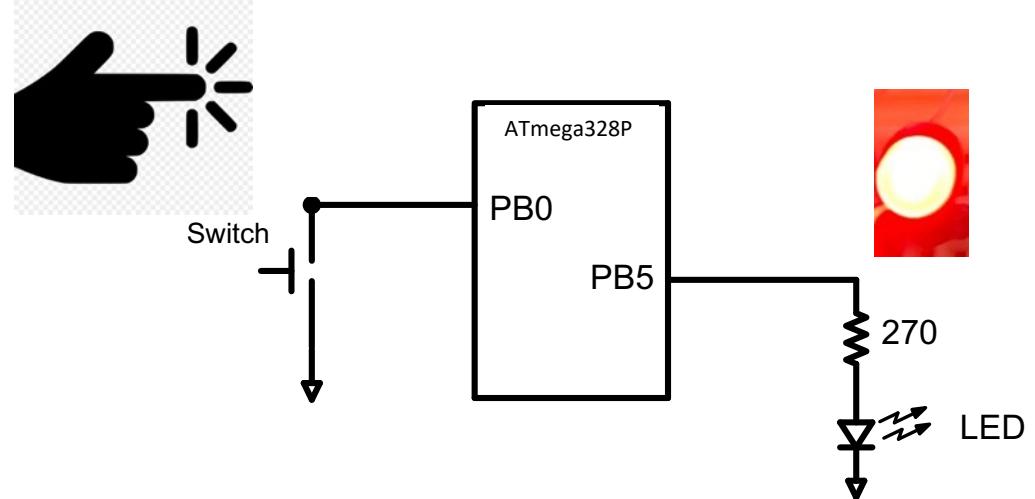


# Lecture 8.4

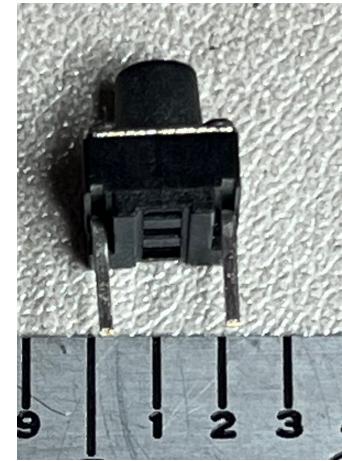
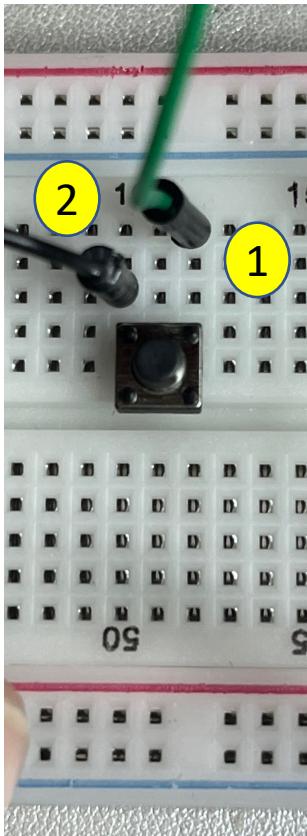
## button switches, LEDs, and GPIO Inputs

Prof. David McLaughlin  
ECE Department  
University of Massachusetts  
Amherst, MA



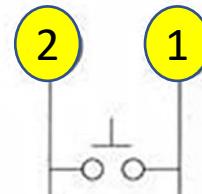
Spring 2024

# Momentary Push-Button Switch (2 pin)



pitch (pin spacing) is 0.2 inches or 5 mm.

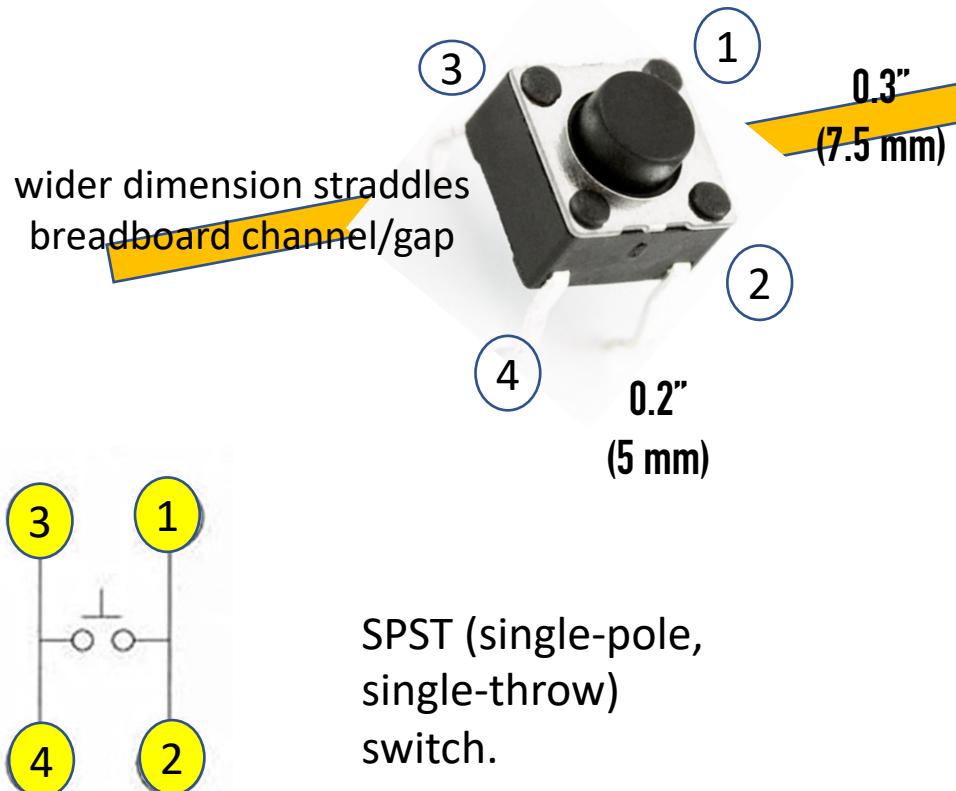
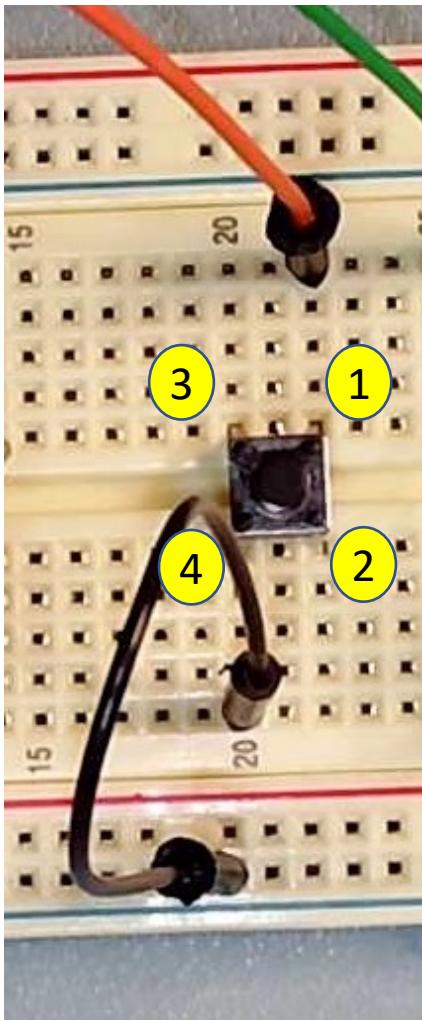
Too narrow to span the channel



SPST (single-pole,  
single-throw)  
switch.

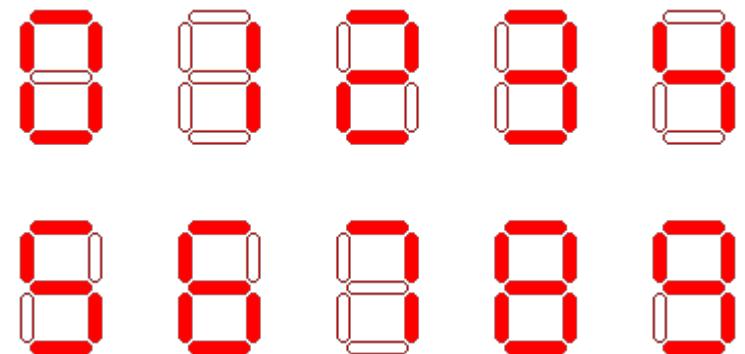
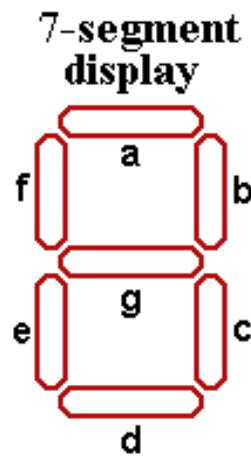
These 2-pin push-buttons are in the Eureak Leap ECE-231 kits Spring 2024  
Marston 221 Parts Depot has 4-pin push-buttons.

