Getting input into the chip

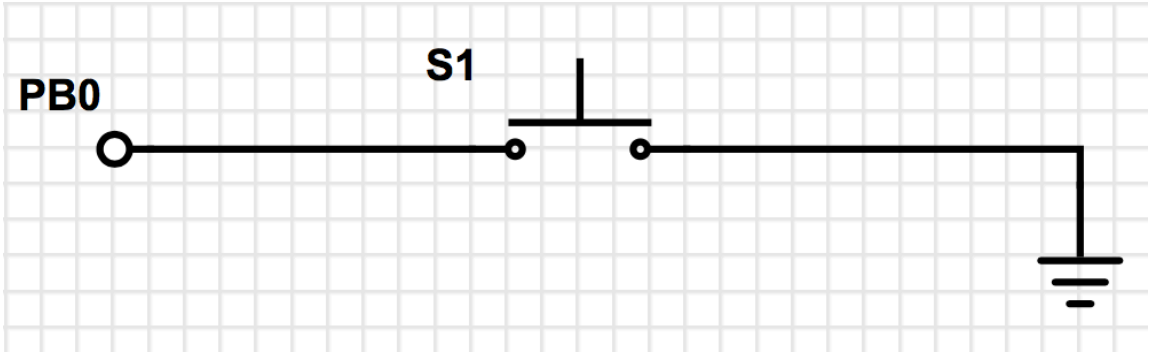
Set the direction

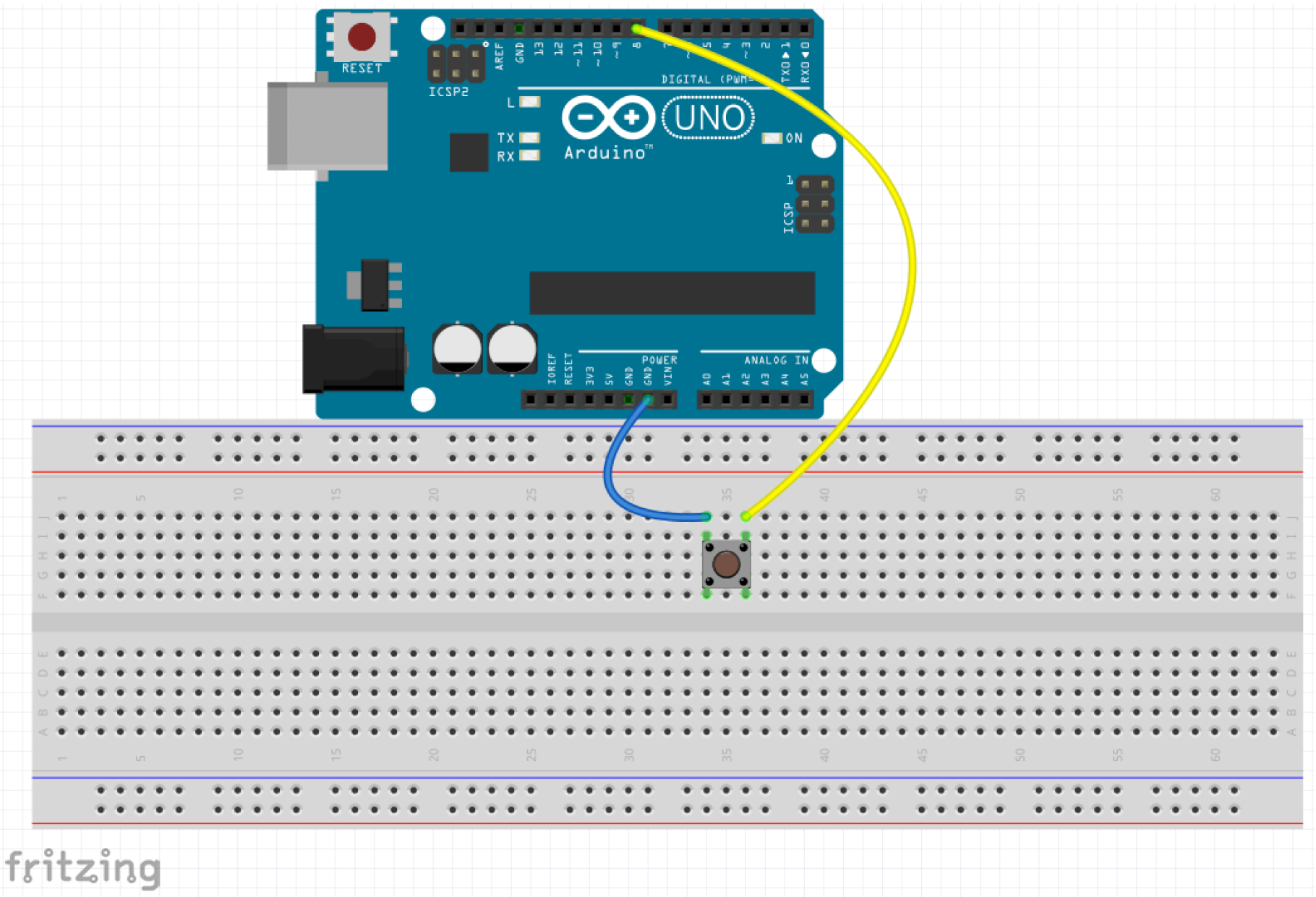
- To make a pin an input pin put a zero in the corresponding location in the DDRX register
- DDRB &= ~(1<<DDB0); //Make PB0 an input pin
- All pins are input pins by default

