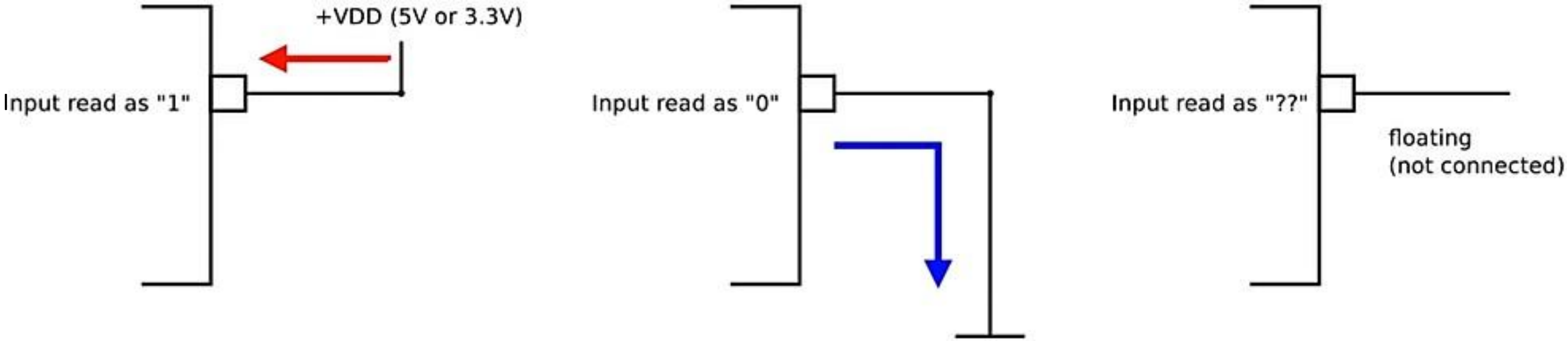
LAB6 Digital I/O

What is “digital I/O”

- It is an interface in which each electrical pin may have two states:
  - Logical 0 (0V)
  - Logical 1 (5V on the basis of the VDD)
- Each line can be programmer as:
  - an output (lit a LED)
  - an input   (read a pushbutton)

Digital Input: Electrical consideration

- An input connected to **VDD** is read (by software) as “**1**”
- An input connected to **Ground** is read (by software) as “**0**”
- If the input is **floating** (not connected), the value read cannot be determined!



Prevent floating state

- The typical pull-up resistor value is **1-10**kΩ .

- If in doubt, a good starting point when using a switch is **5**kΩ.

- Disadvantage

  -a larger resistance : the input pin responses to voltage changes slower

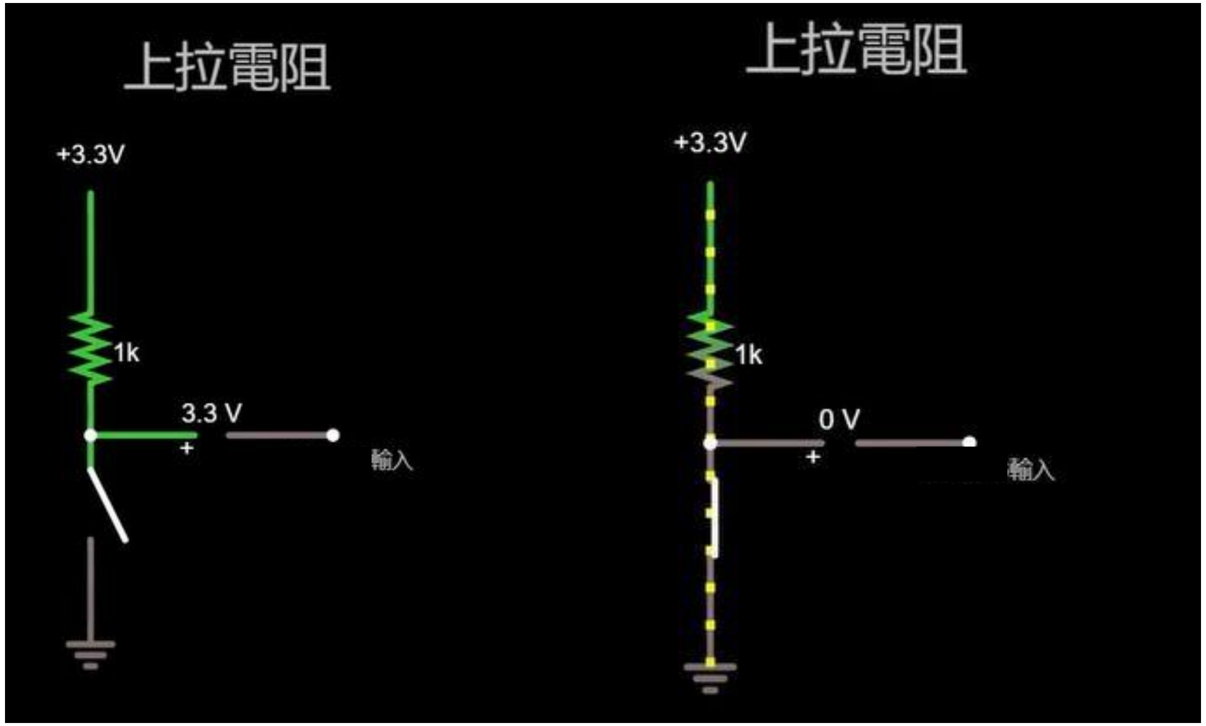
  -a smaller resistance : Too much current flow through