# Momentary Push-Button Switch (4 pin)



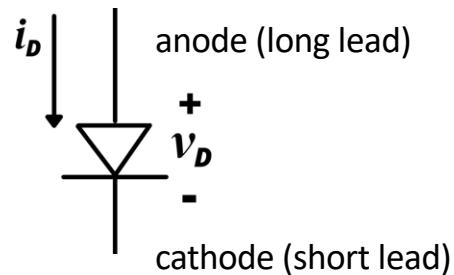
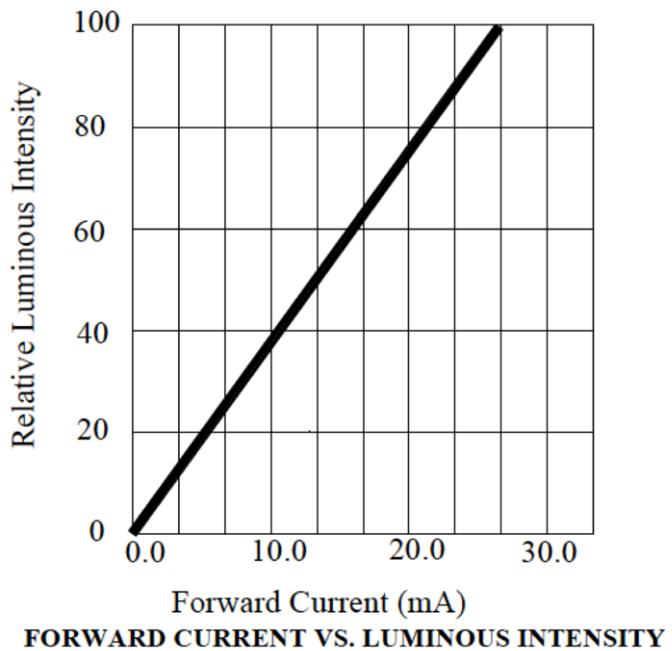
These 4-pin push-buttons are in the Marston 221 Parts Depot

# LED Interfacing



Get the polarity correct; use a current-limiting resistor

Light Emitting Diode (LED): brightness proportional to forward current  
-- we consider here the LEDs in our kits --



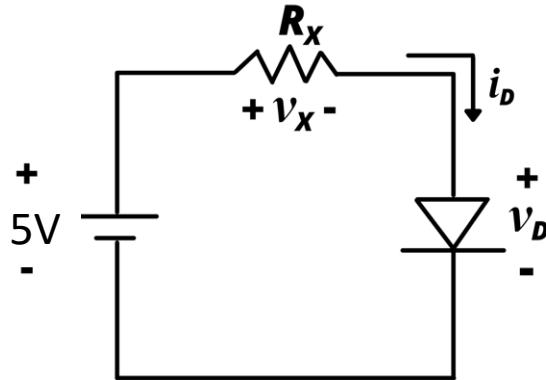
forward voltage:  $v_D = 2V$  (depends on diode color)  
forward current:  $i_d = 30 \text{ mA}$ , max



anode      cathode

Want to know more about diodes? See  
<http://openbooks.library.umass.edu/funee/chapter/4-1/>

## Use a current-limiting resistor to drive an LED



$-5 + v_x + v_D = 0$  (via Kirchoff's Voltage Law)

$-5 + i_D R_x + v_D = 0$  (via Ohm's Law)

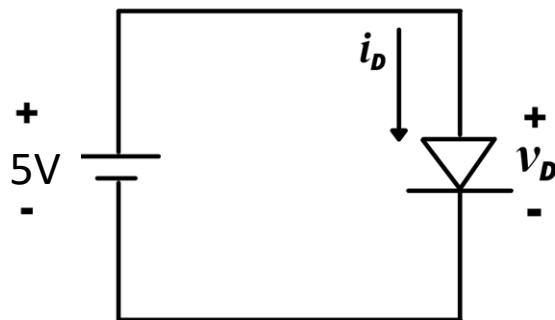
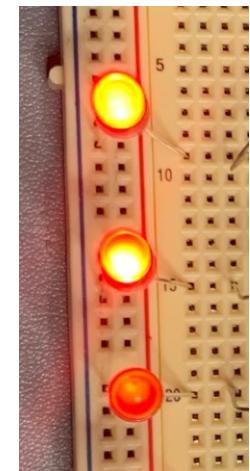
$-5 + i_D R_x + 2 = 0$  (using  $v_D = 2$  volts)

$i_D = 3/R_x$  (current is limited by resistance  $R_x$ )

$$R_x = 100 \Omega \rightarrow i_d = 0.03 \text{ A} = 30 \text{ mA}$$

$$R_x = 1000 \Omega = 1\text{k} \Omega \rightarrow i_d = 0.003 \text{ A} = 3 \text{ mA}$$

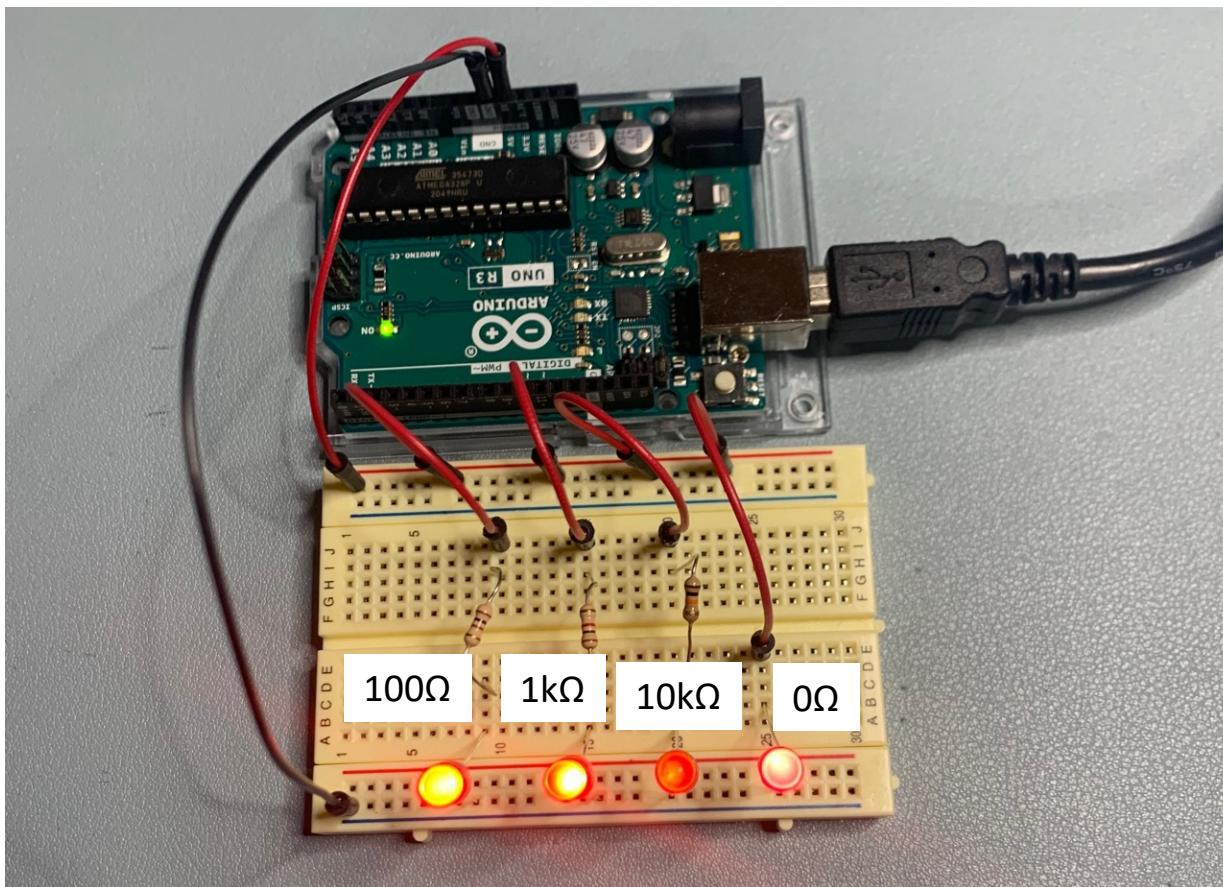
$$R_x = 10,000 \Omega = 1\text{k} \Omega \rightarrow i_d = 0.0003 \text{ A} = 0.3 \text{ mA}$$