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	**DDB7**	**DDB6**	**DDB5**	**DDB4**	**DDB3**	**DDB2**	**DDB1**	**DDB0**	**DDRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Connect something!

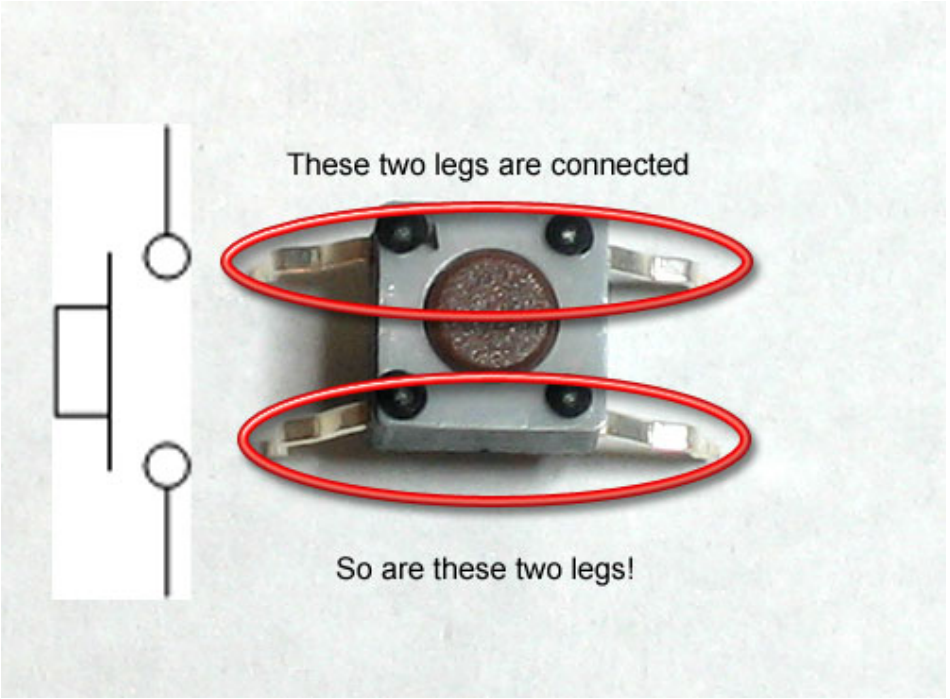
- One of the simplest things to connect is a switch or *momentary push button*.
- Usually connected to GND… will see why shortly




RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
Arduino™
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
fritzing

How does the button “work”