Prevent floating state

- **Use pull-up resistor**

-Switch open, the value read is “1”

-Switch closed, the value read is “0”

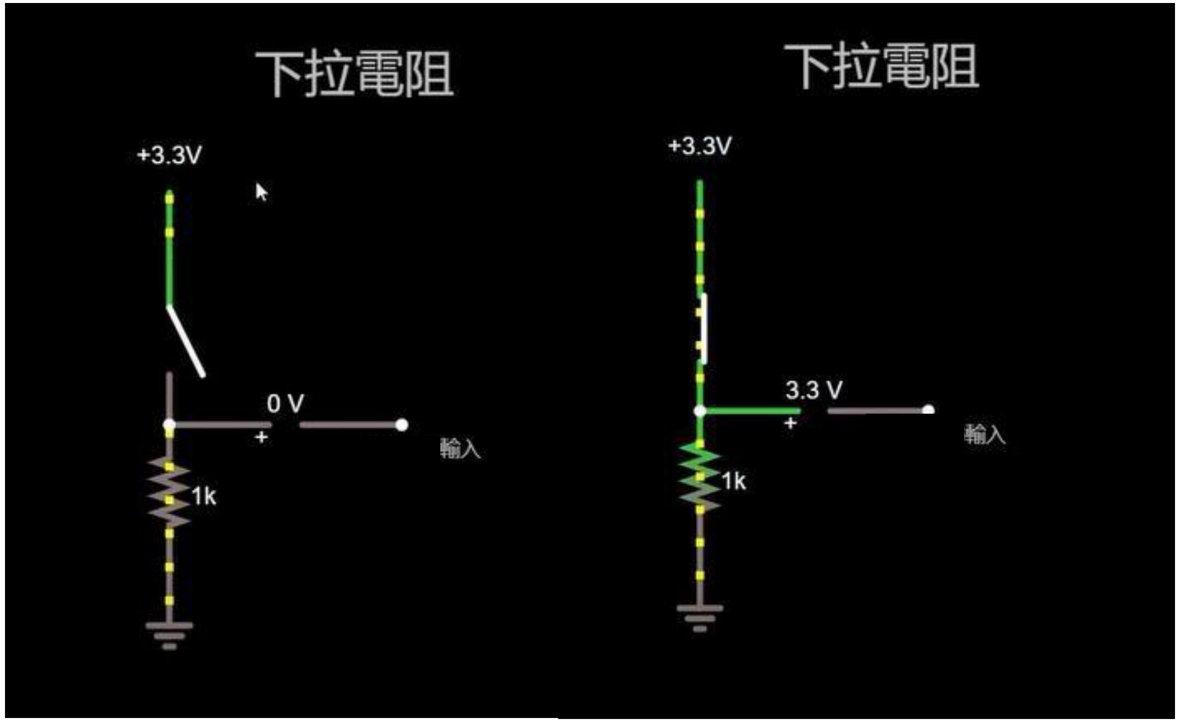


Prevent floating state

- **Use pull-down resistor**

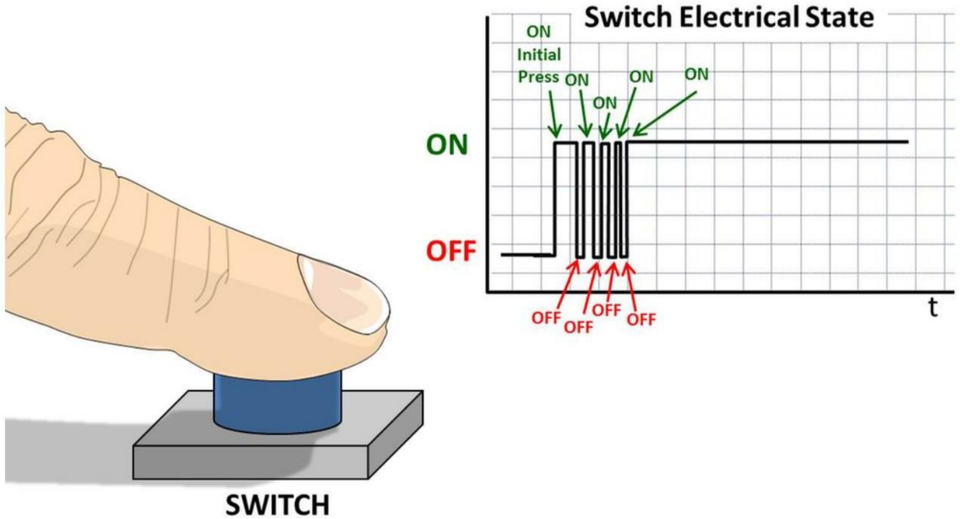
  - Switch open, the value read is “0”

- Switch closed, the value read is “1”



Bouncing problem!

- Due to mechanical reasons, pushbuttons and switches(which have a spring inside) typically generate a bouncing signal when pressed or released.
- The bouncing signal is read by software ,thus causing malfunctioning.