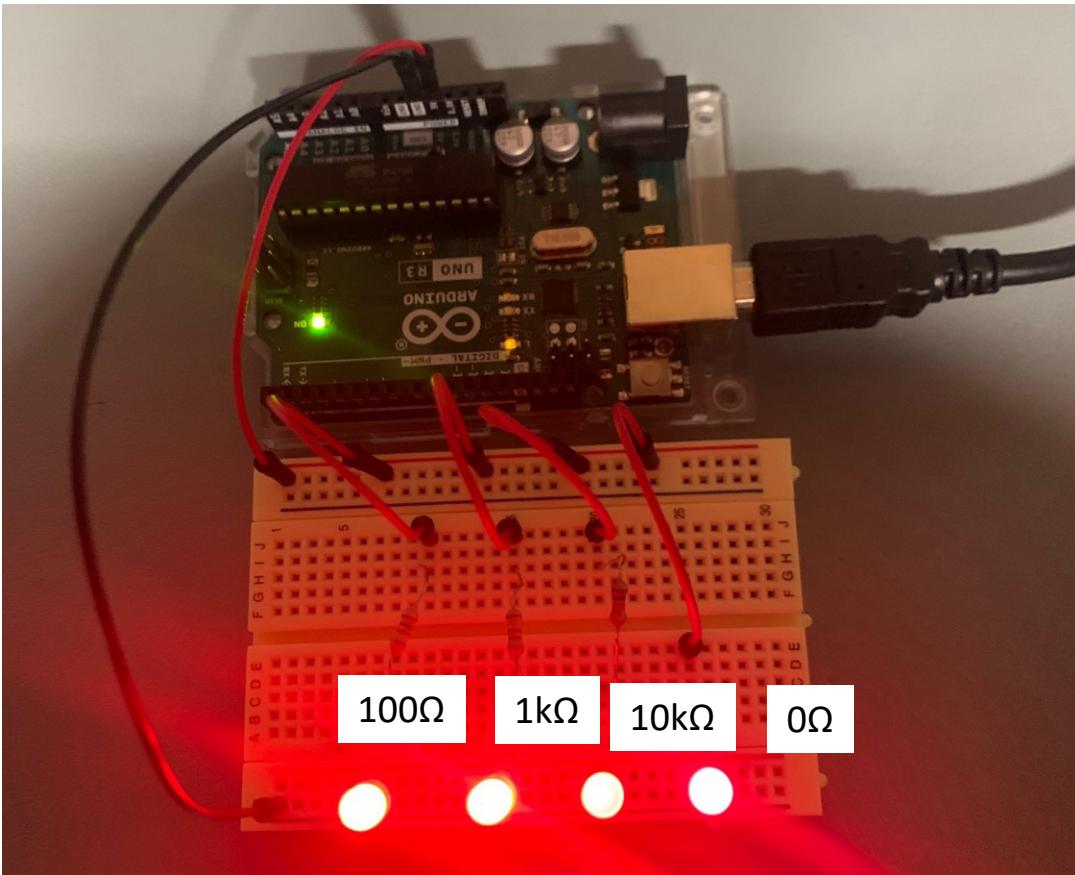
$$R_x = 0 \Omega \rightarrow i_d = \infty \text{ A}$$

(limited by the max current from the power supply or by the resistance of the power supply & wires. In either case, the LED is damaged)



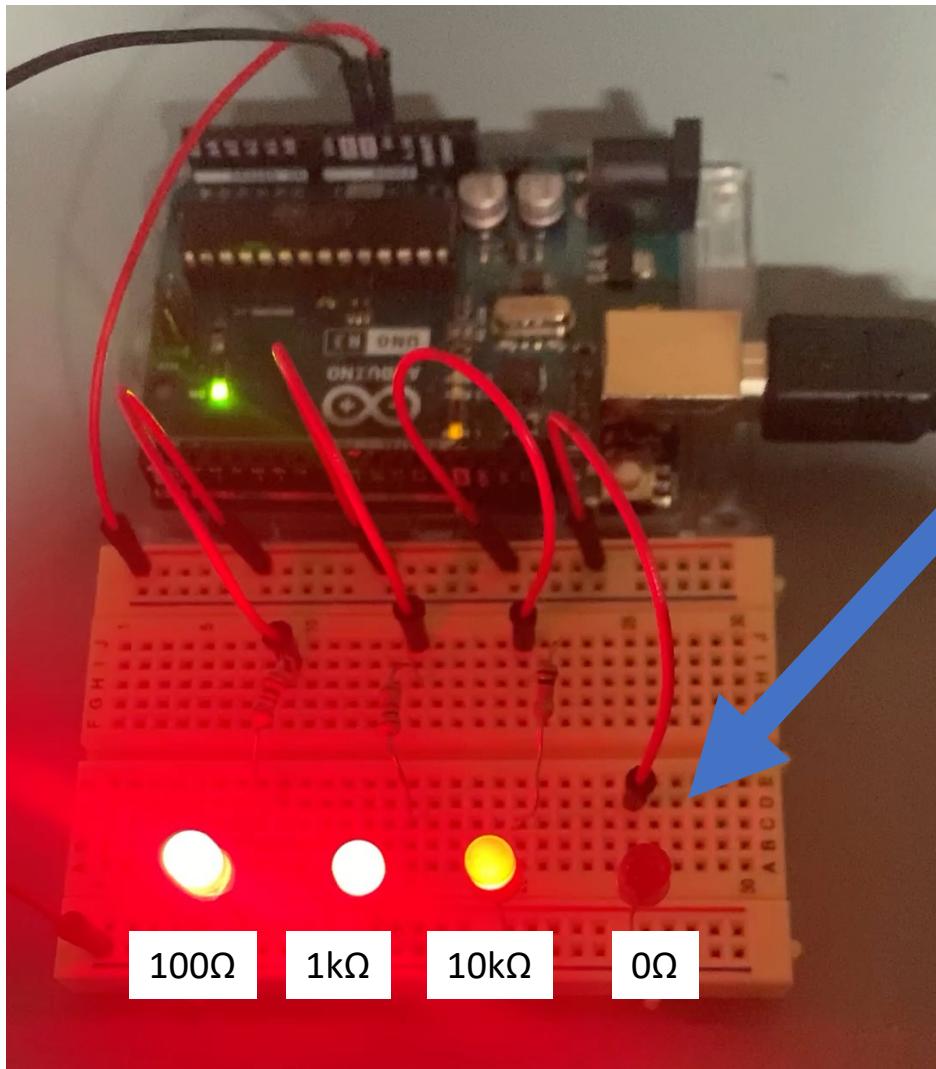


LED's powered by the 5V output pin from Arduino Uno (200 mA max)



LED's powered by the 5V output pin from Arduino Uno (200 mA max)

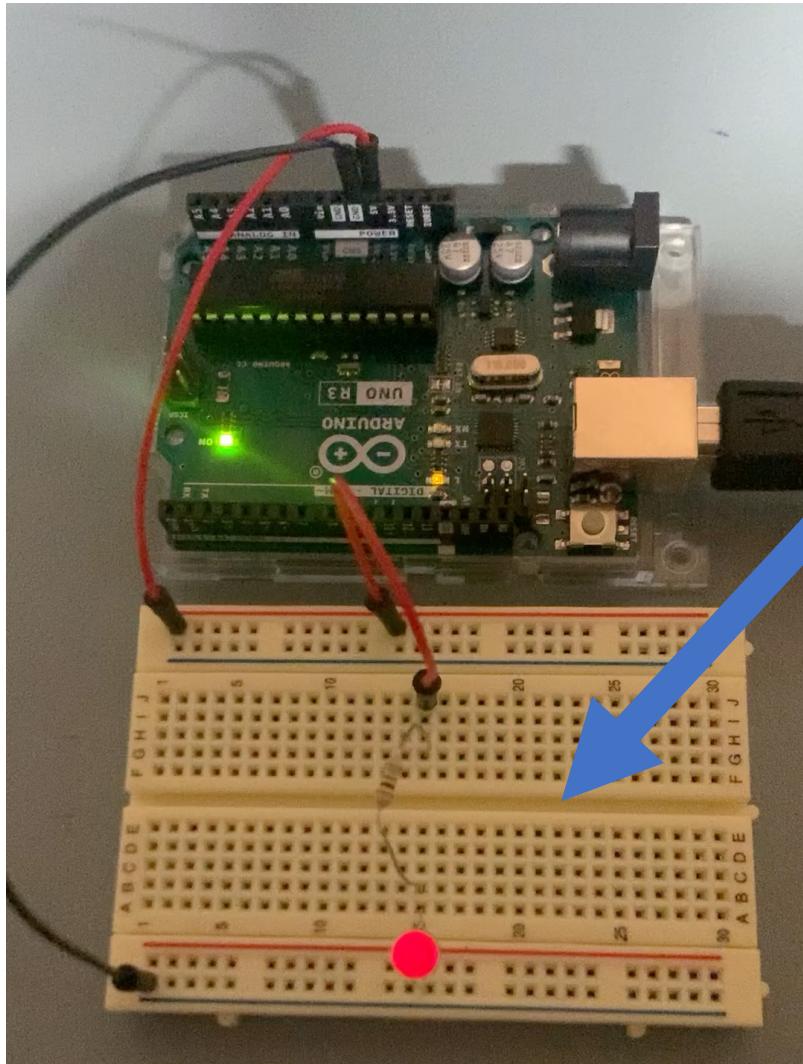
Room lights darkened;  
LEDS look much  
brighter.



## Engineering Malpractice

This LED is damaged. It glows intermittently.

Suppose we return this damaged LED to our parts drawer and re-use it....



Re-using a damaged LED.