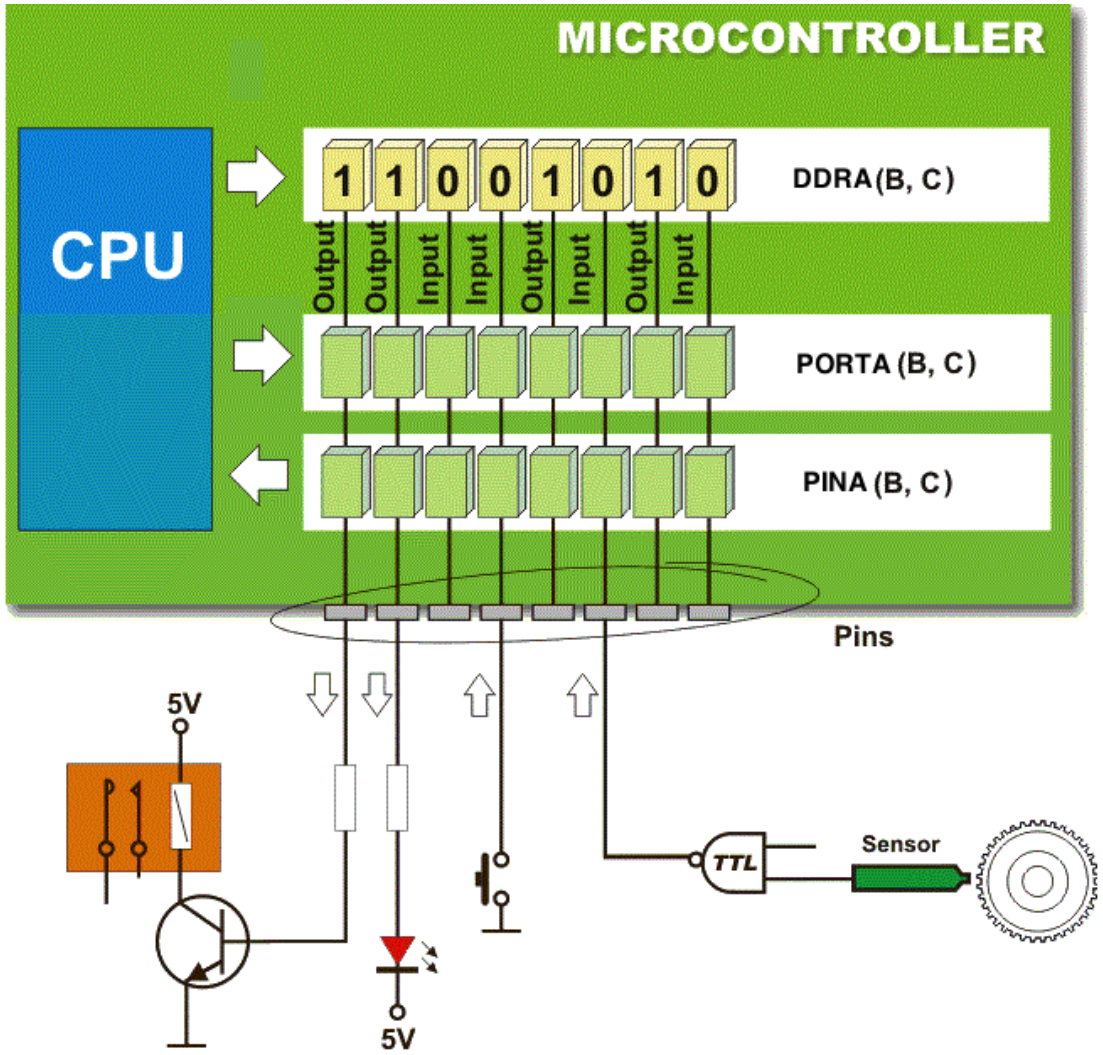
- Top two legs are connected when button is NOT pressed
  - Similar for bottom two legs
- Pressing the button results in a connection between all four legs
- Another way to think of it
  - When button is pressed the top “line” is connected to the bottom “line”