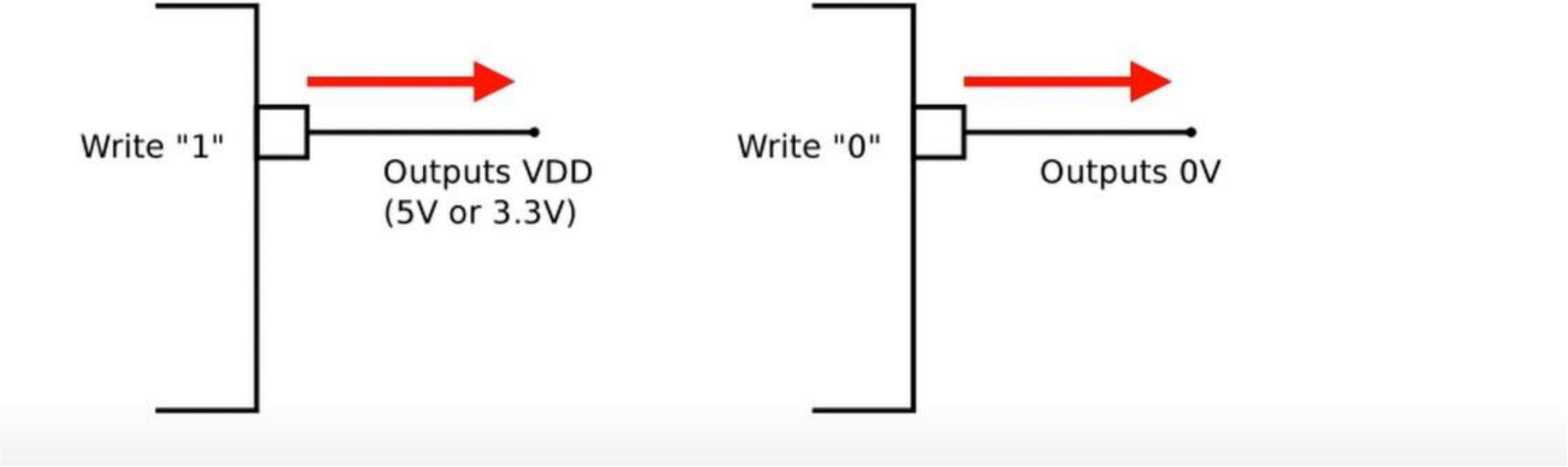


Button debounce

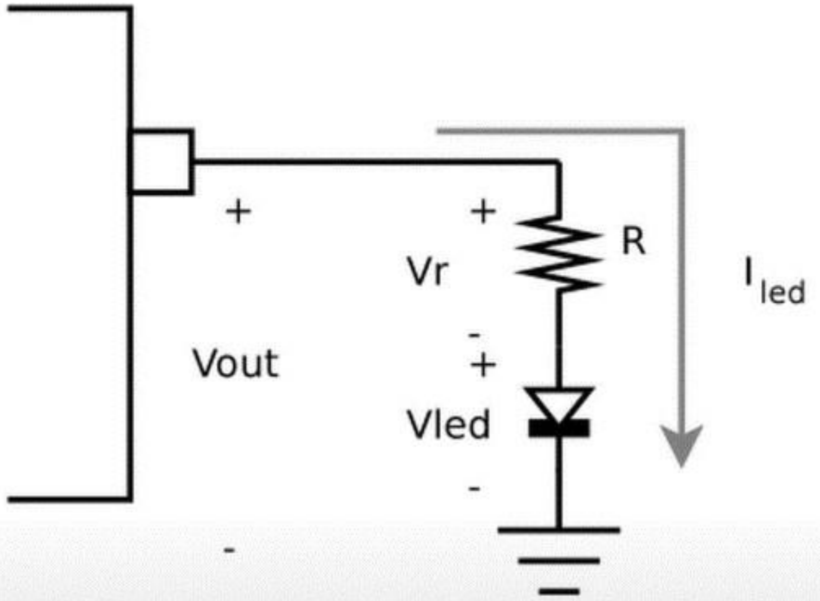
- Hardware solution
  - Add a **capacitor**, in parallel with the button, in order to filter the bouncing signal.
  - Reference: http://puppyodie.blogspot.com/2016/04/button-debounce.html
- Software solution
  - Use **delay** or **interrupt on change** function to check input signal again, if the signal stay the same, then considered as press.

Digital Output Electrical consideration

- Writing “1” : output high voltage(VDD)
- Writing “0” : output low voltage(GND)



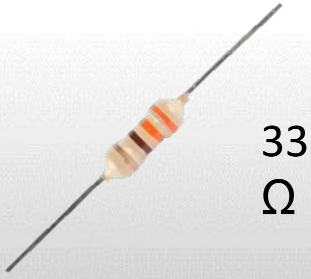
Connecting a LED: calculating the limiting resistor



- I_{led} LED lit current (about 20mA)
- V_{led} LED lit voltage (1.2V for small red leds)