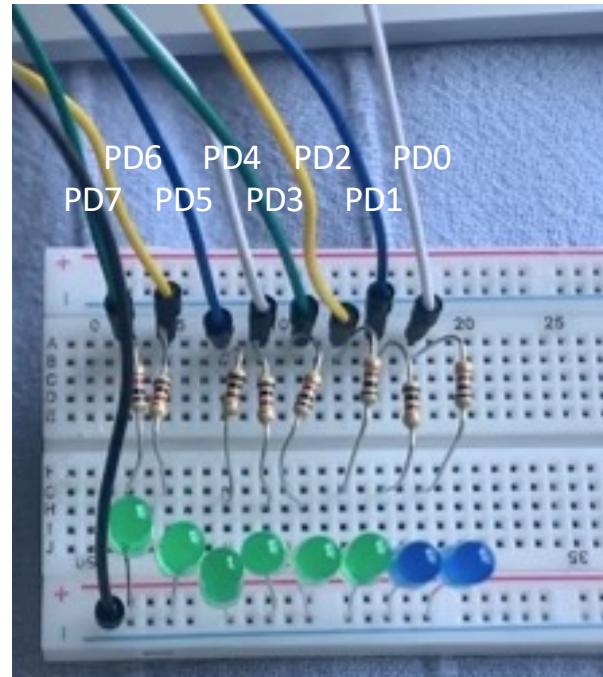
It should be on.  
But it blinks intermittently.  
Imagine using this damaged  
LED to diagnose an  
embedded system

## 8 bit binary counter

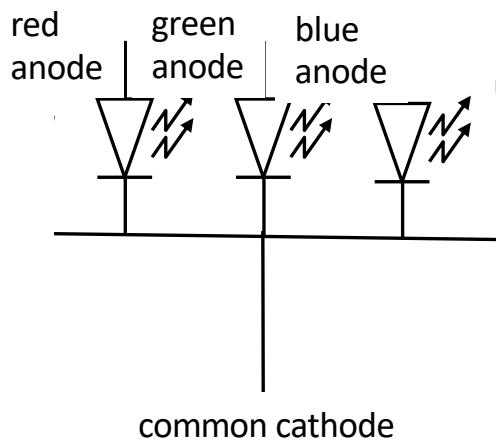
Note: your textbook typically has this line at the top of the code. This tells the compiler the frequency of the clock driving the ATmega328P. (The ATmega328P doesn't know time; it just counts CPU cycles.) We have the line -DF\_CPU 16000000 in makefile that defines this.

```
*/  
  
#define F_CPU 16000000UL  
#include <util/delay.h>  
#include <avr/io.h>  
  
int main(void)  
{  
    unsigned char i=0;  
    DDRD = 0xFF;  
    while (1) {  
        i++;  
        PORTD=i;  
        _delay_ms(100);  
    }  
}
```

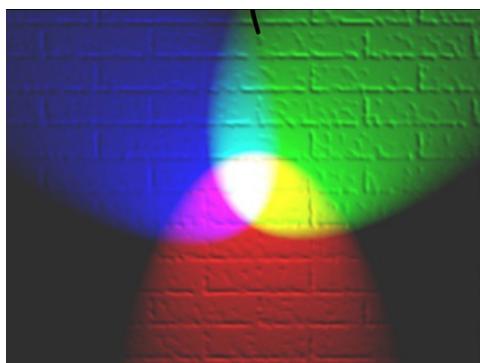
8 LEDs connected to the pins of port D



## Common Cathode RGB LED



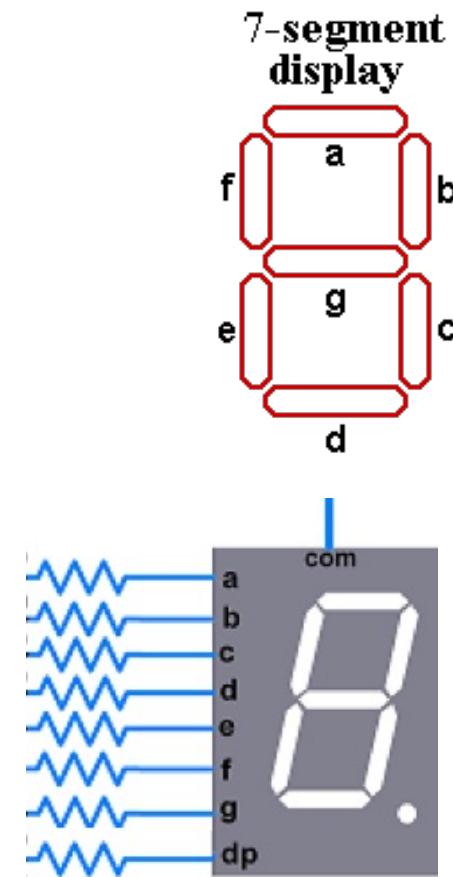
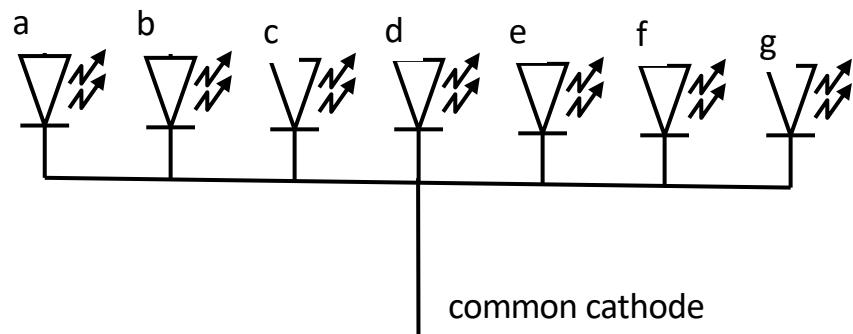
red + green = yellow  
red + blue = magenta  
green + blue = cyan  
red + green + blue = white



red, green, blue  
anodes (+)

common  
cathode (-)

