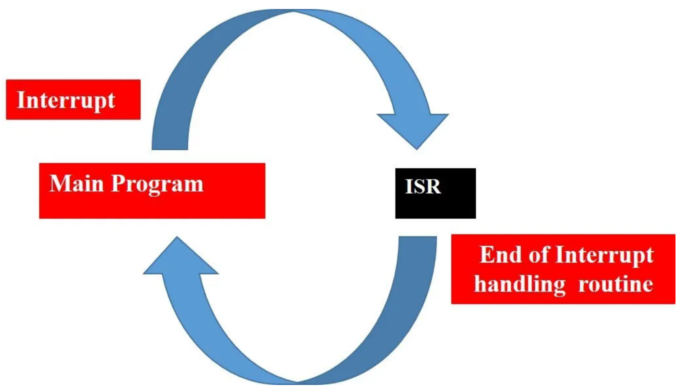
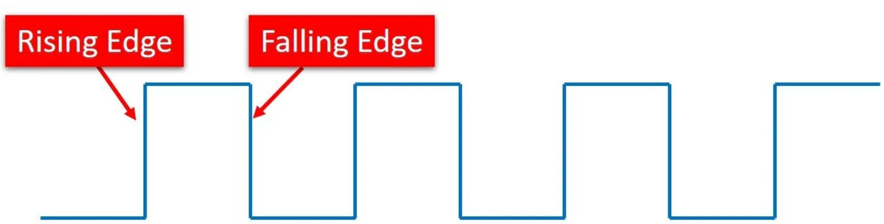
Introduction. Interrupts are used to handle events that do not happen during the sequential execution of a program. Sometimes there are a few tasks that only execute when a special event occurs such as an external trigger signal to the digital input pin of a microcontroller. Then the hardware module (eg. GPIO) may cause an external or hardware interrupt.

When an interrupt occurs, the processor stops the execution of the main program and a function is called upon known as *ISR* or the *Interrupt Service Routine*. The processor then temporarily works on a different task (*ISR*) and then gets back to the main program after the handling routine has ended.

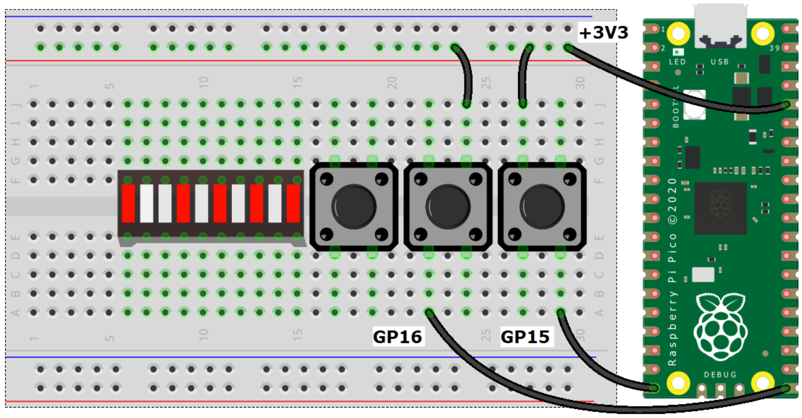


Raspberry Pi Pico can have all GPIO pins configured as an external interrupt pin on the following four changes on the state of GPIO pins:

- level high (`trigger=Pin.IRQ_LOW_LEVEL`)
- level low (`trigger=Pin.IRQ_HIGH_LEVEL`)
- positive edge (`trigger=Pin.IRQ_RISING`)
- negative edge (`trigger=Pin.IRQ_FALLING`)



Task.1. Using either Raspberry Pi Pico Expander Board or breadboard, connect the following circuit.



Code example:

```
from machine import Pin
from utime import sleep

BTN_ON_PIN = 15
BTN_TOGGLE_PIN = 16

led_state = False
led_toggle = False

led_builtin = Pin(25,Pin.OUT)
btn_on = Pin(BTN_ON_PIN,Pin.IN,Pin.PULL_DOWN)
btn_toggle = Pin(BTN_TOGGLE_PIN,Pin.IN,Pin.PULL_DOWN)

def on_interrupt(Pin):
    global led_state
    led_state = not(led_state)
    print("LED on/off int")
    sleep(0.02)
```

```
def toggle_interrupt(Pin):
    global led_toggle
    led_toggle = not(led_toggle)
    print("LED toggle int")
    sleep(0.02)

btn_on.irq(trigger=Pin.IRQ_RISING, handler=on_interrupt)
btn_toggle.irq(trigger=Pin.IRQ_RISING,handler=toggle_interrupt)

while True:
    if led_state:
        if led_toggle:
            led_builtin.toggle()
        else:
            led_builtin.value(1)
    else:
        led_builtin.value(0)
    sleep(0.2)
```