$$V_{out} = V_{led} + V_r \qquad V_r = R \cdot I_{led}$$

$$R = \frac{V_{out} - V_{led}}{I_{led}} = \frac{5 - 1.2}{0.02} = 190\Omega$$



330
Ω

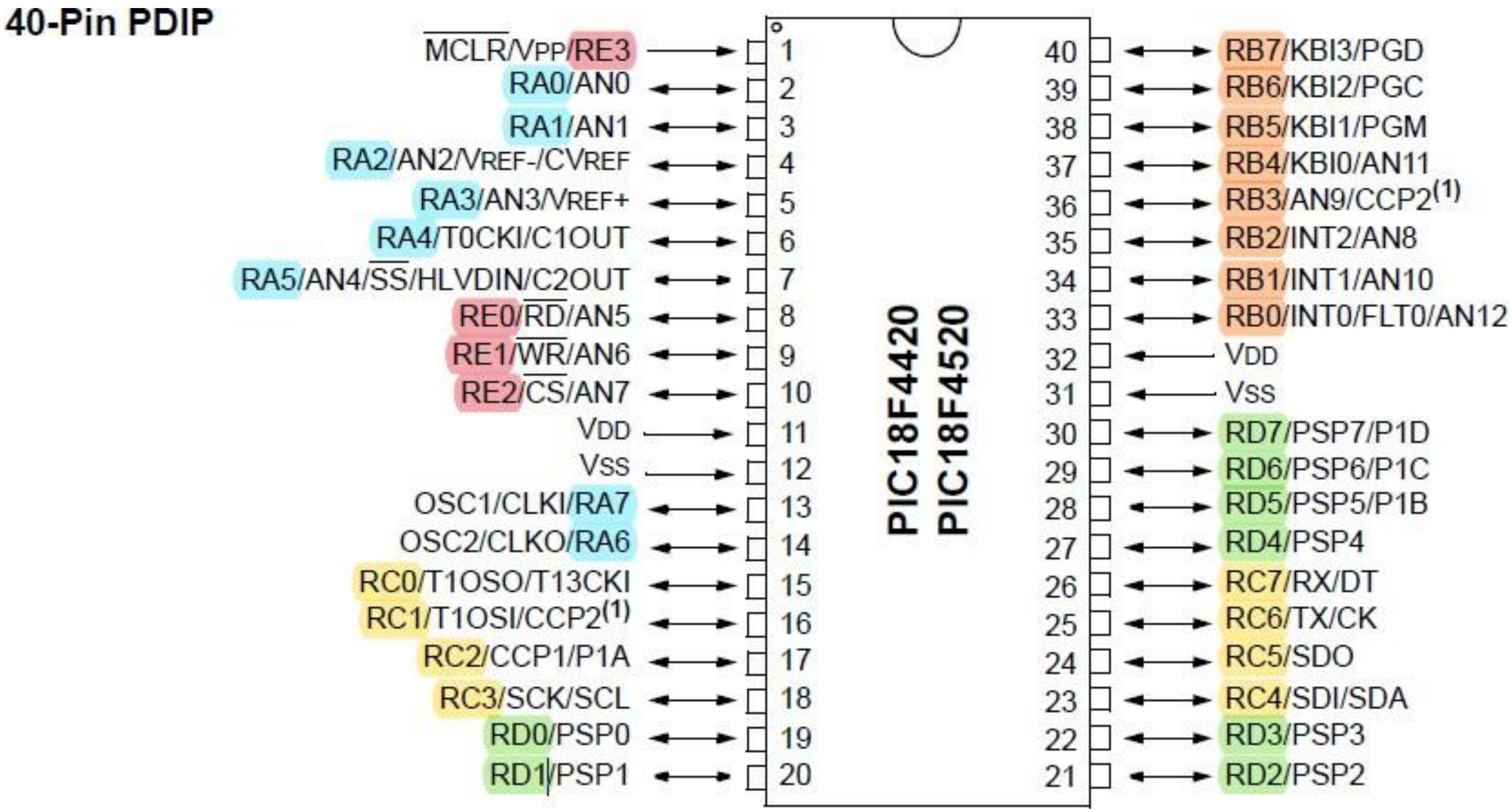
I/O ports of PIC18F4520

I/O ports of PIC18F4520

- PIC18F4520 has five I/O ports, called PORTA, PORTB, …, PORTE.
- Each port (A-D) has 8 bits and thus 8 electrical pins
- Pins are referred as **RXY**, where X is the port name(A,B,…,E)and Y is the bit number (0, 1, …, 7)

  - Ex: RA2→ bit 2 of PORTA
- Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general when a peripheral is enabled, **that pins may not be used as a general purpose I/O pins**.

I/O ports of PIC18F4520



SFR of PIC18F4520 I/O ports

- Each port has 3 registers for its operation. These registers are :
  - TRIS register - configure the port as Input or Output
  - PORT register - reads voltage of the device
  - LAT register - output voltage

SFR of PIC18F4520 I/O ports

- **TRISx**: each bit of this SFR programs the relevant PIN as input or output:
  - 0 means output
  - 1 means input
- Example:
  - TRISC = 0x30 ;// 0x30 = 0011 0000
    - RC0 to RC3 → outputs
    - RC4, RC5 → inputs
    - RC6, RC7 → outputs

SFR of PIC18F4520 I/O ports

- **LATx**: each bit of this SFR programs the output status of the relevant PIN (if it is programmed as output, otherwise it is ignored).

- Example:
  - LATB = 0xe0; // 0xe0 = 1110 0000
    - RB0 to RB4 output 0;
    - RB5 to RB7 output 1;

SFR of PIC18F4520 I/O ports

- **PORTx**: each bit of this SFR reflects the input status of the relevant PIN (if the pin is configured as input, otherwise it replies as the bit of the LATx register

- Example:
  - Let us read into button variable, the status of the RA5 input pin:

```
int button = (PORTA & 0x20);    // 0x20 = 0010 0000
```