## Common Cathode 7 Segment LED



some also have a decimal point as an 8<sup>th</sup> segment

Turn on LEDs when buttons are pressed

left button → red LED  
right button → green LED

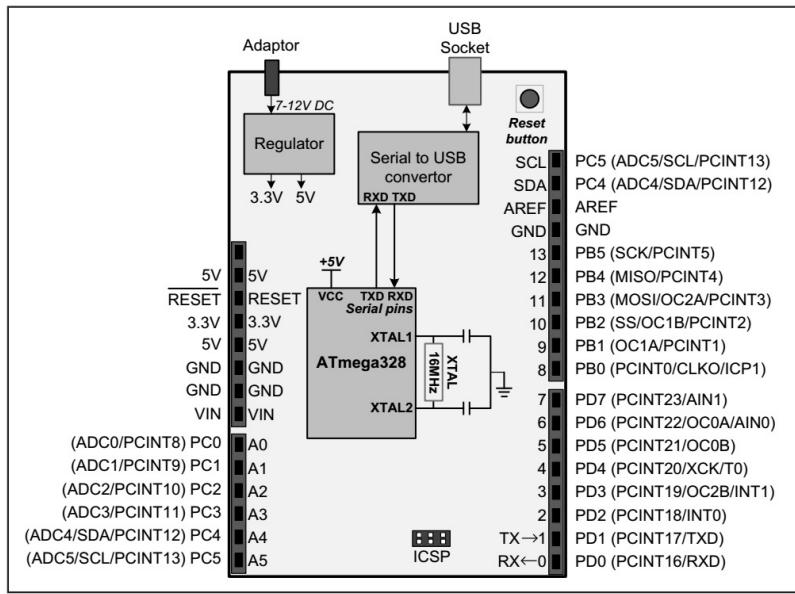
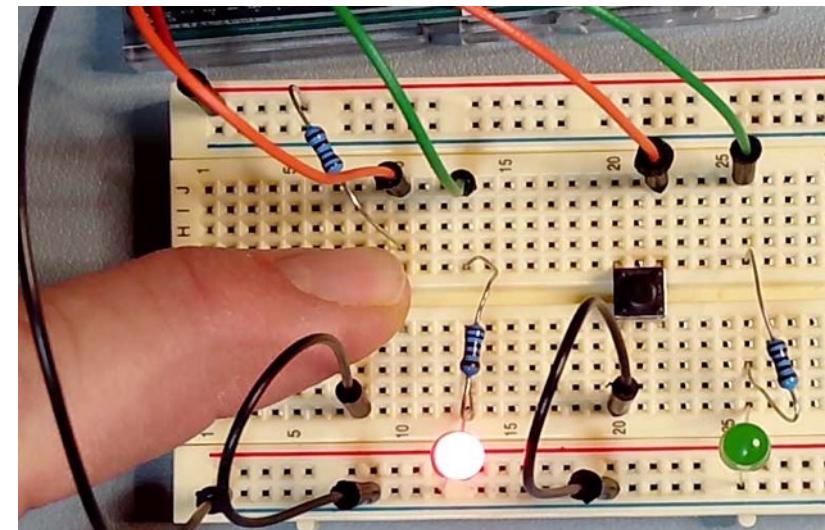
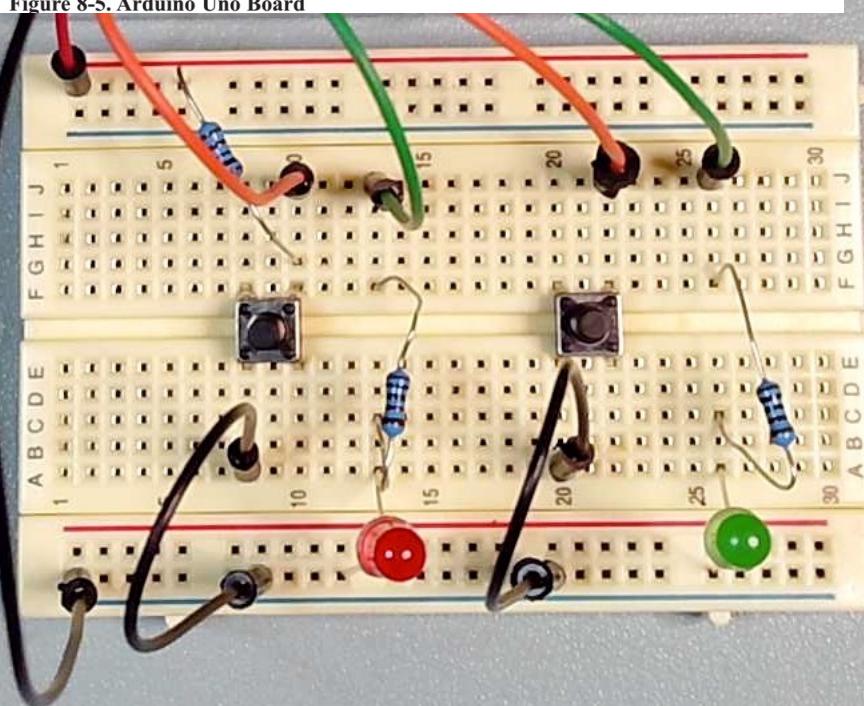
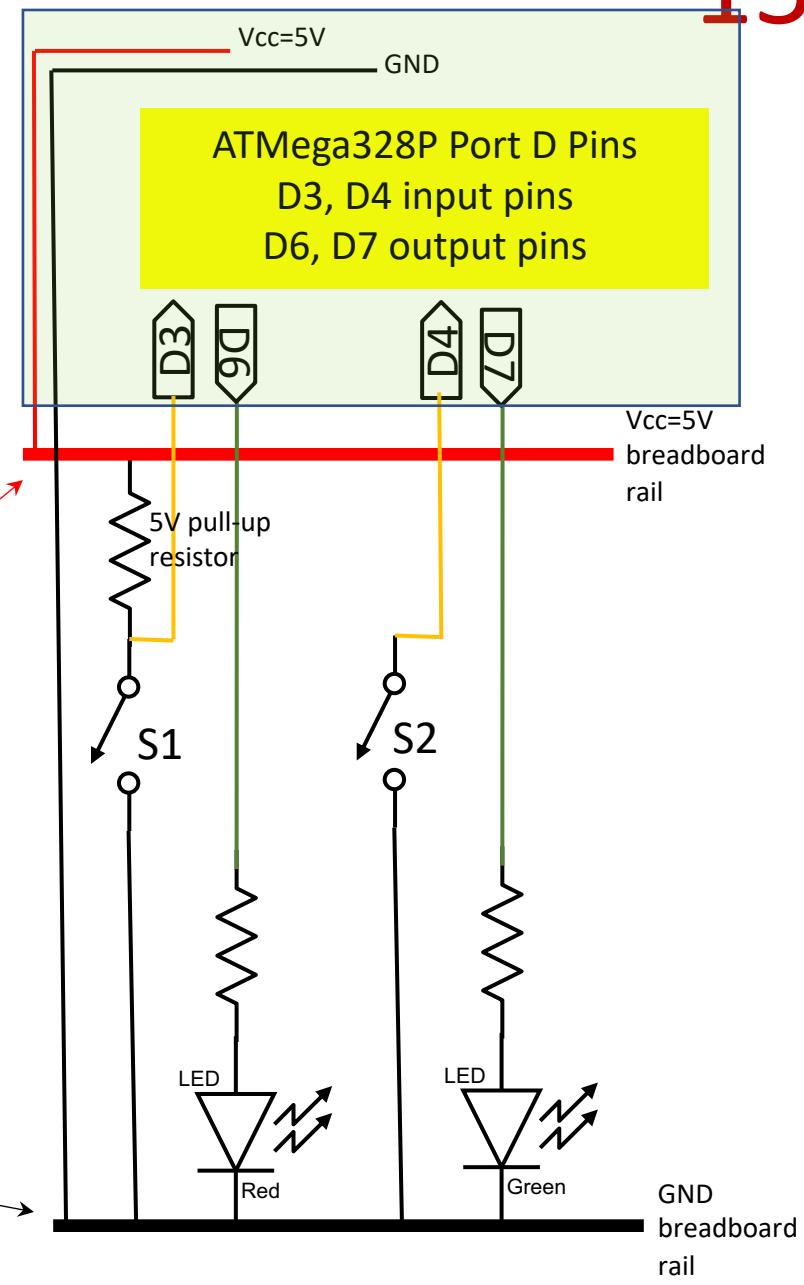
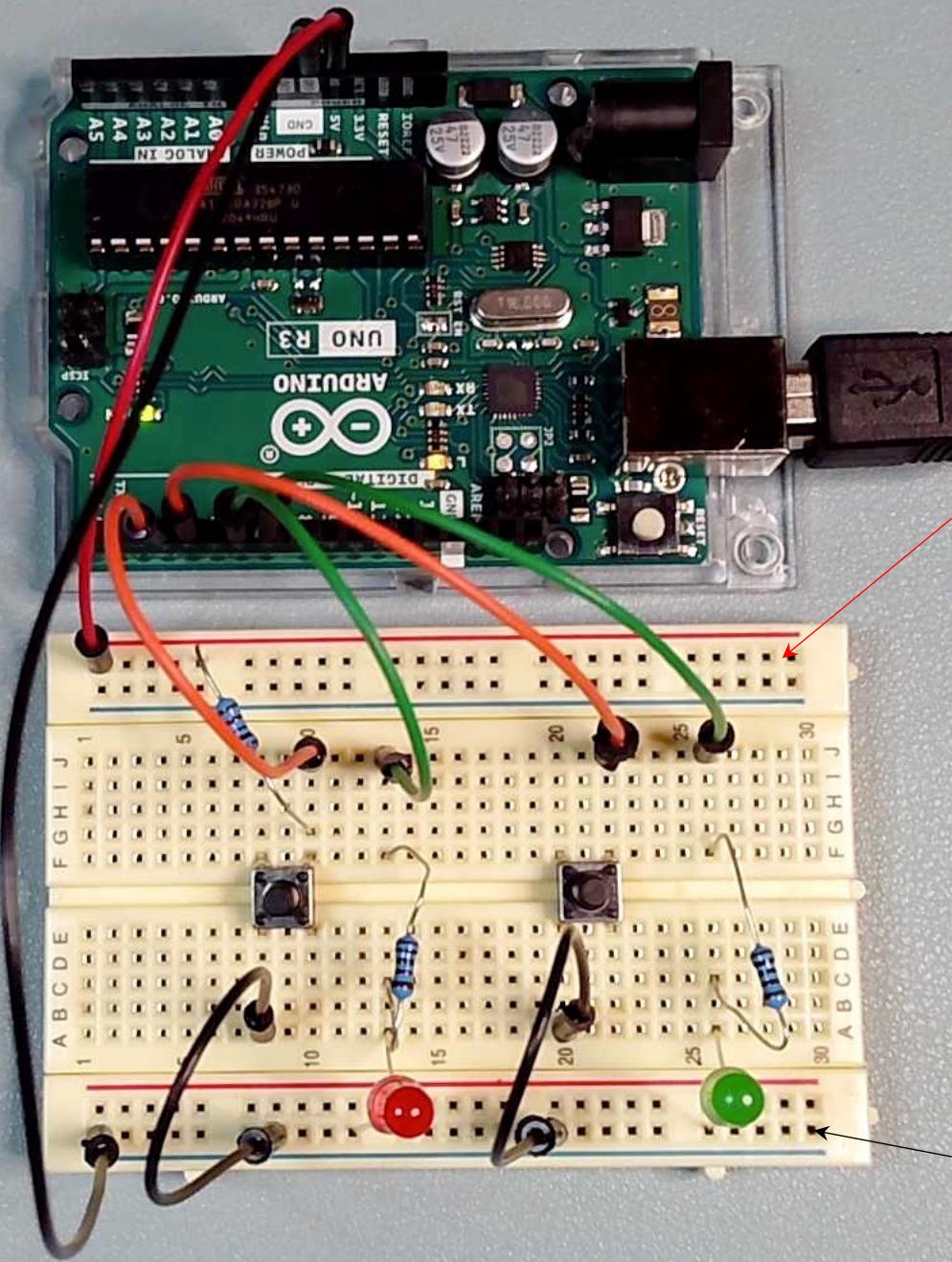


Figure 8-5. Arduino Uno Board



*The code for this app is included at the end of this deck of slides.*



## Setting the pin direction:

```
DDRD = 0x11000000;
OR
DDRD = 1<<DDD6|1<<DDD7; //All others input
OR
DDRD = 1<<6|1<<7;
```

Why does this work?

$$\begin{aligned}1 << 6 &= 0000\ 0001 \ll 6 = 0100\ 0000 \\1 << 7 &= 0000\ 0001 \ll 7 = 1000\ 0000 \\ \text{So, } 1 << 6 | 1 << 7 &= 1100\ 0000\end{aligned}$$

