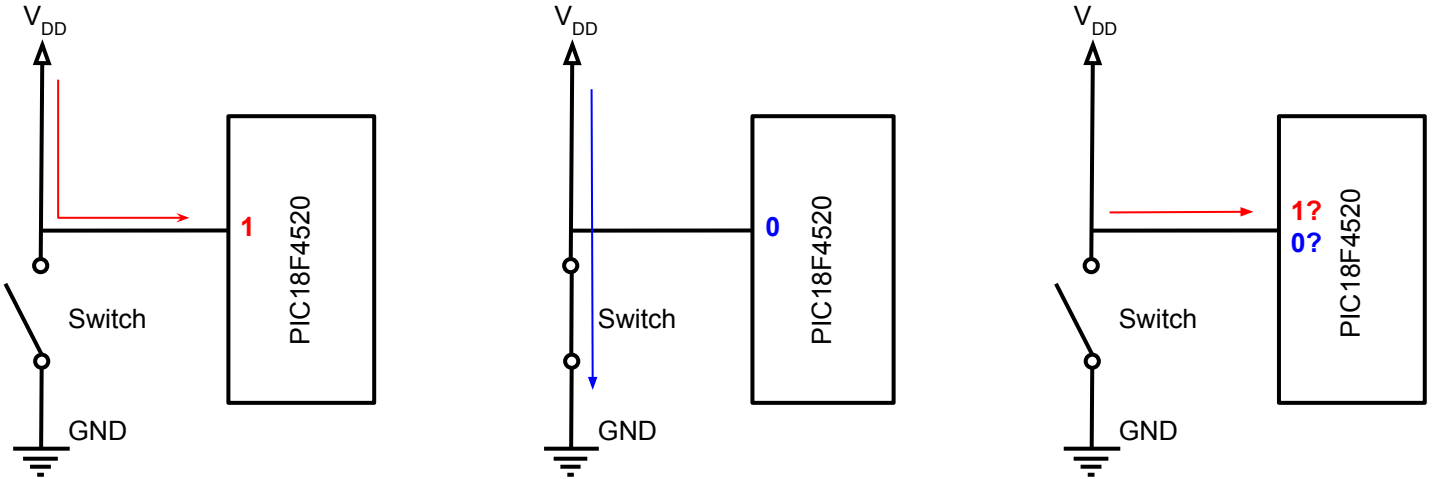
LAB6 Digital I/O

Digital I/O

- It is an interface to configure a channel electronically as either a **digital input** or **digital output**, without the need to change hardware.
- Each electrical pin may have two states:
  - **logical low / 0** (0 ~ 0.8 V)
  - **logical high / 1** (2 ~ 5 V)
- Each line can be programmer as:
  - **an output** (it generates a current and can be used, for example, to **lit a LED**)
  - **an input** (it receives a current and can be used, for example, to **read a pushbutton**)

Digital Input: Electrical consideration

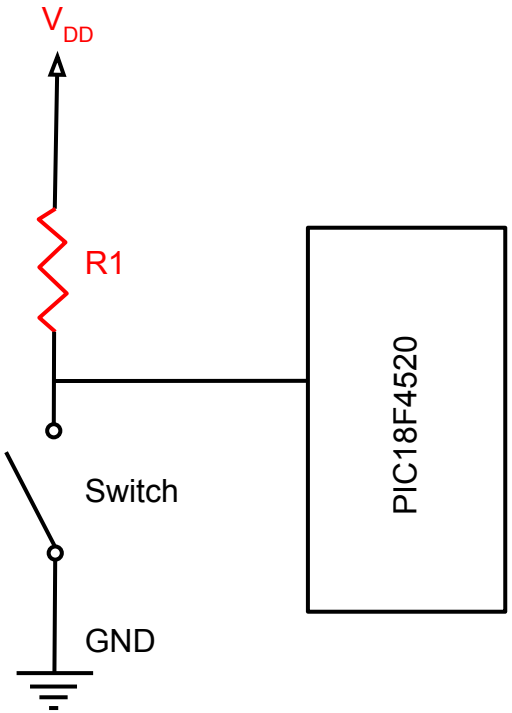
- Ideal condition:
  - An input connected to **V_{DD}** is read (by software) as "*1*"
  - An input connected to **Ground** is read (by software) as "*0*"
- In fact, if the input is **floating** (not connected), the value read **cannot be determined**