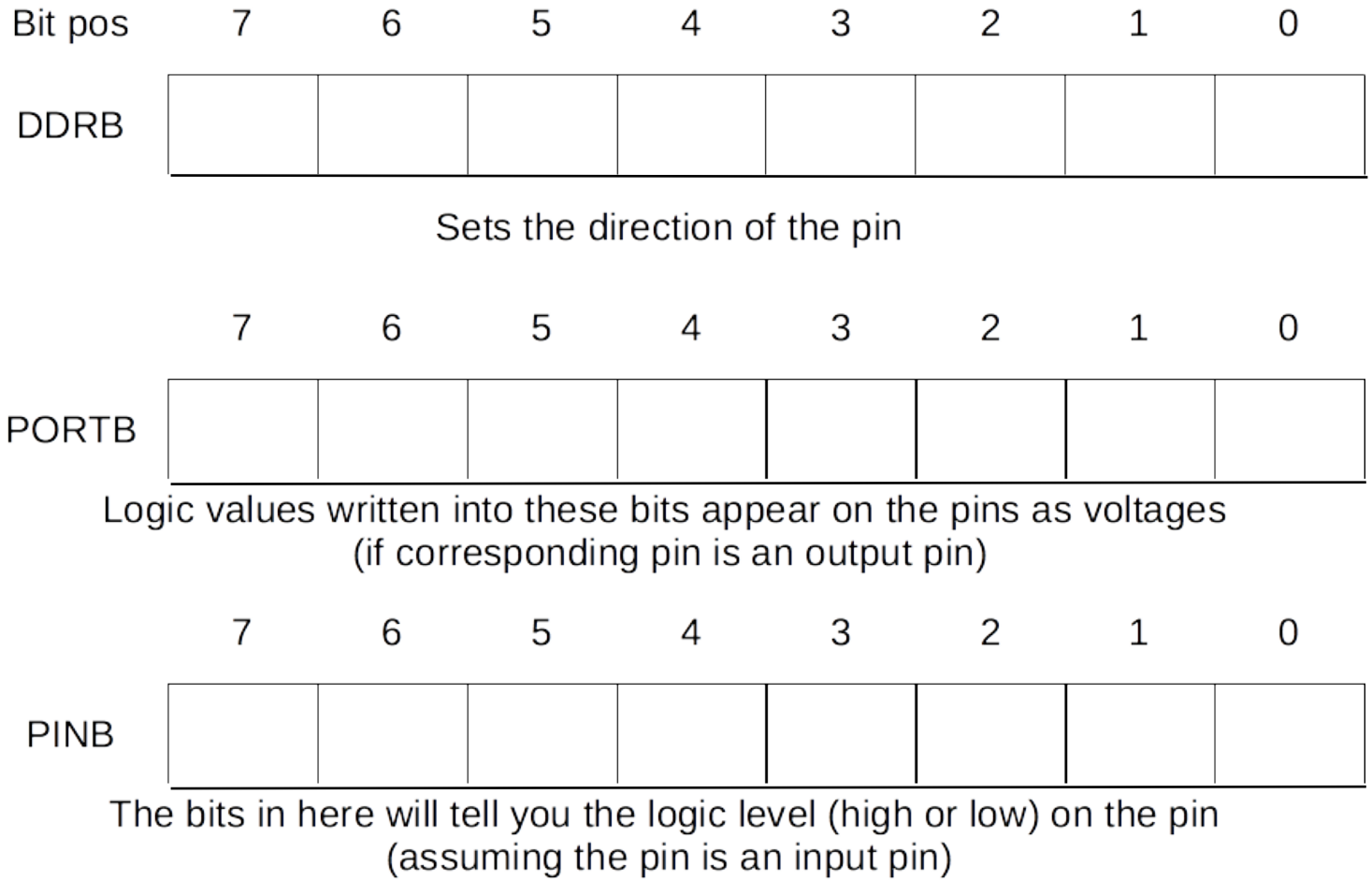
Reading in the logic level at a pin

- No we have the pin (PB0) set up as an input pin how do we “read” the logic level present at the pin?

Remember this?



And this?


Bit pos
7
6
5
4
3
2
1
0
DDRB
Sets the direction of the pin
7
6
5
4
3
2
1
0
PORTB
Logic values written into these bits appear on the pins as voltages
(if corresponding pin is an output pin)
7
6
5
4
3
2
1
0
PINB
The bits in here will tell you the logic level (high or low) on the pin
(assuming the pin is an input pin)

PINB

- PINB is an 8-bit register where the value of each bit represents the logic level at the corresponding pin.
- This is a read-only register

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Reading PINB

- Reading PINB is easy
  - x = PINB;
- We just read PINB and copied its contents into x.
- So how do we check if a bit is set or not in x or even directly in PINB?
- Shifting and masking.

What if x (PINB) contained 0101010?

- Can't just do:
  - `if(x == 1) or if(x == 0)`
  - 1 = 00000001
  - 0 = 00000000
- If x contained 0101010?
  - Would have to do either:
  - `if(x == 0x55)` – bit 0, the ?, is 1
  - `if(x == 0x54)` – bit 0, the ?, is 0
- Trouble is we don't even know what is in x
  - XXXXXXX?

Shifting & Masking to read a bit

- Assume PINB contains XXXXXXX?
  - ? represents the bit we are interested in – is it 0 or 1?
- Create a mask with a “1” in the right place
- `(1<<0)` - 00000001

- We’re not interested in all the X bits so lets clear them to 0

```
x = PINB;
x &= (1<<0);
```

```
XXXXXXX?
00000001   &
--------
0000000?
```

- x now contains 0000000?

Shifting & Masking to read a bit

- x now contains 0000000?
- So now we can do:
  - `if(x == 1) or if(x == 0)`
  - 1 = 00000001
  - 0 = 00000000

What if we're not dealing with bit 0?

- What if we're talking about PB3 and not PB0?
- The bit in PINB we're interested in now is bit 3

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	**PINB7**	**PINB6**	**PINB5**	**PINB4**	**PINB3**	**PINB2**	**PINB1**	**PINB0**	**PINB**
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

What if we're not dealing with bit 0?

- So now PINB contains
  - XXXX?XXX
- So mask is:
  - 00001000
  - (1<<3)
- So this doesn't work any more…

```
x = PINB;    //x contains XXXX?XXX
x &= (1<<3); //x contains 00001000 or 00000000
if(x == 1) or if(x == 0)
```

What if we're not dealing with bit 0?

- Need to do

```
x = PINB;     //x contains XXXX?XXX
x &= (1<<3); //x contains 00001000 = 8 or 00000000 = 0
if(x == 8) or if(x == 0)
```

- So we need to compare against different numbers for different bit positions…

What if we're not dealing with bit 0?

- Bit 0 – 0b00000001 – 0x01 – 1…
- Bit 1 – 0b00000010 – 0x02
- Bit 2 – 0b00000100 – 0x04
- Bit 3 – 0b00001000 – 0x08
- Bit 4 – 0b00010000 – 0x10
- Bit 5 – 0b00100000 – 0x20
- Bit 6 – 0b01000000 – 0x40
- Bit 7 – 0b10000000 – 0x80

Options?

```
x = PINB;           //x contains XXXX?XXX
x &= (1<<3);        //x contains 00001000 = 8
                    //or x contains 00000000 = 0
if(x == (1<<3))…
```

```
x = PINB;           //x contains XXXX?XXX
x &= (1<<3);        //x contains 00001000 = 8
                    //or x contains 00000000 = 0
if((x>>3) == 1)…
```

A more elegant solution?

```
x = PINB;
if(x & (1<<3) != 0)
{
    //bit 3 is a one
}
```

A more elegant solution?

