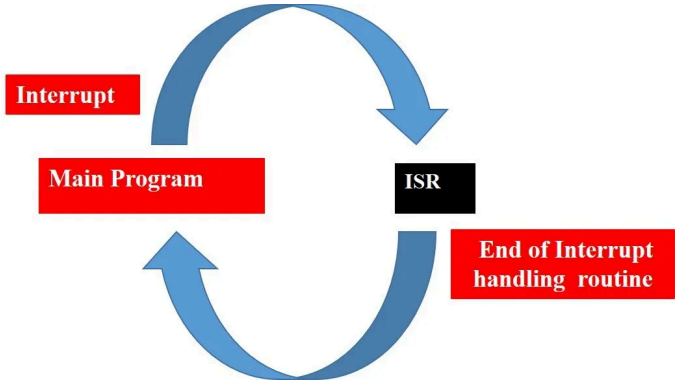
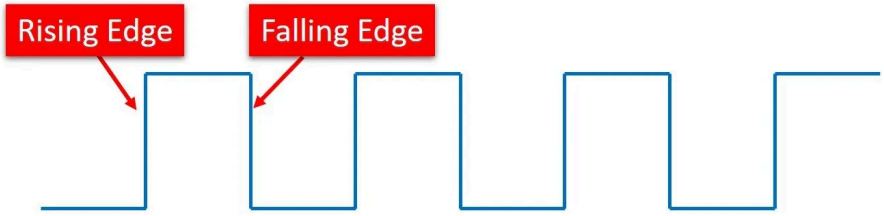
Wprowadzenie. Przerwania, w systemach wbudowanych, są wykorzystywane do reagowania na zdarzenia, które powinny mieć wpływ na wykonywanie głównej sekwencji programu. Przykładem takiego zdarzenia może być wystapienie określonego stanu lub zmiana stanu na wejściu cyfrowym mikrokontrolera. Przerwania powiązane z określonym zasobem sprzętowym (np. GPIO) noszą nazwę przetwań sprzętowych (hardware interrupts). Jeżeli źródło przerwania jest sygnał zewnętrzny dla mikrokontrolera przerwania nosi nazwę zewnętrznego (external interrupt).
W przypadku zgłoszenia przerwania mikrokontroler zatrzymuje wykonywanie głównego programu, a następnie jest wywoływana funkcja obsługi przerwania - ISR (Interrupt Servive Routine). Po wykonaniu kodu znajdującego się w ciele ISR mikrokontroler powraca do wykonywania programu głównego (Main Program).

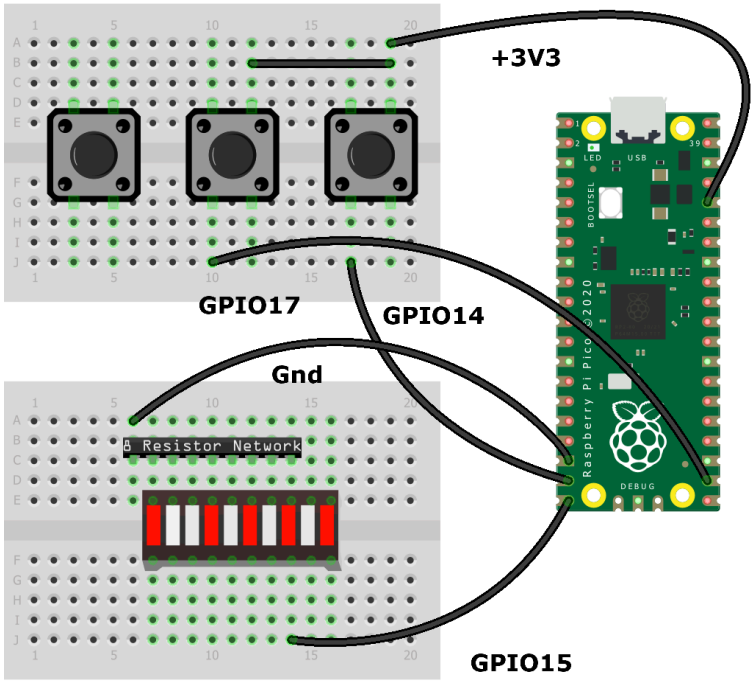


W module Raspberry Pi Pico każdy pin GPIO może zostać tak skonfigurowany, aby zgłaszać przerwanie związane z następującymi zdarzeniami:

- stan niski (`trigger=Pin.IRQ_LOW_LEVEL`)
- stan wysoki (`trigger=Pin.IRQ_HIGH_LEVEL`)
- zbocze narastające - rising/positive edge (`trigger=Pin.IRQ_RISING`)
- zbocze opadajace - falling/negative edge (`trigger=Pin.IRQ_FALLING`)



Zad 1. Korzystając z płytki Raspberry Pi Pico Expander Board lub płytki stykowej, połącz następujący układ:



Przykład kodu:

```
from machine import Pin
from utime import sleep

BTN_ON_PIN = 14
BTN_TOGGLE_PIN = 17

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)
```

Zad 2. Napisz skrypt w MicroPythonie, który wymusi zmianę stanu LED podłączonej do GPIO25 (wbudowany) bez korzystania z modułu *utime*. Okres 1s, wypełnienie 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)
```

Timery są bardziej efektywnym narzędziem do zarządzania zależnościami czasowymi w programie. W porównaniu z funkcjami typu *sleep* nie powodują one zatrzymania wykonania programu głównego. Funkcja *Timer()* ma trzy argumenty:

- *period* - okres wywoływania przerwania (w milisekundach);
- *mode:*
  - *Timer.PERIODIC* - okresowe wywołanie przerwania czasowego;
  - *Timer.ONE_SHOT* - przerwanie jest wywoływane jednokrotnie.
- *callback* - funkcja obsługi przerwania (ISR).

Implementacja z wykorzystaniem funkcji lambda:

```
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())
```

Zad 3. Wykorzystaj *Timer* do rozwiązania problemy drgania styków przy przełączaniu.