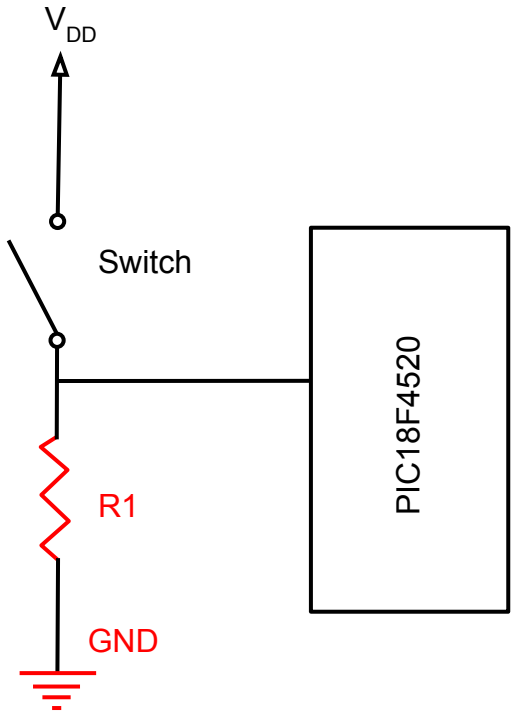
Solution: Pull-up or Pull-down Resistor

- Pull-up Resistors:
  - Connection between the voltage supply and the pin(**V_{DD} - resister - pin**)
  - Switch **opened/closed**, the input read is **1/0**
  - For switch, the typical pull-up resistor value is 1-10 kΩ
  - If in doubt, a good starting point when using a switch is **4.7 kΩ**
  - Calculating a Pull-up Resistor Value (eg. V_{DD} = 5V, I = 1 mA, then R = 5 kΩ)
- Pull-down Resistors:
  - Connection between ground and the pin(**GND - resister - pin**)
  - Switch **opened/closed**, the input read is **0/1**

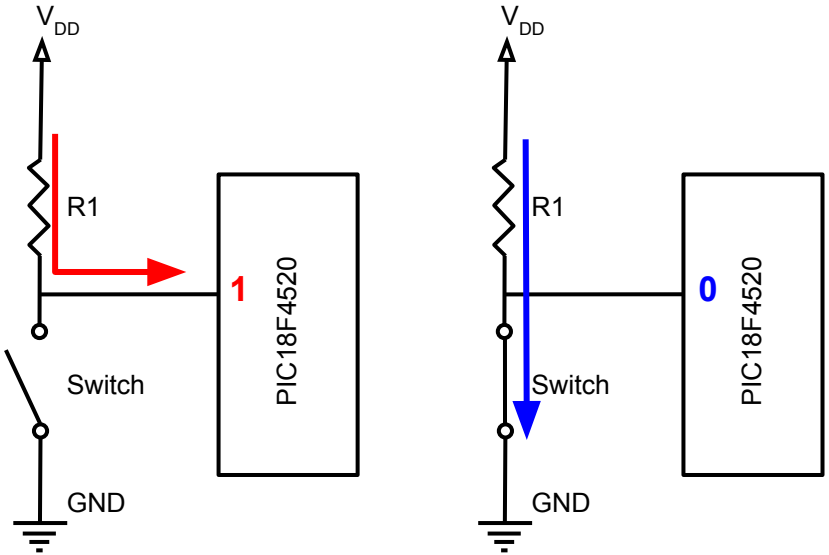
Schematic of Pull-up and Pull-down Resistor