- In C programming
  - “FALSE” means 0
  - “TRUE” means any number not 0.
    - e.g. -1 is true, 9 is true, 0.00001 is true

```
x = PINB;
if(x & (1<<3))
{
   //bit 3 is a one
}
```

```
XXXX?XXX
00001000   &
--------
0000?000

00001000 = 8 = true
00000000 = 0 = false
```

Applies to any bit position

- Bit 0 – 0b00000001 – true
- Bit 1 – 0b00000010 – true
- Bit 2 – 0b00000100 – true
- Bit 3 – 0b00001000 – true
- Bit 4 – 0b00010000 – true
- Bit 5 – 0b00100000 – true
- Bit 6 – 0b01000000 – true
- Bit 7 – 0b10000000 – true

Works for any bit position

```
x = PINB;
if(x & (1<<5))
{
   //bit 5 is a one
}
```

```
XX?XXXXX
00100000   &
--------
00?00000

00100000 = 32 = true
00000000 = 0 = false
```

- Notice x doesn't change…
- The if statement checks the *result* of the calculation
- Get rid of x and just use PINB...

A more elegant solution

```
if(PINB & (1<<5))
{
   //bit 5 is a one
}
```

```
XX?XXXXX
00100000   &
--------
00?00000
```

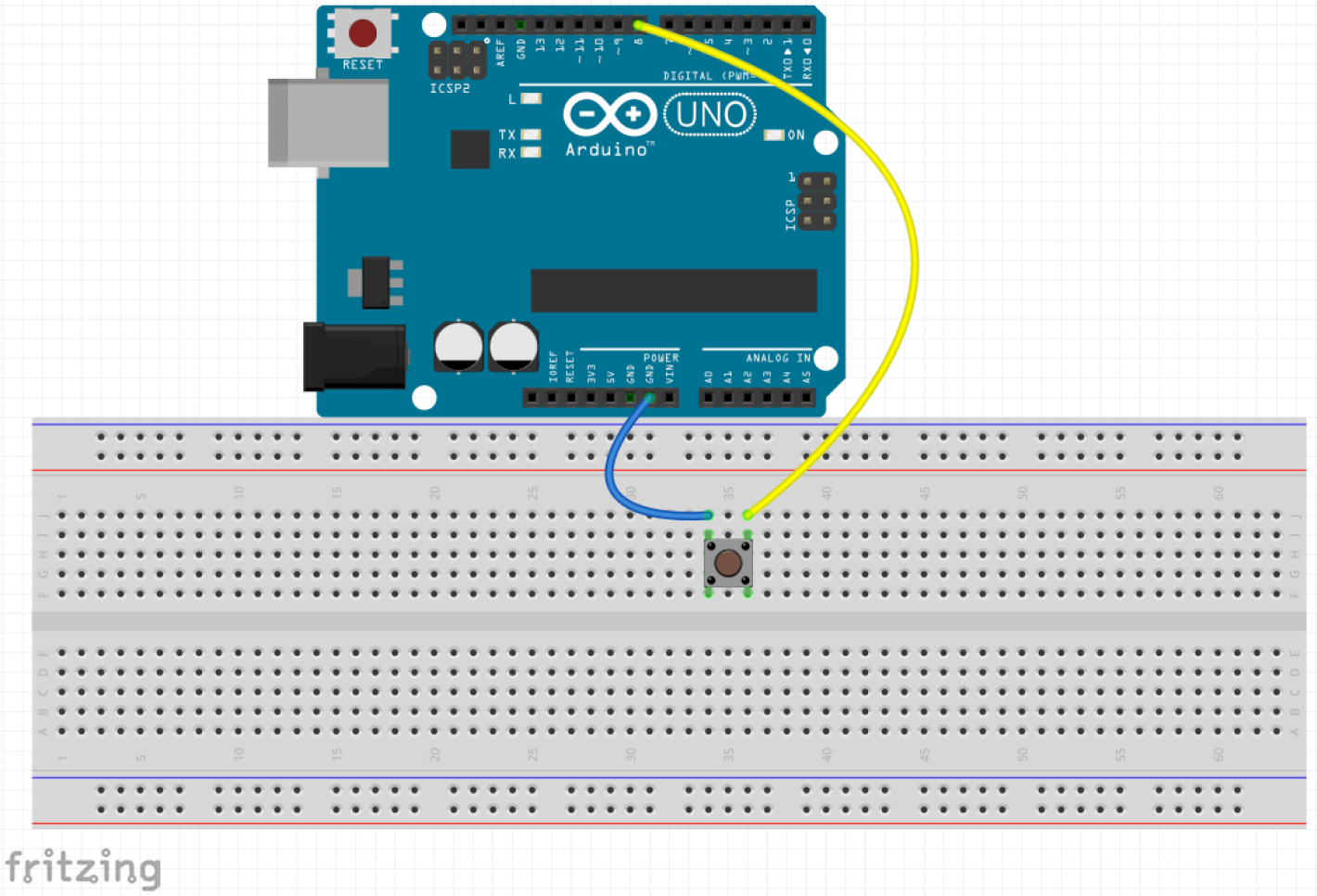
Works for any register

- Need to know if bit Y in REGX is a 1?

```
if(REGX & (1<<Y))
{
   //bit Y is a one
}
```

