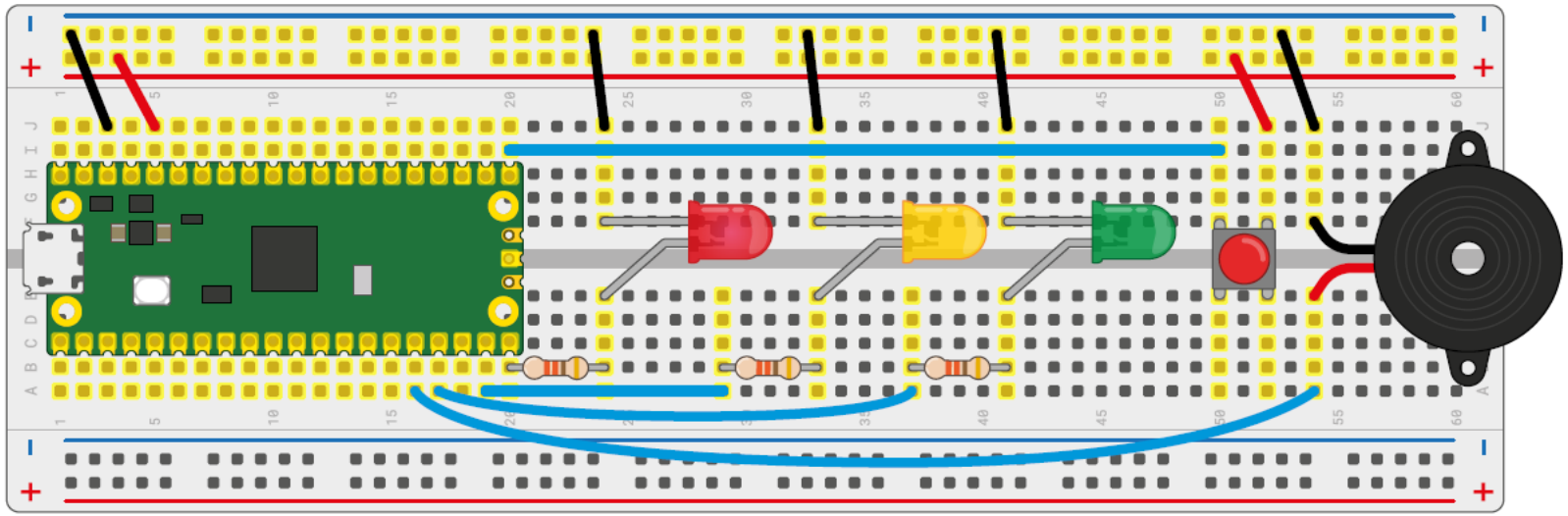
Execute this MULTITHREADING in THONNY and check the processing time



Puffin Crossing (Mini pedestrian crossing system with proper control)

In puffin crossing our program should able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: **_thread.**





A simple traffic light Control system with proper control

- **controls the lights, as the main thread**
- New thread to pass information back to the main thread – and you can do this using **global variables.** The variables working with prior to this are known as **local variables**, and only work in one section of program; a global variable works everywhere, meaning **one thread can change the value and another can check to see if it has been changed.**

Puffin Crossing (Mini pedestrian crossing system with proper control)

WOKWI SAVE SHARE

main.py diagram.json

```
from machine import Pin
from utime import sleep
import _thread

led_red = Pin(15, Pin.OUT)
led_yellow = Pin(14, Pin.OUT)
led_green = Pin(13, Pin.OUT)
button = Pin(16, Pin.IN, Pin.PULL_UP)
buzzer = Pin(12, Pin.OUT)

global button_status
button_status = 0

def button_thread():
    global button_status
    while True:
        if button.value() == 1:
            button_status = 1

_thread.start_new_thread(button_thread, ())

while True:
    if button_status == 1:
        led_red.value(1)
        for i in range(10):
            buzzer.value(1)
            sleep(0.5)
            buzzer.value(0)
            sleep(0.5)
    global button_status
    button_status = 0

    led_red.value(1)
    sleep(5)
    led_red.value(0)

    led_yellow.value(1)
    sleep(2)
    led_yellow.value(0)

    led_green.value(1)
    sleep(5)
    led_green.value(0)

    led_yellow.value(1)
    sleep(2)
    led_yellow.value(0)
```

Simulation