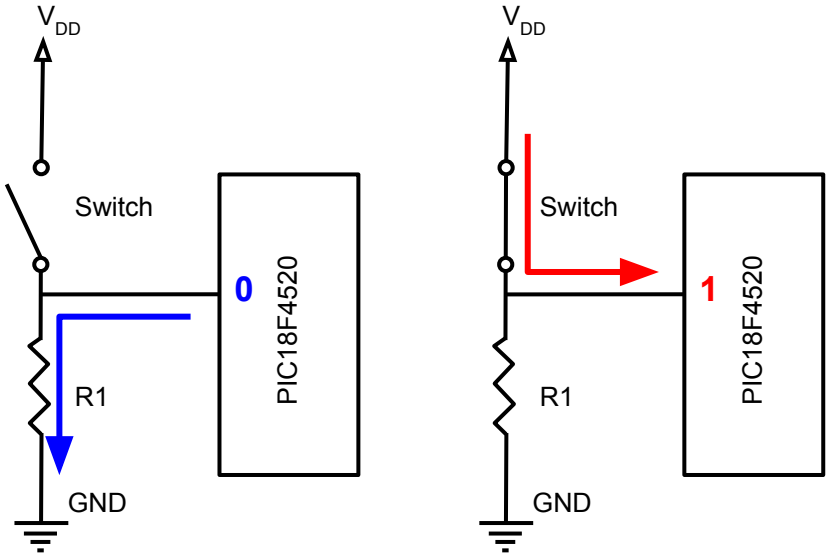
Pull-up resistor



Pull-down resistor



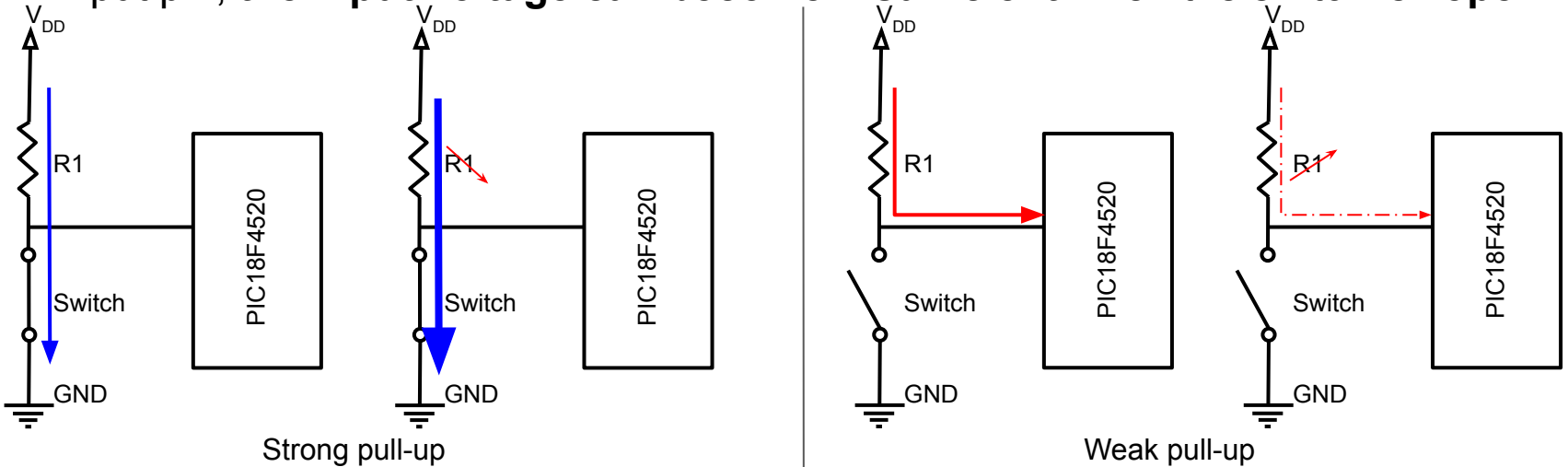
Pull-up resistor



Pull-down resistor

Pull-up resistor

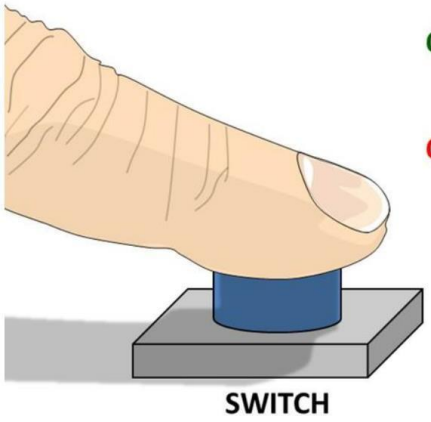
- **Strong pull-up**(a **smaller** resistance): **a high current** will flow through the pull-up resistor, heating the device and using up an unnecessary amount of power when the switch is **“closed“**
- **Weak pull-up**(a **larger** resistance): combined with a large leakage current of the input pin, **the input voltage can become insufficient** when the switch is **“open”**

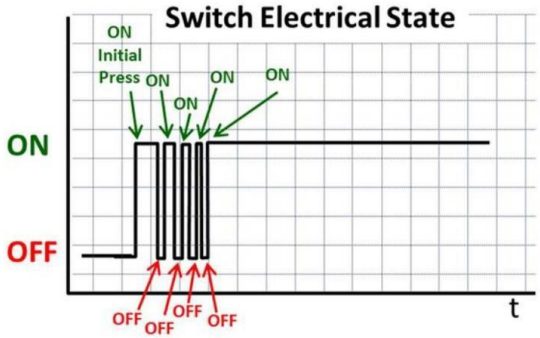


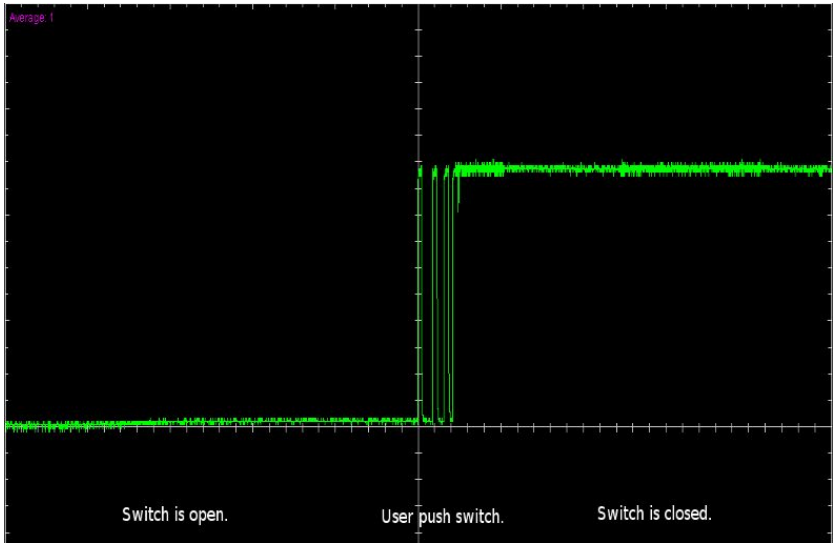
Bouncing Problem

- When we push a button,
  - for the user, it might seem that the contact is only made once
  - for hardware, it strikes a metal contact and physically bounces a few times