Task.2. Make the GPIO25 LED blink without using the *utime* module. The required blinking period equals 1s with a duty cycle of 50%.

```
from machine import Pin,Timer

#on/off time 500ms
DELAY = 500

led_builtin = Pin(25,Pin.OUT)

def led_control(timer):
    led_builtin.toggle()

#timer initialization
timer = Timer(period=DELAY,mode=Timer.PERIODIC,
              callback=led_control)
```

Timers are more efficient than using the sleep function because they are not blocking functions as opposed to the latter. A blocking function stops the program from performing any task until the previous one is completed. The `Timer()` function has three arguments, namely:

- *period* - the period of the interrupt signal in milliseconds;
- *mode:*
  - `Timer.PERIODIC` - callback is called after every period;
  - `Timer.ONE_SHOT` - callback is called once.
- *callback* - callback function is called whenever a timer is triggered.

Different implementation:

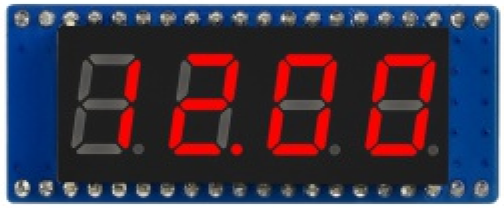
```
from machine import Pin,Timer

led_builtin = Pin(25,Pin.OUT)

DELAY = 500

# virtual timer
timer = Timer(-1)

#timer initialization
timer.init(period = DELAY,mode = Timer.PERIODIC,
           callback = lambda led:led_builtin.toggle())
```

Task.3. Use a *`Timer`* function to eliminate the bouncing effect.

```
from machine import Pin,Timer
from utime import sleep

BTN_DELAY = 200
MAIN_DELAY = 500

BTN_ON_PIN = 15
BTN_TOGGLE_PIN = 16
```

```
led_state = False
led_toggle = False
led_builtin = Pin(25,Pin.OUT)
btn_on = Pin(BTN_ON_PIN,Pin.IN,Pin.PULL_DOWN)
btn_toggle = Pin(BTN_TOGGLE_PIN,Pin.IN,Pin.PULL_DOWN)

def on_interrupt(Pin):
    global led_state
    led_state = not(led_state)
    btn_on.irq(handler = None)
    timer = Timer(period = BTN_DELAY,mode = Timer.ONE_SHOT,
                  callback = lambda t:
                  btn_on.irq(handler=on_interrupt))
    print("LED on/off int")

def toggle_interrupt(Pin):
    global led_toggle
    led_toggle = not(led_toggle)
    btn_toggle.irq(handler = None)
    timer = Timer(period = BTN_DELAY,mode = Timer.ONE_SHOT,
                  callback = lambda t:
                  btn_toggle.irq(handler=toggle_interrupt))
    print("LED toggle int")

def main_loop(main_timer):
    if led_state:
        if led_toggle:
            led_builtin.toggle()
        else:
            led_builtin.value(1)
    else:
        led_builtin.value(0)

btn_on.irq(trigger=Pin.IRQ_RISING, handler=on_interrupt)
btn_toggle.irq(trigger=Pin.IRQ_RISING,
handler=toggle_interrupt)

main_timer = Timer(period = MAIN_DELAY,mode = Timer.PERIODIC,
                   callback = main_loop)
```

Task.4. Connect *Pico-8SEG-LED* module.