Is the button pressed?

```
if(PINB & (1<<0))
{
   //bit 0 is a one
   //switch on LED
   PORTB |= (1<<5);
}
else
{
   //bit 0 is a 0
   //switch off LED
   PORTB &= ~(1<<5);
}
```



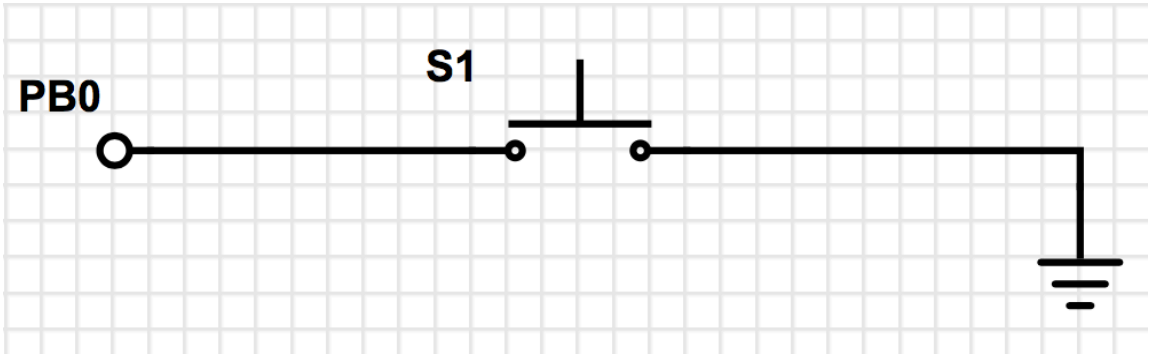
Doesn’t work!

- Aaaargh!
- We’re missing something fundamental…

Floating voltage?

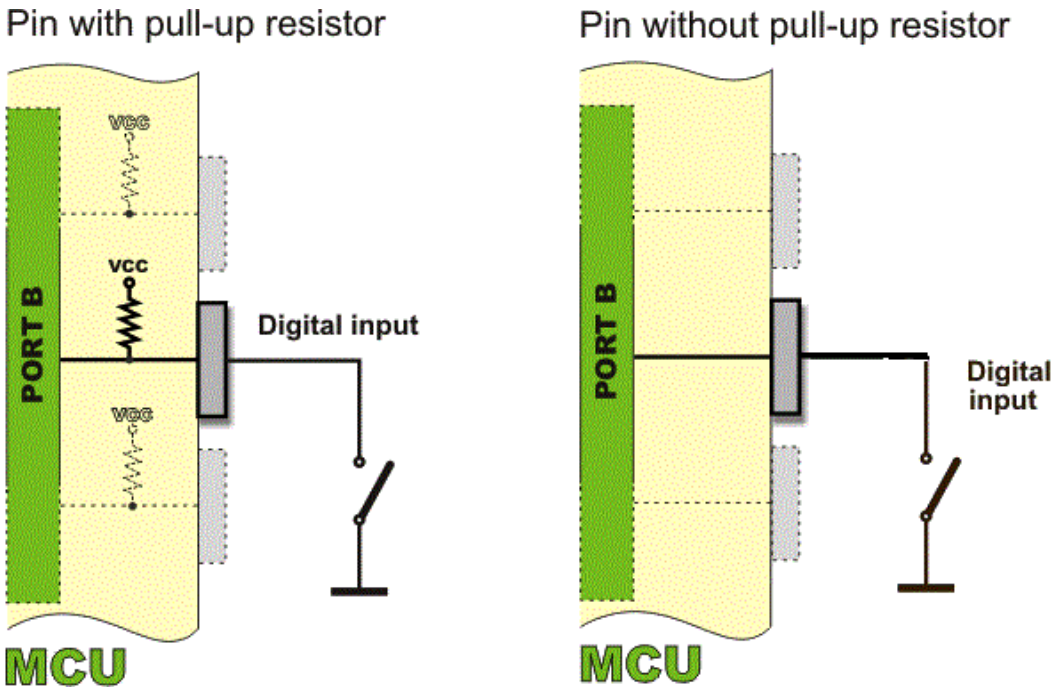
- So what voltage is at PB0 when the button is un-pressed?
  - Floating…
  - So what is in bit 0 of PINB?

- When button is pressed PB0 is connected to GND - 0V.