    because the switch contact is metal and it has elasticity

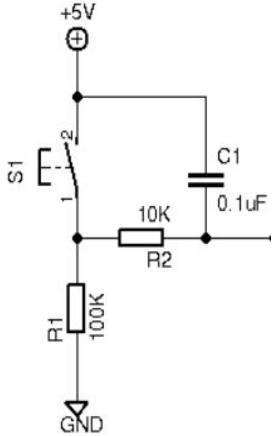


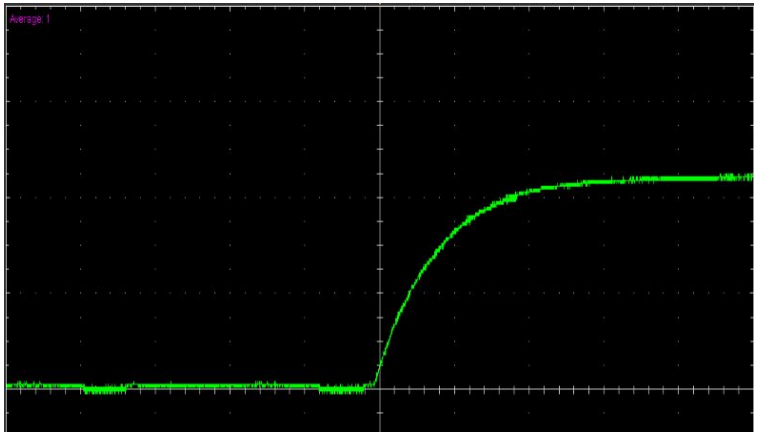




How to Deal with It

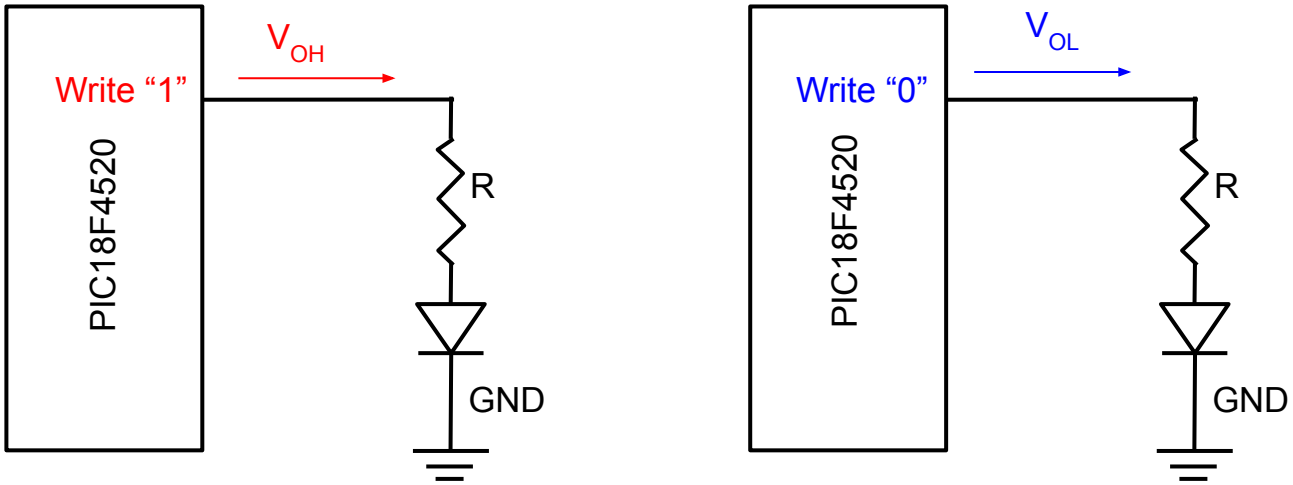
- **Hardware setup**: **Add a capacitor**, in parallel with the button, in order to filter the bouncing signal
- **Software debounce**: Use **delay** or **interrupt** on change function to force the microcontroller to wait some time for the bouncing to stop or detect changes in the switch state (typically on the rising or falling edge)





Digital Output: Electrical consideration

- Writing **“1”**: output **high** voltage($V_{OH} \approx V_{DD} = 5$ V)
- Writing **“0”**: output **low** voltage($V_{OL} \approx$ GND = 0 V)



Resistors in LED Circuits

- The intensity of the light is dependent on the amount of current
- A current limiting resistor is always connected to the LED to prevent burning out the LED
- The value of this resistor is calculated using the formula