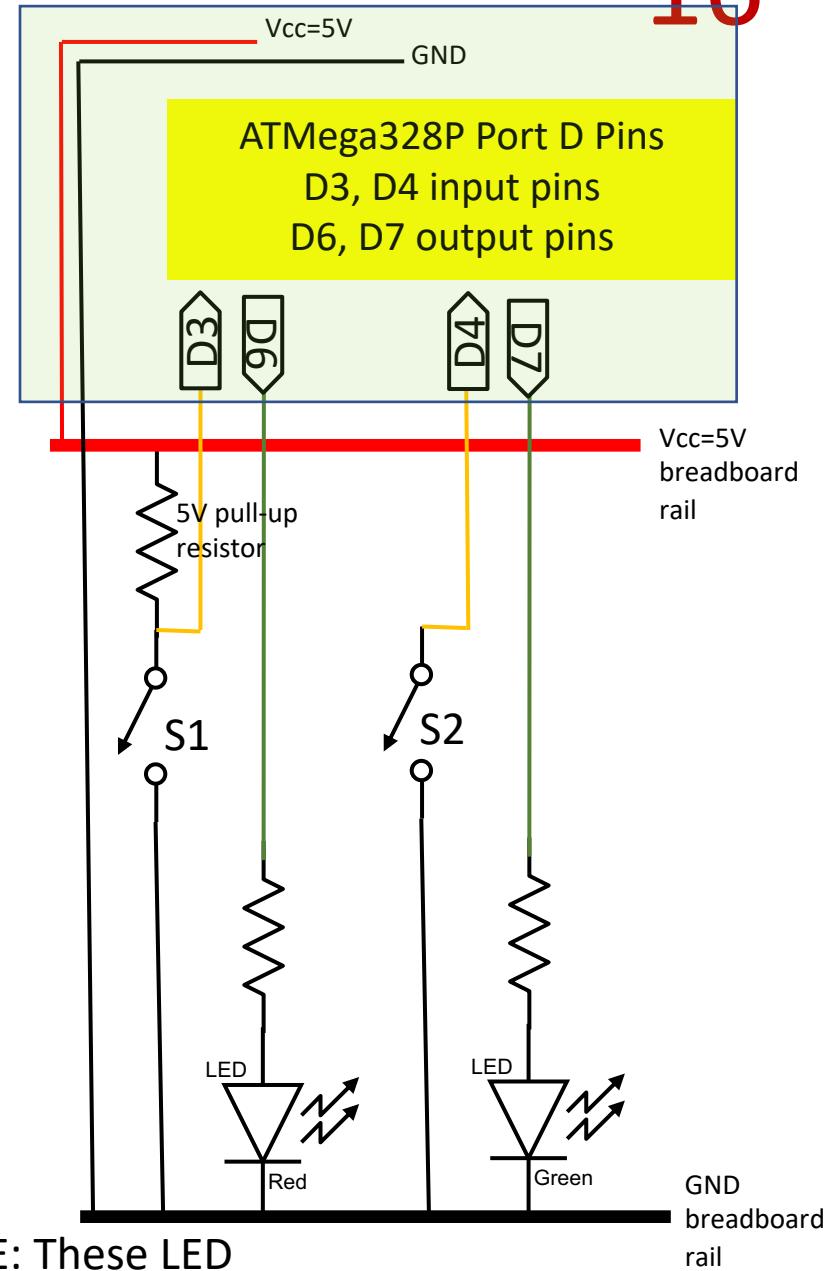
## Turn the LEDs on or off:

```
PORTD |= 1<<PORTD6; //D6 high; red LED on
PORTD &= ~(1<<PORTD6); //D6 low; red LED off
```

Why does this work?

$$\begin{aligned}\sim(1 << \text{PORTD6}) &= \sim(0000\ 0001 \ll 6) \\&= \sim(0100\ 0000) \\&= 1011\ 1111\end{aligned}$$

$\text{PORTD} = \text{PORTD} \& 1011\ 1111$



Turn Green LED on & off similarly via PORTD7. NOTE: These LED instructions leave the other bits of PORTD alone!

## Reading the input pins:

we will get to this in a few slides...

## Switches, wires, voltages, and pin values

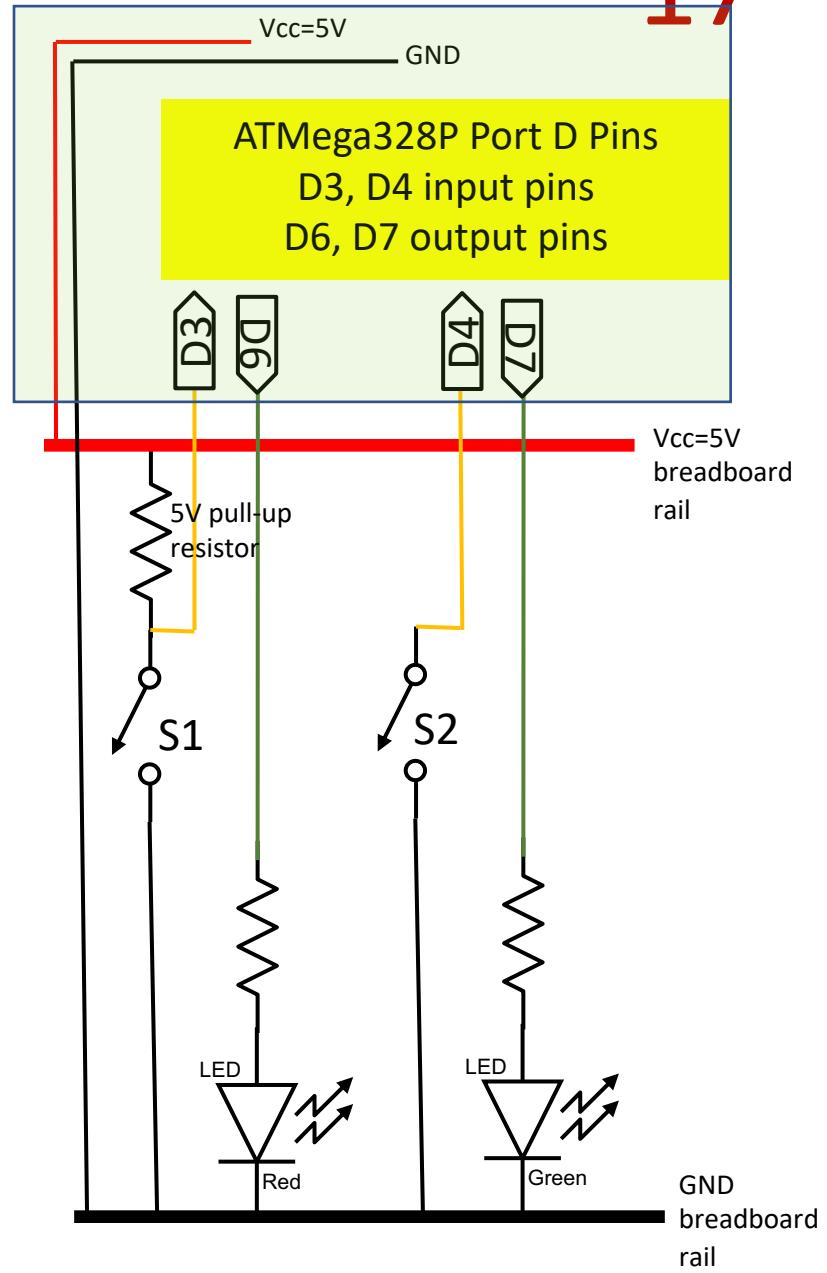
Let's see how the buttons make D3 & D4 high and low ...

S1	D3	S2	D4
Open	5V	Open	floating*
Closed	GND	Closed	GND

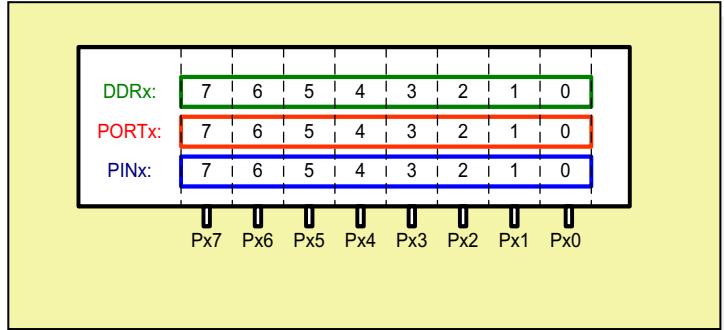
External pull-up resistor on S1 ensures that D3=5V when switch is open.

No external pull-up resistor on S2 means D6 is floating\* when switch is open

\* Depends on the input circuit for D4



# The structure of I/O pins & registers



we access the ports by writing to and reading from these registers.