```
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_on = 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_tog = 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)
```

Zad 4. Podłącz moduł Pico-8SEG-LED.

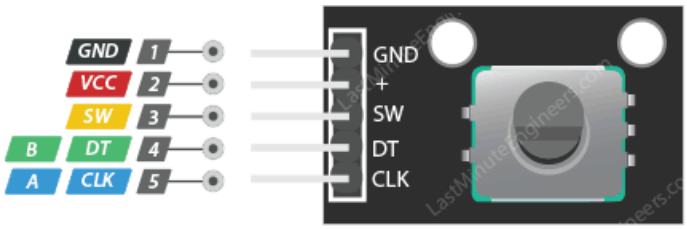
Przykład kodu:

```
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():
```

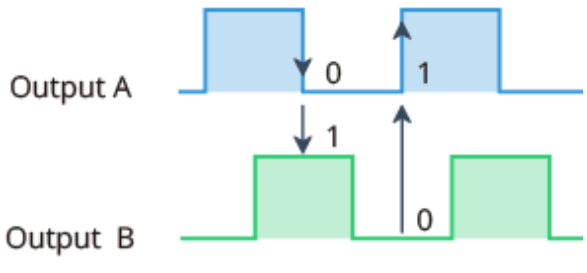
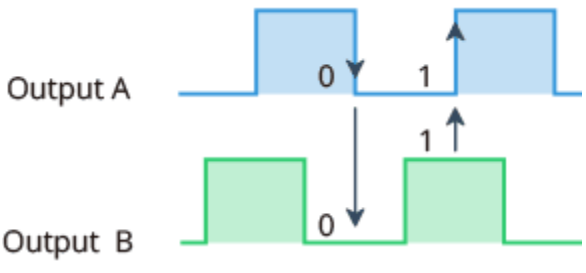
```
    def __init__(self):
        self.rclk = Pin(RCLK,Pin.OUT)
        self.rclk(1)
        self.spi = SPI(1)
        self.spi = SPI(1,1000_000)
        self.spi = SPI(1,10000_000,polarity=0,
                       phase=0,sck=Pin(SCK),
                       mosi=Pin(MOSI),miso=None)
        self.SEG8=SEG8codes

    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)
```

Zad 5. (**0.5 pkt.**) Napisz skrypt w MicroPython, który pozwoli na zwiększanie i zmniejszanie o 1 wartości wyświetlanej za pomocą modułu Pico-8SEG-LED. Wykorzystaj układ z zadania 1. Wykorzystaj wszystkie segmenty wyświetlacza. Dodaj przycisk do zerowania stanu wyświetlacza.

Zad 6. (**1 pkt**) Napisz skrypt w MicroPython, który będzie wysyłał do komputera bieżącą pozycję, ilość wykonanych pełnych obrotów (CW/CCW) oraz bieżący kierunek obrotu osi enkodera EC-11/HW-040.



GND	potencjał odniesienia
VCC	zasilanie - +3.3V or +5V
SW	Przycisk. Przycisk nie jest naciśnięty - stan LOW.
DT (output B)	Wyjście impulsowe - kanał B (przesunięcie fazowe o ¼ okresu - 90°).
CLK (output A)	Wyjście impulsowe - kanał A.

B != A, obrót CW	B = A, obrót CCW
	

Zad 7. (**1.5 pkt.**) Wykorzystaj enkoder EC-11 do kontroli pozycji wału mikroserwomechanizmu SG-90. Wymagana pozycja powinna być ustawiona i widoczna na wyświetlaczy *Pico-8SEG-LED*. Naciśnięcie przycisku powinno wymuszać obrót. Długie naciśnięcie przycisku powinno zmieniać kierunek obrotu - wyświetlany na wyświetlaczu.

Dla zainteresowanych:

1. MicroPython strona:
   micropython.org/download/rp2-pico/

2. Sterowanie serwomechnizmem w MicroPython:
   circuitdigest.com/microcontroller-projects/control-a-servo-motor-with-raspberry-pi-pico-using-pwm-in-micropython

3. MicroPython obsługa przerwań:
   docs.micropython.org/en/latest/reference/isr_rules.html?highlight=interrupt

4. Raspberry Pi Pico - obsługa przerwań zewnętrznych:
   microcontrollerslab.com/pir-motion-sensor-raspberry-pi-pico-external-interrupts-tutorial/

5. Zarządzanie timerami RP2040 w MicroPython:
   microcontrollerslab.com/generate-delay-raspberry-pi-pico-timers-micropython/

6. Waveshare Wiki Pico-8SEG-LED:
   www.waveshare.com/wiki/Pico-8SEG-LED