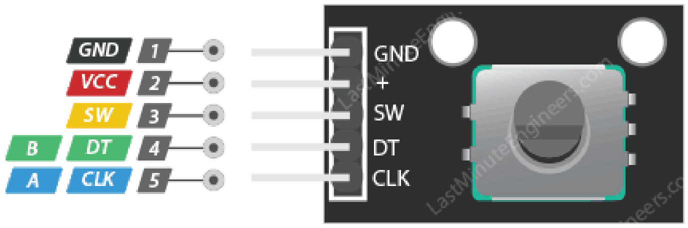
Code example:

```
from machine import Pin,SPI
from utime import sleep

RCLK = 9
# SPI pins
SCK = 10
MOSI = 11

# displays
LED_1 = 0xFE  # thousands 0x11111110
LED_2 = 0xFD  # hundreds  0x11111101
LED_3 = 0xFB  # tens      0x11111011
LED_4 = 0xF7  # units     0x11110111
DOT = 0x80

# images
SEG8codes = [0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
            #   0    1   2    3    4    5    6    7
             0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71]
            #   8    9   A    b    C    d    E    F

class LED_8SEG_DISPLAY():
    def __init__(self):
```

```
        self.rclk = Pin(RCLK,Pin.OUT)
        self.rclk(1)
        self.spi = SPI(1)
        self.spi = SPI(1,1000_000)
        self.spi = SPI(1,10000_000,polarity=0,
                       phase=0,sck=Pin(SCK),
                       mosi=Pin(MOSI),miso=None)
        self.SEG8=SEG8codes

    def write_cmd(self, Num, Seg):
        self.rclk(1)
        self.spi.write(bytearray([Num])) # segment select
        self.spi.write(bytearray([Seg])) # segment code
        self.rclk(0)
        sleep(0.002)
        self.rclk(1)

DISPLAY = LED_8SEG_DISPLAY()

while True:
    DISPLAY.write_cmd(LED_1,SEG8codes[1])
    sleep(0.001)
    DISPLAY.write_cmd(LED_2,SEG8codes[3]|DOT)
    sleep(0.001)
    DISPLAY.write_cmd(LED_3,SEG8codes[5])
    sleep(0.001)
    DISPLAY.write_cmd(LED_4,SEG8codes[7]|DOT)
    sleep(0.001)
```

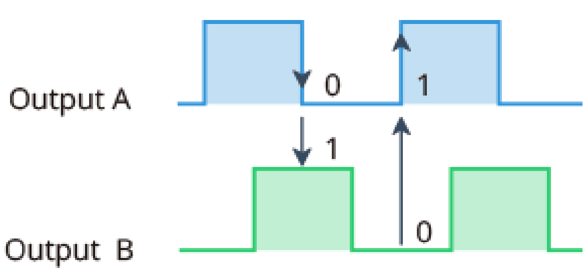
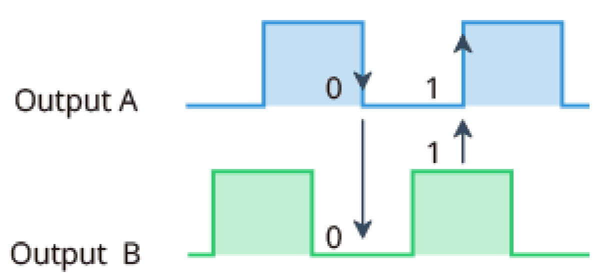
Task.5. Write a MicroPython script to increment and decrement the number visible on the LED display.

Task.6. Connect the EC-11 encoder.



GND	the ground connection
VCC	the positive supply voltage - +3.3V or +5V
SW	the output of the push button switch (active low). When the knob is depressed, the voltage goes LOW
DT (output B)	similar to CLK output, but it lags behind CLK by a 90° phase shift. This output is used to determine the direction of rotation
CLK (output A)	is the primary output pulse used to determine the amount of rotation. Each time the knob is turned in either direction by just one detent (click), the 'CLK' output goes through one cycle of going HIGH and then LOW

When A changes state:

if B != A, then the knob is turned clockwise	if B = A, the knob is turned counterclockwise.
	

Create a MicroPython script to send to Your computer the current position, current rotation direction (CW/CCW), and the total number of CW and CCW full rotations.

Task.7. Use EC-11 encoder to control SG90 servo shaft position.

For those interested:

1. MicroPython web page:
   micropython.org/download/rp2-pico/
2. Control a Servo Motor with Raspberry Pi Pico Using PWM in MicroPython tutorial:
   circuitdigest.com/microcontroller-projects/control-a-servo-motor-with-raspberry-pi-pico-using-pwm-in-micropython
3. MicroPython - interrupts handling.
   docs.micropython.org/en/latest/reference/isr_rules.html?highlight=interrupt
4. Raspberry Pi Pico - how to use External Interrupts.
   microcontrollerslab.com/pir-motion-sensor-raspberry-pi-pico-external-interrupts-tutorial/
5. Generate Delay with Raspberry Pi Pico Timers using MicroPython.
   microcontrollerslab.com/generate-delay-raspberry-pi-pico-timers-micropython/
6. Waveshare Wiki on Pico-8SEG-LED.
   www.waveshare.com/wiki/Pico-8SEG-LED