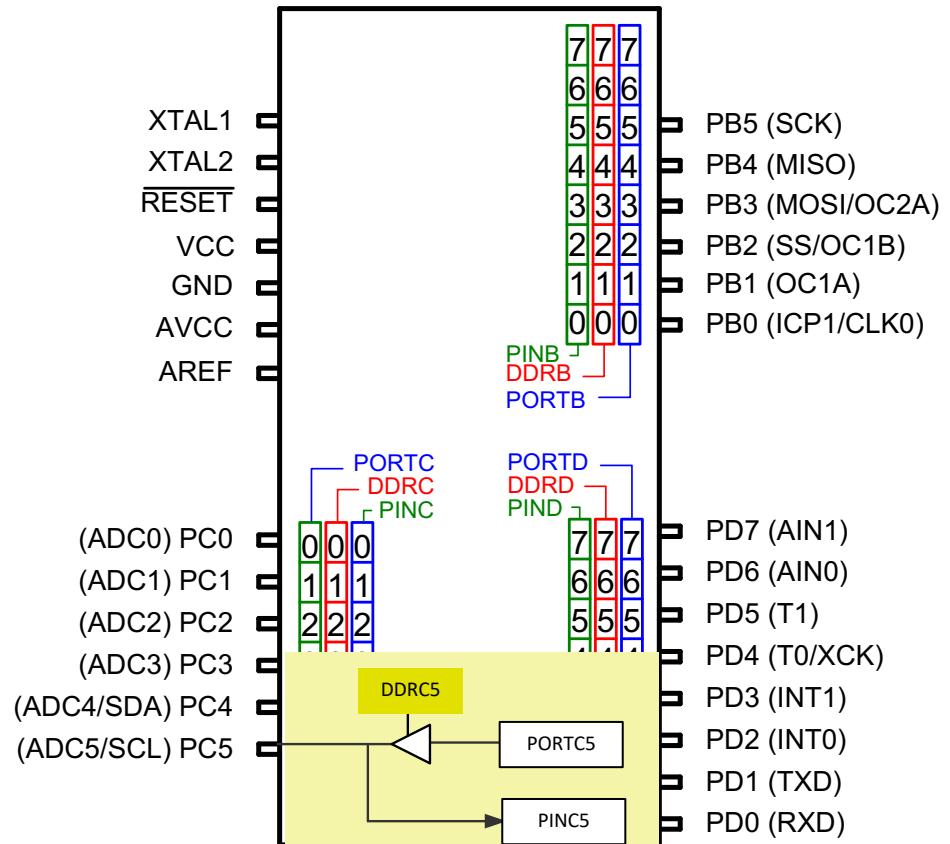
Example:

```
DDRB = 0xFF; //set as outputs
```

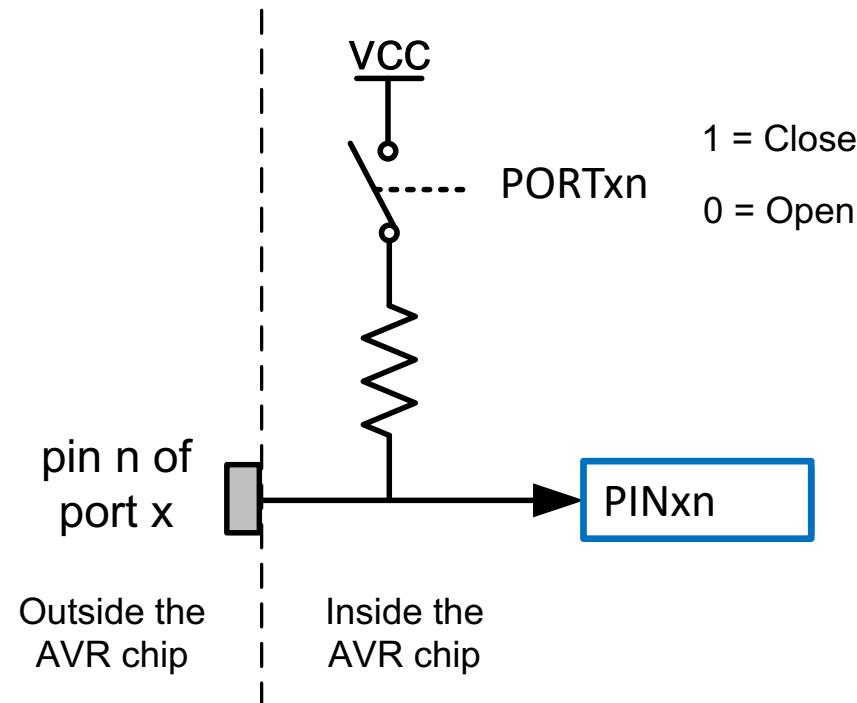
```
DDRB = 0x00; //set as inputs
```

```
PORTB = 0x00; //set pins low
```

```
input_value = PINC; //read pins
```

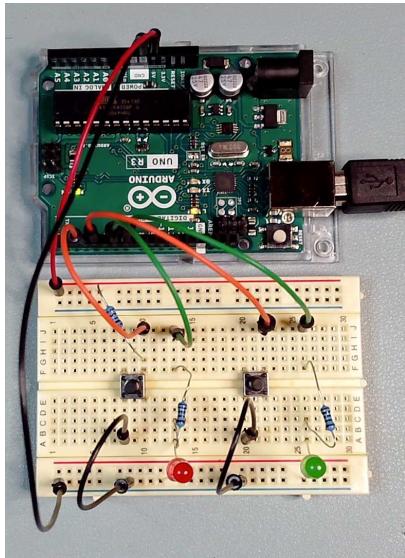


# Input pins & pull-up resistors



DDRXn = 0 (pin set as input)		
PORTxn	Pxn (input voltage)	PINxn (input logic)
0, high Z	0V	0
0, high Z	5V	1
0, high Z	float	don't use
1, pullup	0V	0
1, pullup	5V	1
1, pullup	float	1

# Example: Port D, pin 3

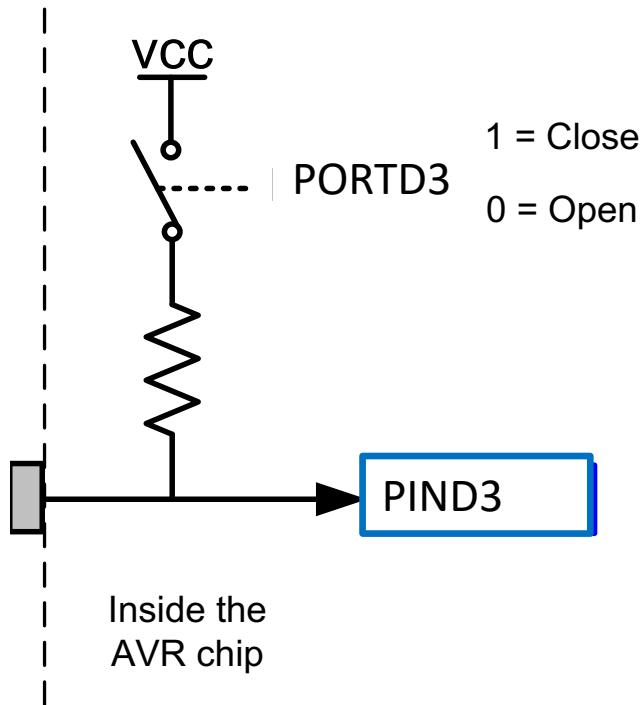


`DDRD3 = 0` (pin set as input)

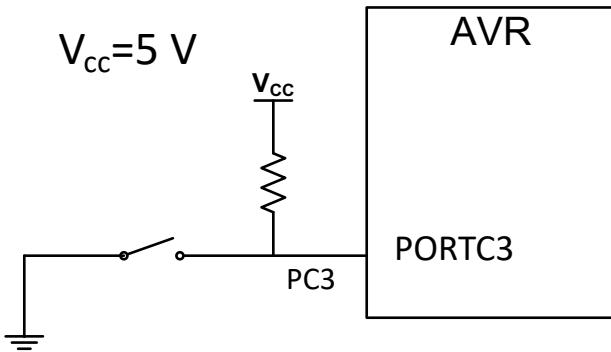
<code>PORTD3</code>	<code>PD3</code> (input voltage)	<code>PIND3</code> (input logic)
0, high Z	0V	0
0, high Z	5V	1
0, high Z	float	don't use
1, pullup	0V	0
1, pullup	5V	1
1, pullup	float	1

pin 3 of port  
D (PD3)

Outside the  
AVR chip

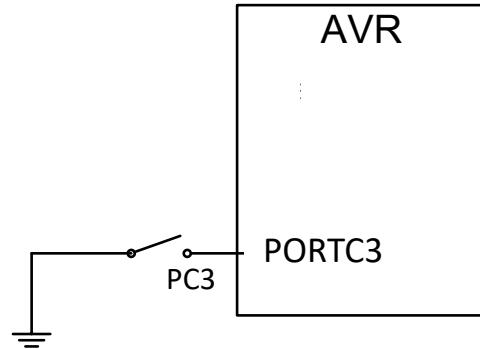


# Momentary SPST switch input (1)



**External Pull-up (DDRC3=0; PORTC3=0)**

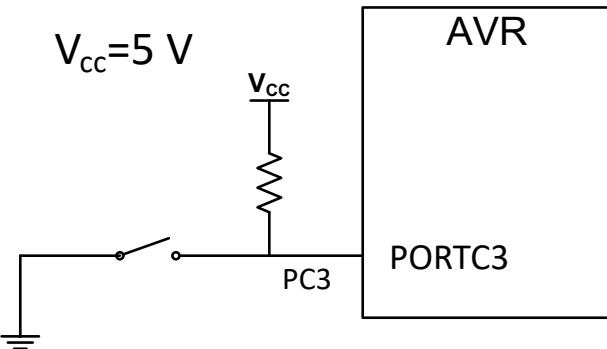
Switch Closed	PC3 = 0V	PINC3 = 0
Switch Open	PC3 = 5V	PINC3 = 1



**No External Pull-up (DDRC3=0; PORTC3=0)**

Switch Closed	PC3 = 0V	PINC3 = 0
Switch Open	PC3 = floating	PINC3 = ?