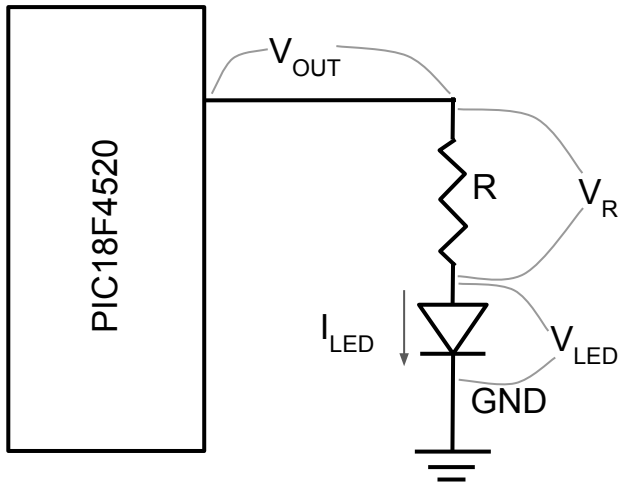
$V_{OUT} = V_R + V_{LED}$ (Kirchhoff's circuit laws)

$V_R = I_{LED} * R$ (Ohm's law)

$\Rightarrow R = (V_{OUT} - V_{LED}) / I_{LED}$

$= (5V - 1.2V) / 20\ mA = 190\ \Omega$

- For small red led, $I_{LED} = 20$ mA, $V_{LED} = 1.2$ V

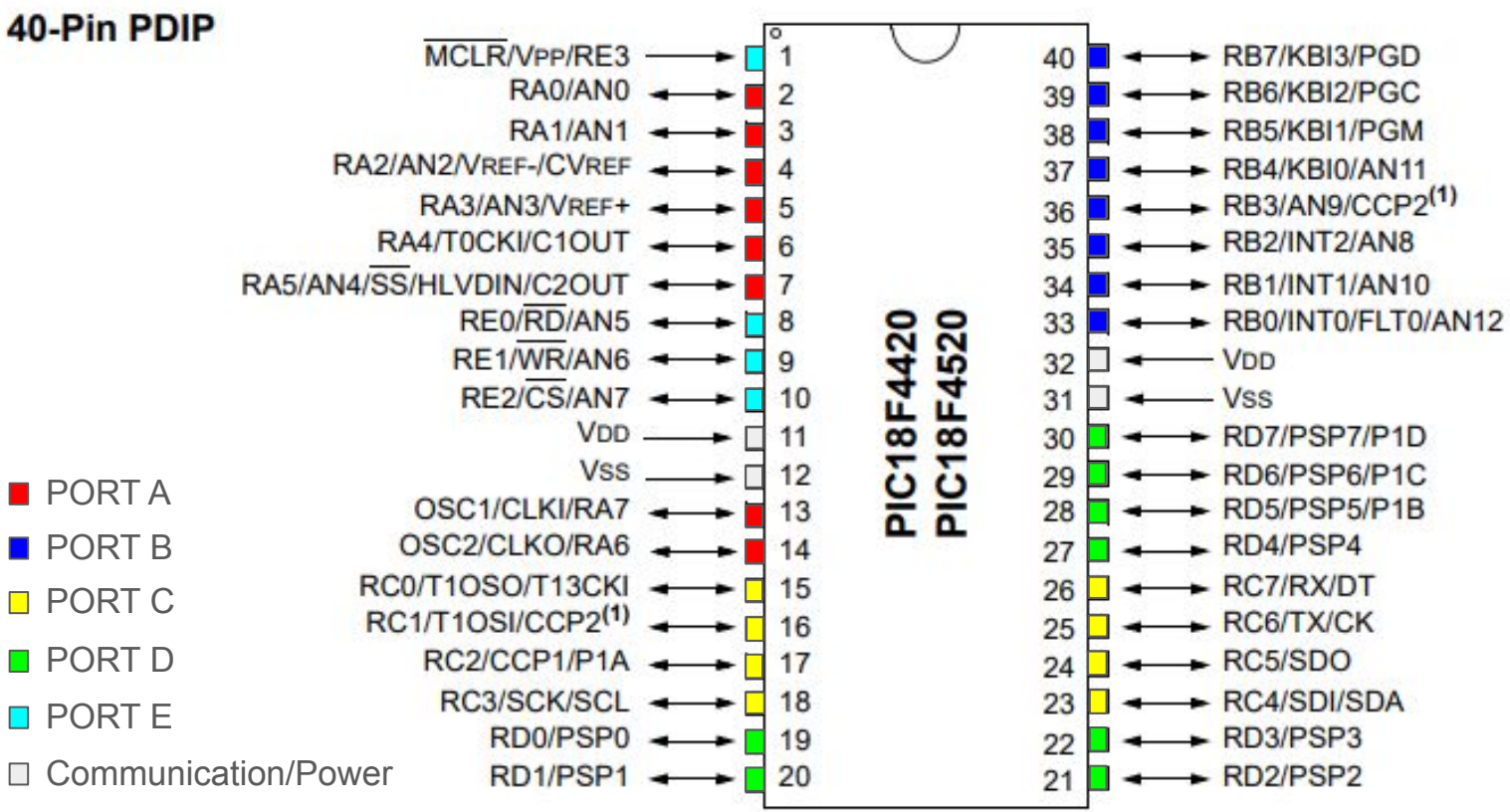


I/O Ports of PIC18F4520

I/O Ports of PIC18F4520

- MCUs of the PIC18 family have several **ports**, called **PORTA, PORTB, PORTC, ...,**
- Each port 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** (0, 1, ..., 7) eg. the pin RA3 is the bit 3 of the port A
- **Some pins** of the I/O ports are **multiplexed** with an alternate function from the peripheral features. In general, when **a peripheral is enabled**, that **pins may not be used as a general purpose I/O pins**