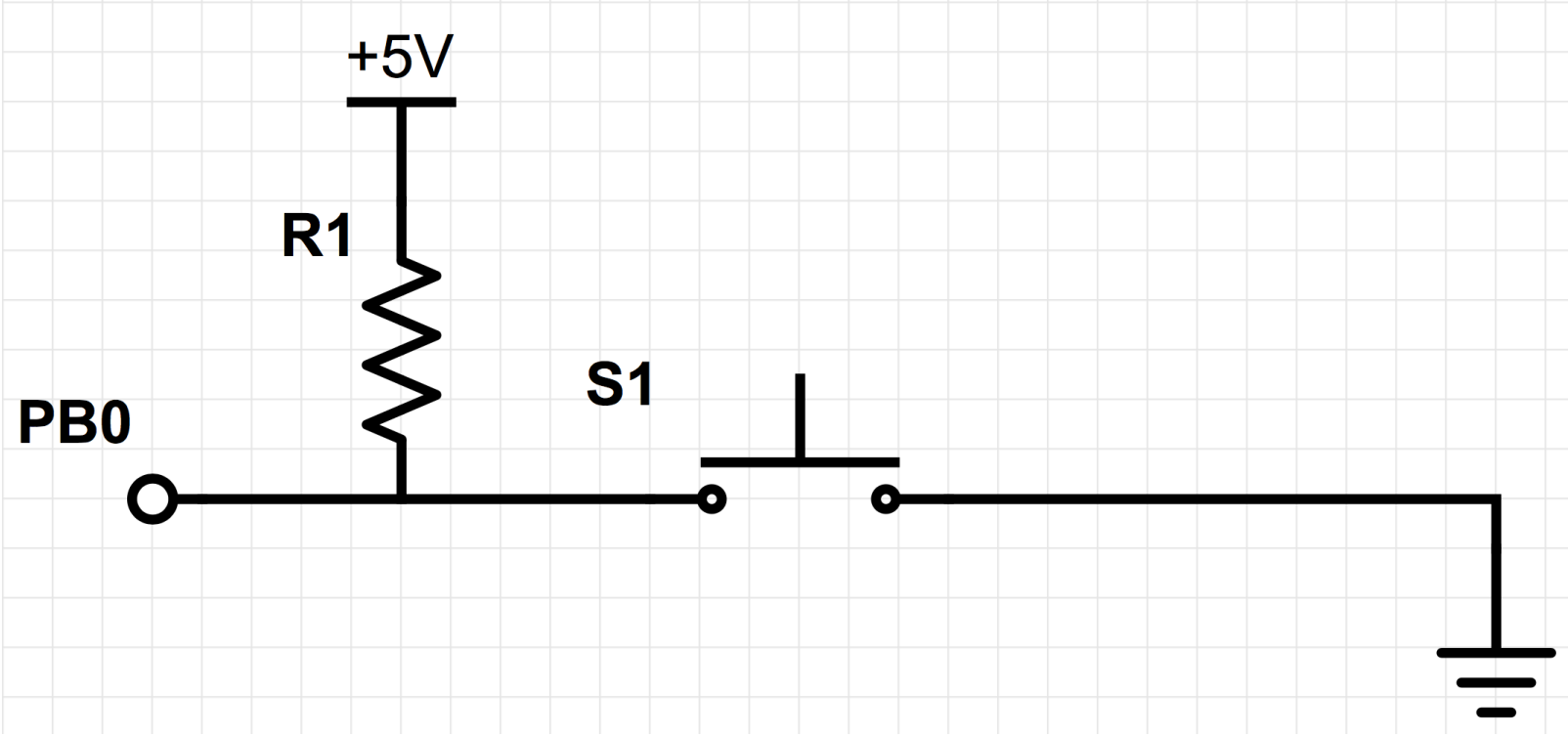
We need pull-ups…

- Pull-up resistors stop unconnected pins “floating”
- Floating logic levels are bad news in a digital systems!
- Value read in corresponding bit in PIN register would flicker…
- On AVR you can switch on and off the pull-up resistor on an input pin using the corresponding PORT bit.
- What PORT register does depends on direction of pin(DDR reg setting)