# Momentary SPST switch input (2)



**External Pull-up (DDRC3=0; PORTC3=0)**

Switch Closed

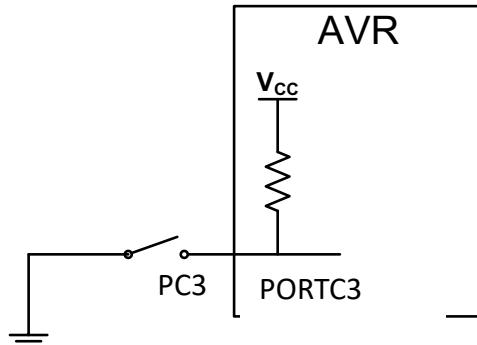
PC3 = 0V

PINC3=0

Switch Open

PC3 = 5V

PINC3=1



**Internal Pull-up (DDRC3=0; PORTC3=1)**

Switch Closed

PC3 = 0V

PINC3=0

Switch Open

PC3 = 5V

PINC3=1

These are not C assignment statements. Our C compiler does not allow for bit-level assignments. We use bit-level manipulation instead.

```
DDRC &= ~(1<<DDC3); //makes DDRC3=0; other bits of DDRC not altered
PORTC |= (1>>PORT3); // makes PORTC3=1; other bits of PORTC not altered
```

# Checking a bit in a Byte

Let's check if bit 5 of a particular byte is high...

Bit-by-bit AND with a mask  
0010 0000 (1 in bit 5 position, 0 elsewhere)

Result is 0 if bit 5 is 0.

Result is non-zero if bit 5 is 1.

- We can use & operator to see if a bit in a byte is 1 or 0

XXXX XXXX

& 0010 0000

-----  
00x0 0000

XXXX XXXX

& (1 << 5)

-----  
00x0 0000

Let's apply this logic to testing the status (1 or 0) of PINC5

```
if( ((PINC & 0b00100000) != 0) // Test if bit 5 of PINC is high
```

```
if( ((PINC & (1<<5)) != 0) // Same test using shift operator for mask
```

```
if( ((PINC & (1<<PINC5)) != 0) // Same test using PINC5 instead of 5
```

```
if( ((PINC & (1<<PINC5)) == 0) // Test if PINC5 is low
```

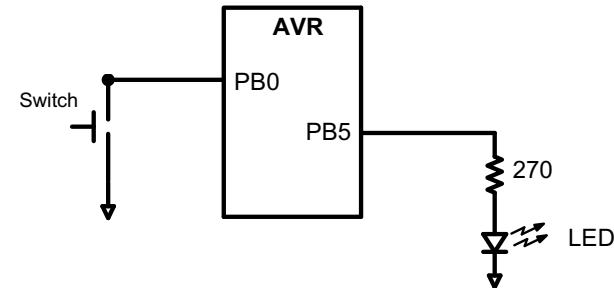
why don't we do this: `if( ((PINC & (1<<PINC5)) == 1)`

Assume PINC5=1. Then PINC & (1<<PINC5) = XX1XXXXX & 00100000 = 00100000 = 32

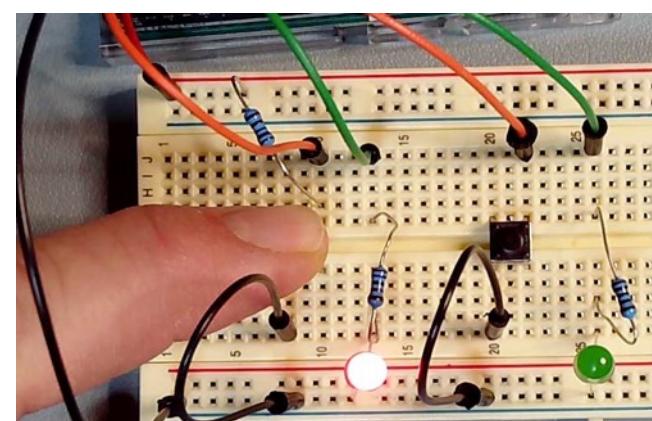
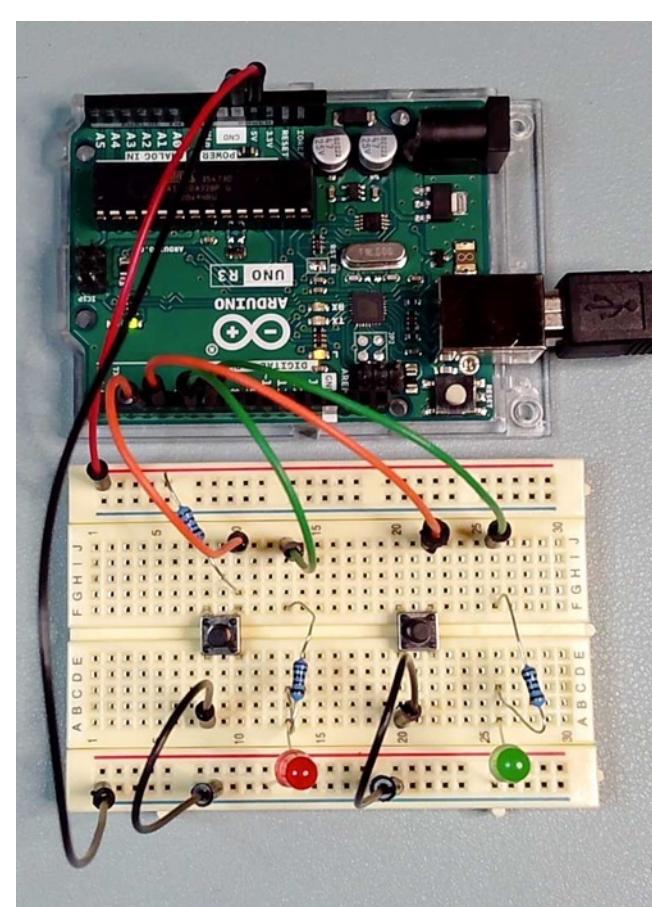
# Example

A switch without external pullup is connected to pin PB0. An LED is connected to pin PB5. Write a program segment to turn on the LED when the switch is pressed.

logic: PB5 = ~ PB0



```
DDRB = (1<<DDB5); // PB5 output; other PORTB pins inputs
PORTB = (1<<PORTB0); // Activate internal pullup on PB0
while(1)
{
    if ( (PINB & (1<<PINB0) ) == 0) // Is PB0=0 (button pressed?)
        PORTB |= (1<<PORTB5); // Make PB5 high; LED on
    else
        PORTB &= ~ (1<<PORTB5); // Make PB5 low; LED off
}
```



switches2.c

```
C switches2.c 3 X
Users > davemclaughlin > Documents > codingProjects > embedded > blinkswitch > C s
● 1  /* switches2.c This code turns on LEDs when momentary pushbutton
2  switches are pressed. Input pins on PD3 and PD4. LEDs on PD6 on PD7
3  D. McLaughlin 2/21/22 */
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7
8  int main(void){
9      DDRD = 1<<DDD6|1<<DDD7;           // Set D6, D7 as output
10     PORTD = 1<<PORTD3|1<<PORTD4;       // Set pullup on D3 & D4
11
12     while(1){
13
14         if ((PIND & (1<<PIND3)) == 0){ // PIND3 is 0 when pressed
15             PORTD |= (1<<PORTD6);        // P6 high when P3 is pressed
16         } else {
17             PORTD &= ~ (1<<PORTD6);       // P6 low when P3 not pressed
18         }
19
20         if ((PIND & (1<<PIND4)) == 0){ // PIND4=0 when pressed
21             PORTD |= (1<<PORTD7);        // P7 high when P4 is pressed
22         } else {
23             PORTD &= ~ (1<<PORTD7);       // P7 low when P4 not pressed
24         }
25
26     }
27     return(0);
28 }
29
30 **** End of File ****
31
32
33
```

Internal pull-ups activated, so external pull-up resistors not needed

## Summary: Set, Clear, Toggle, and test a single bit:

**Consider PIN 4 of PORTB, corresponding to hardware pin PB4**

### Set PB4 (make 5V output voltage on PB4)

```
PORTB = 0xFF;           // Sets all bits high. This is brute force!
PORTB = PORTB | 0b00010000; // Sets bit 4 high; other pins unaltered. Difficult to read.
PORTB = PORTB | (1<<4); // Set PORTB4 high; other pins unaltered
PORTB |= (1<<4);      // Set PORTB4 high; other pins unaltered
PORTB |= (1<<PORTB4); // Set PORTB4 high; other pins unaltered
```

PORTB: 

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

### Clear PB4 (make 0V output voltage on PB4)

```
PORTB = PORTB & 0b11101111; // Clear PORTB4 using a mask
PORTB &= 0b11101111;       // Same as above
PORTB &= ~(1<<4);        // ~(1<<4) = ~(00010000) = 11101111
PORTB &= ~(1<<PORTB4)    // Easiest to understand
```

### Toggle PB4 (5V to 0V; 0V to 5V)

```
PORTB = ~ PORTB;          // Toggles all bits of PORTB. This is brute force!
PORTB = PORTB ^ 0b00010000; // Toggles bit 4 of PORTB
PORTB ^= 0b00010000;      // Toggles bit 4 of PORTB using shorthand
PORTB ^= (1<<4);        // Another way of writing previous statement
PORTB ^= (1<<PORTB4);   // Easiest to understand,
```

### Test (determine if input on PB4 is 0V)

```
if ( (PINB & 0b00010000) == 0) // Test if PINB4 is equal to zero
  if ( (PINB & (1<<4)) == 0) // Same as above
    if ((PINB & (1<<PINB4)) == 0) // Same as above. Supposedly easier to understand
```