Bit Field Manipulation

Bit Field Manipulation in assembly

- **Single bit** manipulation
  - **BCF f, b, a** - clear bit b of register f

    ex: BCF LATB, 0, 0 // will clear LATB bit 0
  - **BSF f, b, a** - set bit b of register f

    ex: BSF TRISA, 5, 0 // will set TRISA bit 5
  - **BTG f, b, a** - toggle bit b of register f

    ex: BTG LATC, 3, 0 // will toggle LATC bit 3

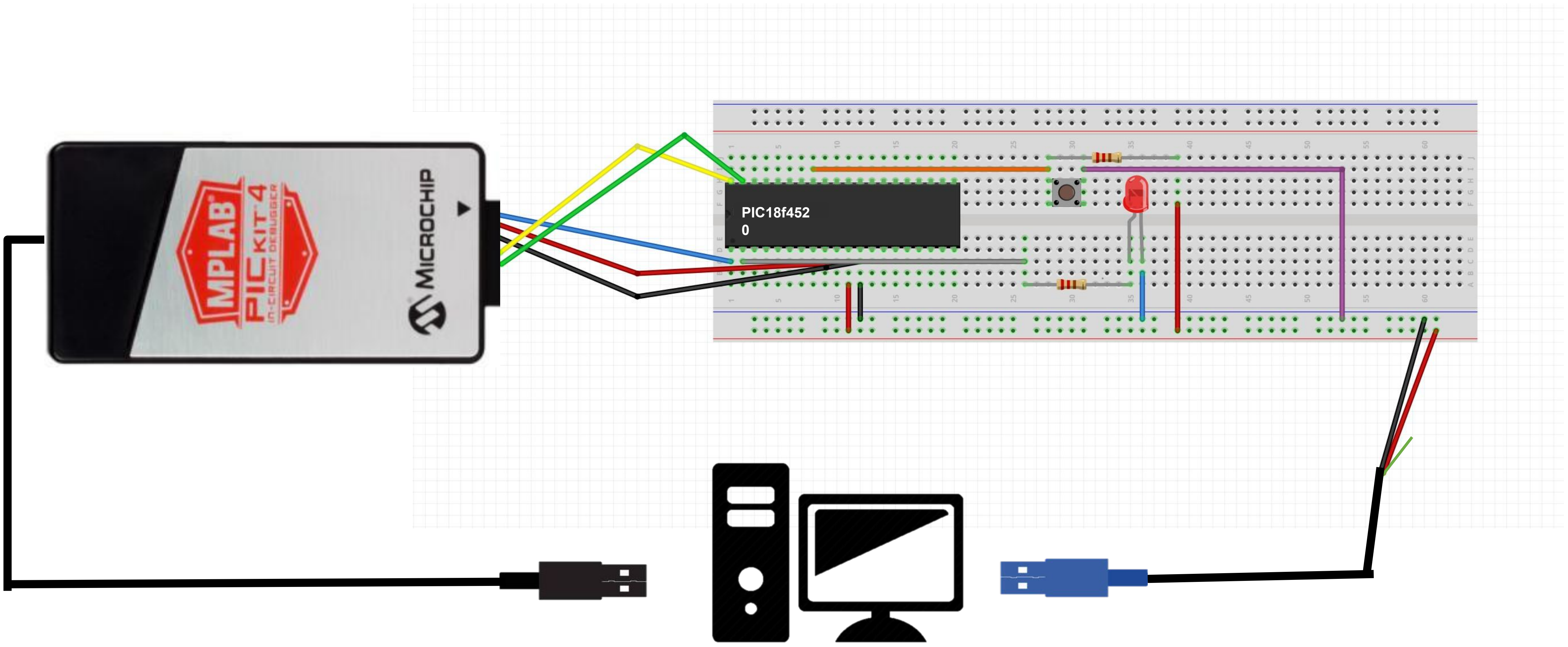
Bit Field Manipulation in assembly

- **Multiple bits** manipulation
  - Clear bit: use **ANDWF** operation
  - Set bit: use **IORWF** operation
  - Toggle bit: use **XORWF** operation

Bit Field Manipulation in C

- The processor-specific header file includes a structure definition that allows the user to access individual bits of a register.

- Example
  - PORTBbits.RB0 = 1; // pull PORTB bit 0 to high
  - LATBbits.LATB0 = 0; // pull LATB bit 0 to low
  - TRISBbits.TRISB0 = 0; // pull TRISB bit 0 to low
  - STATUSbits.C = 0; // clear the C flag to 0


MPLAB
PICkit 4
in-circuit debugger
MICROCHIP
PIC18f452
0