Pins of PICF4520



I/O Ports Control Registers

- Each I/O port has **three registers** for its operation, where x is a letter that denotes the particular I/O port:
  - **TRISx**: PORTx Data Direction Control register
  - **PORTx**: I/O Port register
  - **LATx**: PORTx Data Latch register
- In PIC18F4520, there are **5 PORTs**(PORTA, PORTB, PORTC, PORTD and PORTE). Since, there are **5 TRISx** and **LATx**

TRISx (TRIState)

- Control the direction of the PORTx pins
  - **Inward (Input, 1, high)**
  - **Outward (Output, 0, low)**
- By default, every TRISx is inputs (1111 1111) - the pins will not conflict with any logic outputs or other voltage sources connected to them
- eg. TRISB = 0x50 ; 0x50 = 0101 0000

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	1	0	0	0	0
Output	Input	Output	Input	Output	Output	Output	Output

PORTx

- Data on an I/O pin is accessed via a PORTx register
- A read of the PORTx register reads the value of I/O pin
- A write to the PORTx register writes the value to the port data latch(LATx)
- Input/Output decided by TRISx(1 for input, 0 for output)
- By default, every PORT is inputs PORT
- If PORTB is used as **input port** then to **read data** from all bit of PORTB

```
PORTB = 0x03;    // reading the status of the pins
```

- If PORTB is used as **output port** and we want to **write data** 0x03 on PORTB

```
PORTB = 0x03;    // assigning high logic to RB0 and RB1
```

LATx(LATch)

- The LAT (Latch) register is associated with an I/O pin and eliminates the problems that could occur with read-modify-write instructions
- Read-modify-write operations on the LATx register read and write the latched output value for PORTx
    - Latch Read: returns **the values held in the port output latches** instead of the values on the I/O pins.
    - Latch Write: the same effect as a write to the PORT register. A write to the PORT register writes the data value to the port latch.

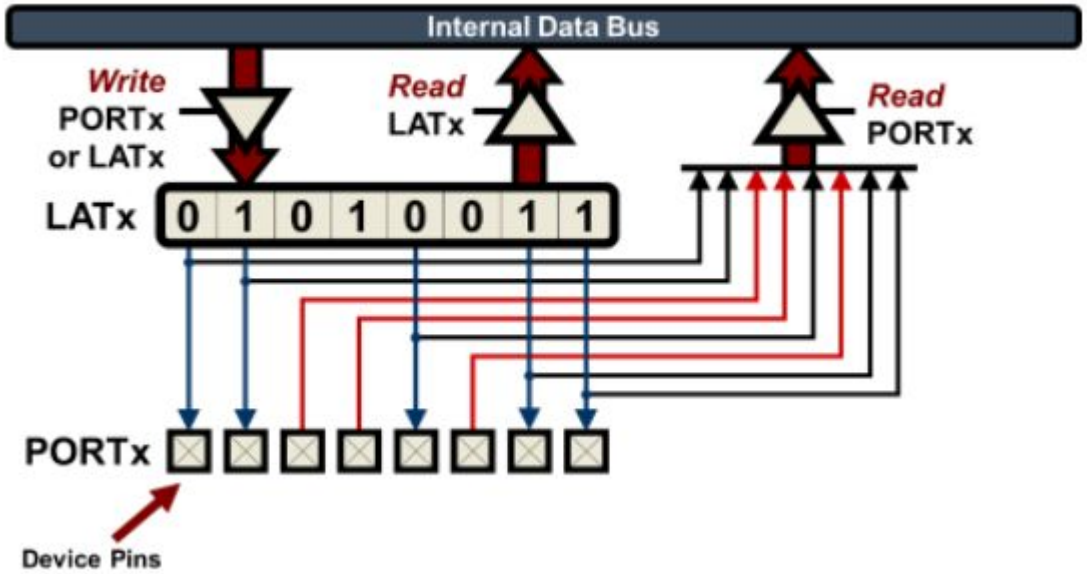
PORT vs LATCH

The differences between the PORT and LAT registers can be summarized as follows:

	PORTx	LATx
Read	Reads data value **on the I/O pin**	Reads data value held **in the port latch**
Write	Writes data value to the port latch	Writes data value to the port latch

PORT vs LATCH

The differences between the PORT and LAT registers can be summarized as follows:



PORTA, TRISA and LATA Registers

- RA4/T0CKI/C1OUT pin: multiplexed with the Timer0 module clock input and one of the comparator outputs
- RA6 and RA7: multiplexed with the main oscillator pins
- RA<3:0> and RA5: multiplexed with **analog inputs**, the analog V_{REF+} and V_{REF-} inputs and the comparator voltage reference output

EXAMPLE 10-1: INITIALIZING PORTA

```
CLRF    PORTA   ; Initialize PORTA by
                ; clearing output
                ; data latches
CLRF    LATA    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Fh     ; Configure A/D
MOVWF   ADCON1  ; for digital inputs
MOVLW   07h     ; Configure comparators
MOVWF   CMCON   ; for digital input
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISA   ; Set RA<3:0> as inputs
                ; RA<5:4> as outputs
```

The operations of pins RA<0:3> and RA5 as A/D Converter inputs is selected by clearing or **setting** the control bits in the **ADCON1 (A/D Control Register 1 register)**

REGISTER 19-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

PORTB, TRISB and LATB Registers

- Each of PORTB pin has a **weak internal pull-up**
  - RBPU (INTCON2<7> = 0) can turn on all the pull-ups
  - automatically turned off when the port pin is configured as an output)
-

```
CLRF    PORTB   ; Initialize PORTB by
                ; clearing output
                ; data latches
CLRF    LATB    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Fh     ; Set RB<4:0> as
MOVWF   ADCON1  ; digital I/O pins
                ; (required if config bit
                ; PBADEN is set)
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISB   ; Set RB<3:0> as inputs
                ; RB<5:4> as outputs
                ; RB<7:6> as inputs
```

PORTC, TRISC and LATC Registers

EXAMPLE 10-3: INITIALIZING PORTC

```
CLRF    PORTC   ; Initialize PORTC by
                ; clearing output
                ; data latches
CLRF    LATC    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISC   ; Set RC<3:0> as inputs
                ; RC<5:4> as outputs
                ; RC<7:6> as inputs
```

PORTD, TRISD and LATD Registers

EXAMPLE 10-4: INITIALIZING PORTD

```
CLRF    PORTD   ; Initialize PORTD by
                ; clearing output
                ; data latches
CLRF    LATD    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISD   ; Set RD<3:0> as inputs
                ; RD<5:4> as outputs
                ; RD<7:6> as inputs
```

PORTE, TRISE and LATE Registers

- PORTE is a **4-bit wide** port
- 4^{th} pin of PORTE(MCLR/Vpp/RE3) is **an input only pin** - when selected as a port pin(MCLRE = 0), it functions as a digital input only pin

PORTE, TRISE and LATE Registers

EXAMPLE 10-5: INITIALIZING PORTE

```
CLRF    PORTE   ; Initialize PORTE by
                ; clearing output
                ; data latches
CLRF    LATE    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Ah     ; Configure A/D
MOVWF   ADCON1  ; for digital inputs
MOVLW   03h     ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISE   ; Set RE<0> as inputs
                ; RE<1> as inputs
                ; RE<2> as outputs
```

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

  Let WREG = 0x56 (0101 0110), TRISA = 0xA4 (1010 0100)

  - **Clear bits**: use **ANDWF** operation

    ex: ANDWF TRISA, **1** // WREG = 0x56, **TRISA = 0x04 (0000 0100)**
  - **Set bits**: use **IORWF** operation

    ex. IORWF TRISA, **0** // **WREG = 0xF6 (1111 0110)**, TRISA = 0xA4
  - **Toggle bits**: use **XORWF** operation

    ex. XORWF TRISA, **1** // WREG = 0x56, **TRISA = 0xF2 (1111 0010)**

BCF

BCF	**Bit Clear f**
Syntax:	BCF f, b {,a}
Operands:	$0 \le f \le 255$ $0 \le b \le 7$ $a \in [0,1]$
Operation:	$0 \rightarrow f<b>$
Status Affected:	None
Encoding:	1001 bbba ffff ffff
Description:	Bit 'b' in register 'f' is cleared. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BCF FLAG_REG, 7, 0`

Before Instruction
FLAG_REG = C7h
After Instruction
FLAG_REG = 47h

BSF

BSF	**Bit Set f**
Syntax:	BSF f, b {,a}
Operands:	$0 \le f \le 255$ $0 \le b \le 7$ $a \in [0,1]$
Operation:	$1 \rightarrow f<b>$
Status Affected:	None
Encoding:	1000 bbba ffff ffff
Description:	Bit 'b' in register 'f' is set. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BSF FLAG_REG, 7, 1`

Before Instruction
FLAG_REG = 0Ah
After Instruction
FLAG_REG = 8Ah

BTG

BTG **Bit Toggle f**

Syntax: BTG f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

Operation: $(\overline{f<b>}) \rightarrow f<b>$

Status Affected: None

Encoding:

0111	bbba	ffff	ffff

Description: Bit 'b' in data memory location 'f' is inverted.
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BTG PORTC, 4, 0`

Before Instruction:
PORTC = 0111 0101 [75h]
After Instruction:
PORTC = 0110 0101 [65h]

ANDWF

ANDWF	**AND W with f**
Syntax:	ANDWF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .AND. (f) → dest
Status Affected:	N, Z
Encoding:	0001 01da ffff ffff
Description:	The contents of W are ANDed with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default). If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default). If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ANDWF REG, 0, 0`

Before Instruction

W = 17h

REG = C2h

After Instruction

W = 02h

REG = C2h

IORWF

IORWF **Inclusive OR W with f**

Syntax: IORWF f {,d {,a}}

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (W) .OR. (f) → dest

Status Affected: N, Z

Encoding:

0001	00da	ffff	ffff

Description: Inclusive OR W with register 'f'. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example:

```
IORWF  RESULT, 0, 1
```

Before Instruction
RESULT = 13h
W = 91h
After Instruction
RESULT = 13h
W = 93h

XORWF

XORWF **Exclusive OR W with f**

Syntax: XORWF f {,d {,a}}

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (W) .XOR. (f) → dest

Status Affected: N, Z

Encoding:

0001	10da	ffff	ffff

Description: Exclusive OR the contents of W with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in the register 'f' (default).
If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See

Words: 1

Cycles: 1

Q Cycle Activity:

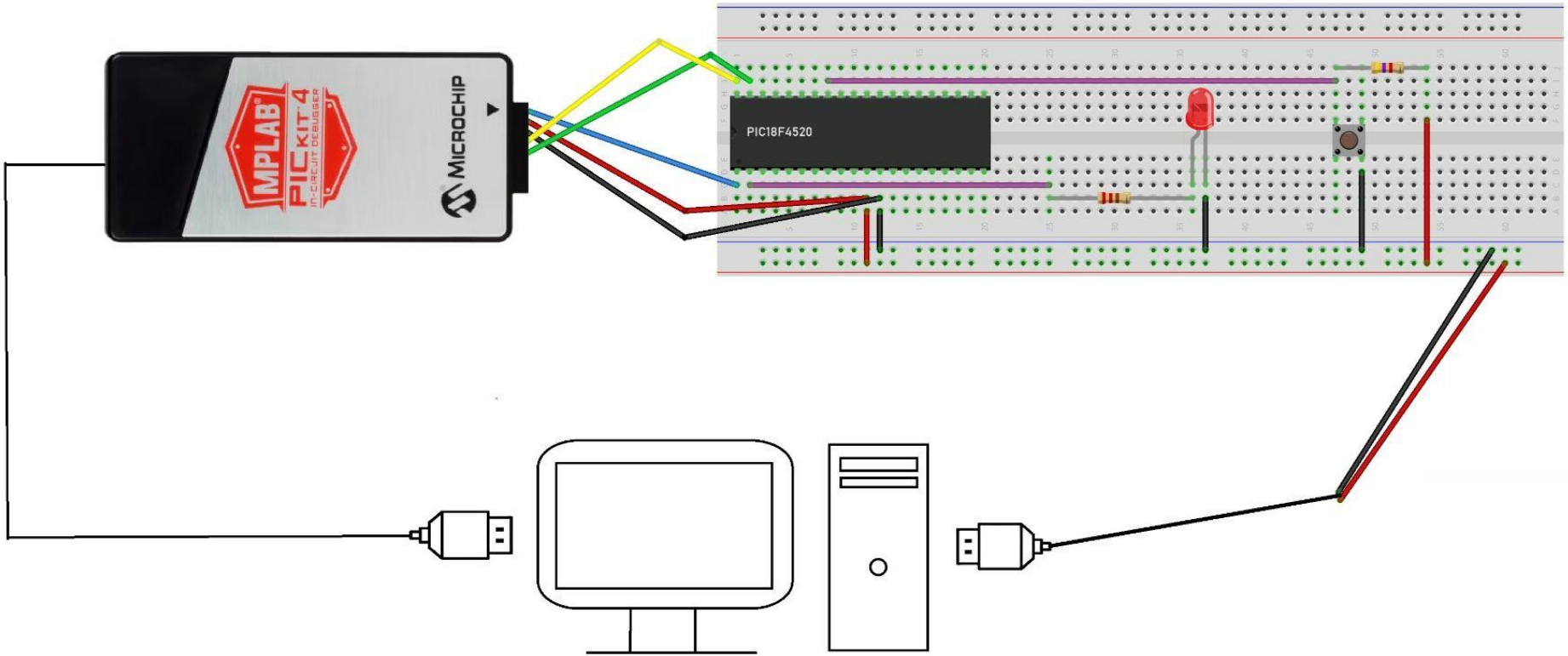
Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example:
```
XORWF   REG, 1, 0
```

Before Instruction
REG = AFh
W = B5h
After Instruction
REG = 1Ah
W = B5h

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
PIC18F4520