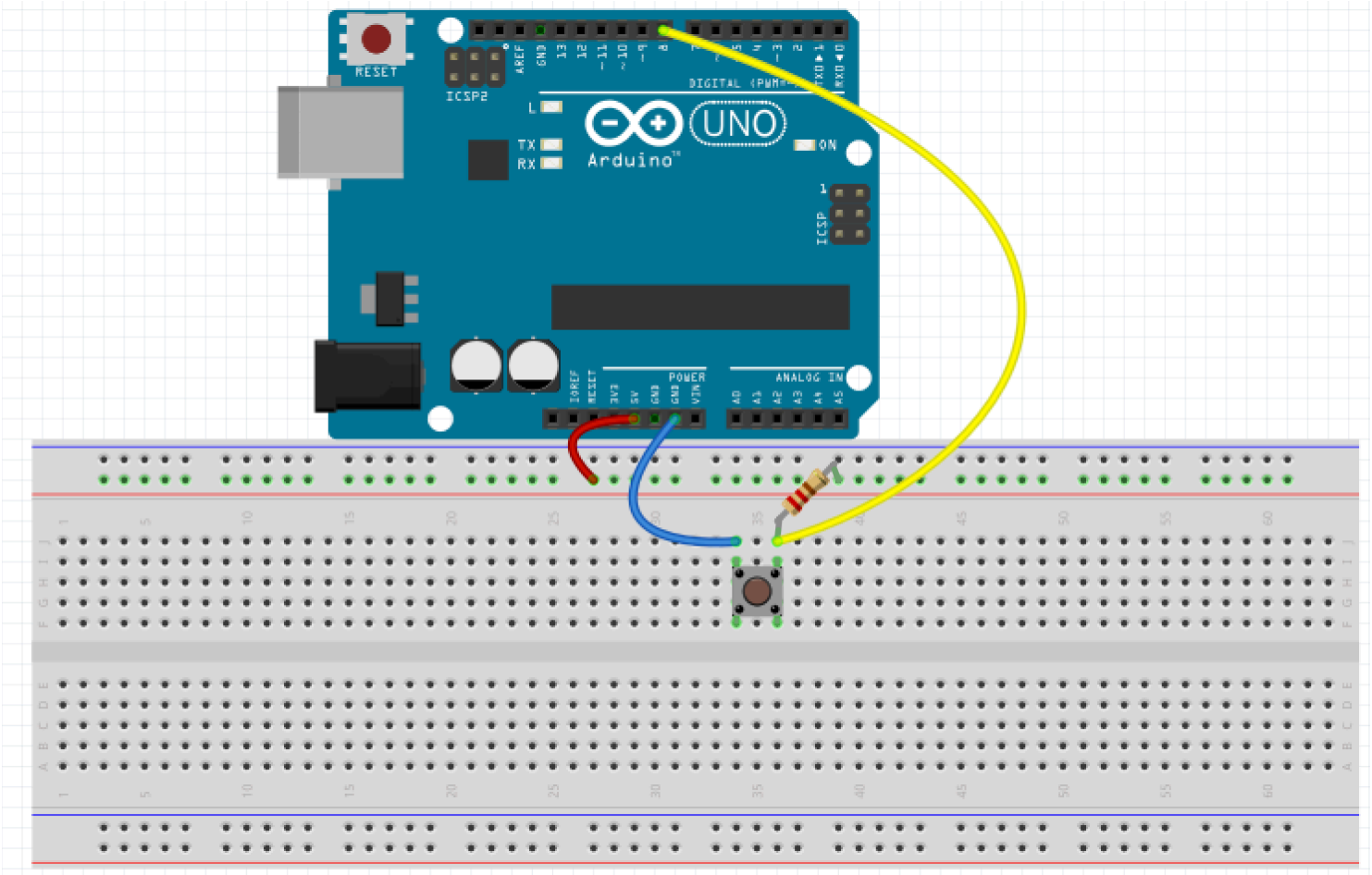


Now what voltage is at PB0?

- 5V when switch is open (un-pressed)
- 0V when switch is closed (pressed)



Can do it manually



Built-in to I/O controller hardware

- So common that it is built into the I/O controller part of the chip in the silicon.
- From the SW we can switch on and off a pull-up resistor on each pin.
- 2 steps
  1. Define pin as input pin:
     DDRB &= ~(1<<0); //PB0 as input pin

  2. Enable pull-up resistor using PORT register
     PORTB |= (1<<0);
- Function of PORTB register bits depends on setting in DDRB
  - Used either to drive the logic level on the pin (output)
  - Or to enable pull-up (input)

Try this…

```
DDRB |= (1<<5);  //PB5 as output pin – LED

DDRB &= ~(1<<0); //PB0 as input pin – button
PORTB |= (1<<0); //Switch on pullup resistor
for PB0

if(PINB & (1<<0))
{
   //bit 0 is a one
   //switch on LED
   PORTB &= ~(1<<5);
}
else //bit 0 is a zero
{
   //bit 0 is a 0
   //switch off LED
   PORTB |= (1<<5);
}
```

- Is LED on or off with button not pressed?
- What happens when I press the button?
- Inverse of what I want…
- Change circuit?

Better to change code

```
DDRB |= (1<<5);  //PB5 as output pin – LED

DDRB &= ~(1<<0); //PB0 as input pin – button
PORTB |= (1<<0); //Switch on pull-up resistor for PB0

if(PINB & (1<<0))
{
   //bit 0 is a one so button is un-pressed!
   //switch off LED
   PORTB &= ~(1<<5);
}
else //bit 0 is a zero so button is pressed!
{
   //bit 0 is a 0
   //switch on LED
   PORTB |= (1<<5);
}
```