Embedded Software

CS 145/145L

Caio Batista de Melo

Project 1

Links should be available and rubric visible:

- https://canvas.eee.uci.edu/courses/45047/assignments/929268
- https://canvas.eee.uci.edu/courses/45047/assignments/929269

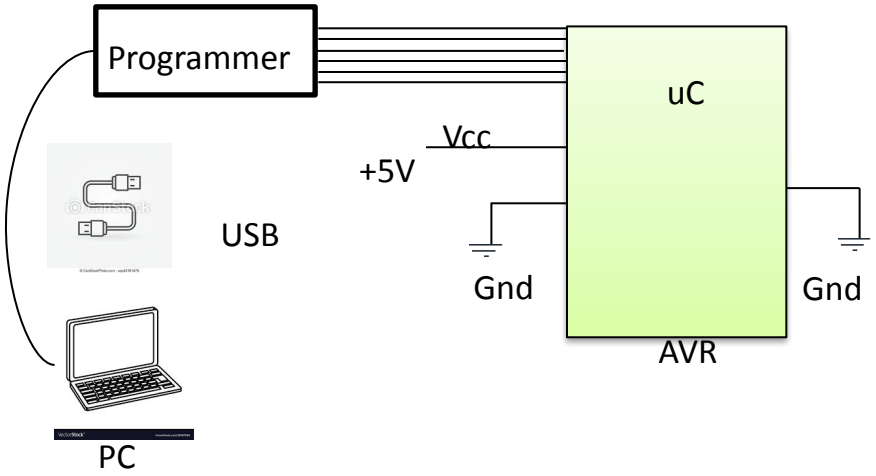
Office Hours

Tuesday and Thursday: 10-11am @ ICS 415
Starting today :)

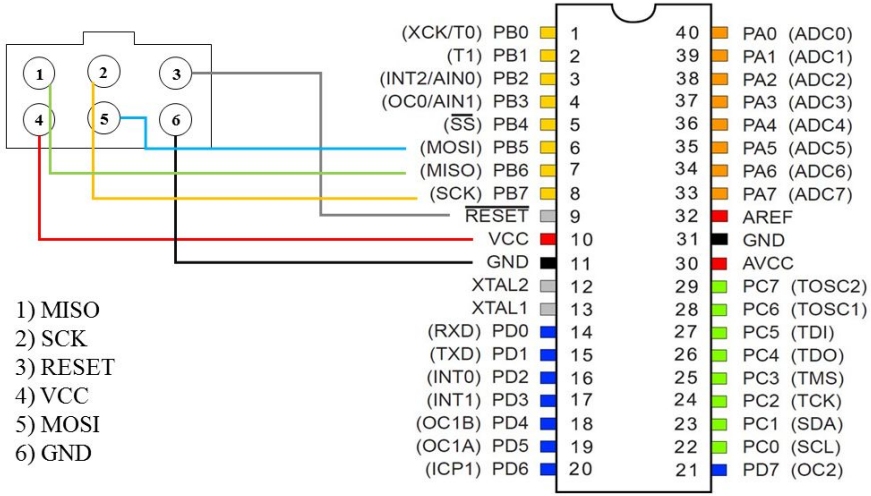
Recap

- Design an embedded computer centered around the ATMega32 microcontroller. For input, use a push button. For output, use an LED. Write a C program that blinks the LED on/off for as long as the push button is pressed. Initially, use **instruction timing** to control the LED on/off rate (for this step, use the **internal 1MHz clock**). Then, revise your timing based on one of the ATMega32 internal timers (for this step, use the external 8MHz crystal). The blinking rate should be 500ms on and 500ms off.
- Template resources on Canvas
- Make sure you work it at 1MHz completely before jumping to 8MHz.
- After downloading Microchip make the USB connection to your Atmega Processor for the prerequisite tests so that you can check if you can communicate properly.
- You need to select the ATMEGA32 or ATMEGA32A as the processor in the dropdown.

Connecting to ATmega32 Microcontroller



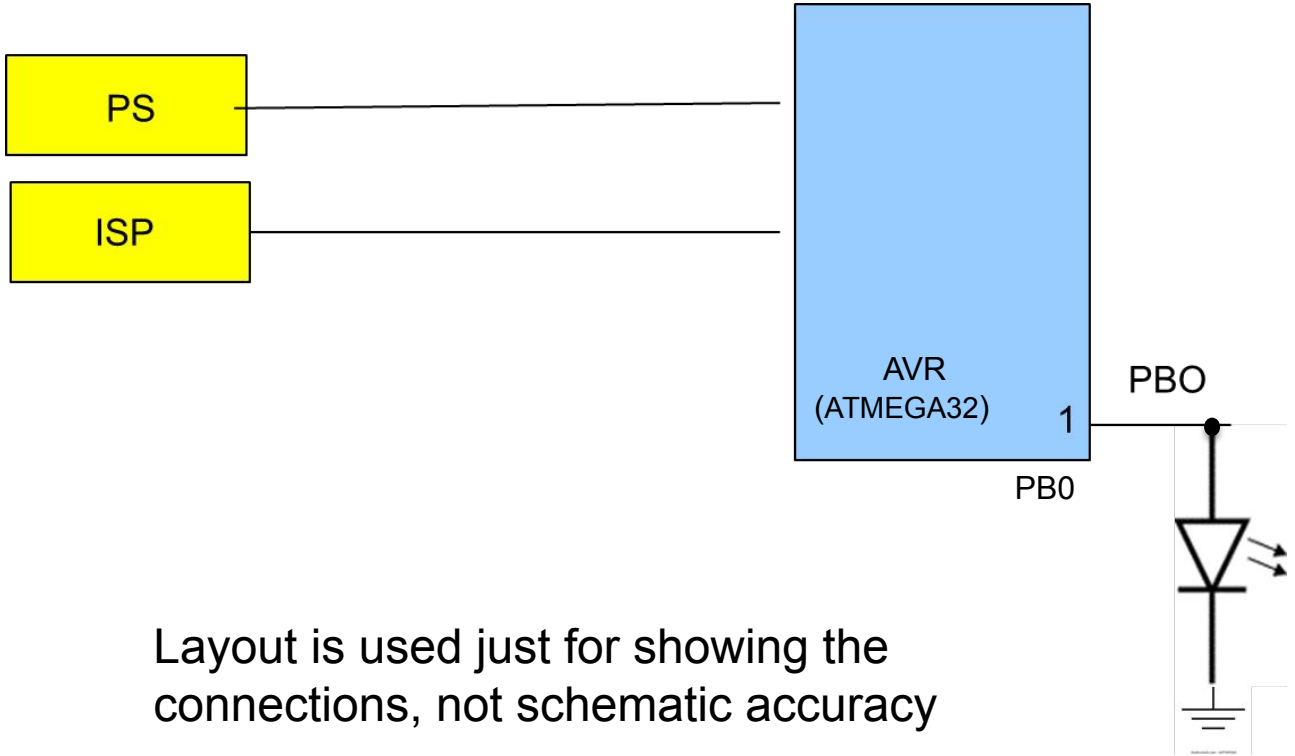
In-System Programming Interface (ISP)



1) MISO
2) SCK
3) RESET
4) VCC
5) MOSI
6) GND

https://caiobatista.com/uploads/courses/uci/s22/cs145/connector.png

LED Connection Layout



Layout is used just for showing the connections, not schematic accuracy

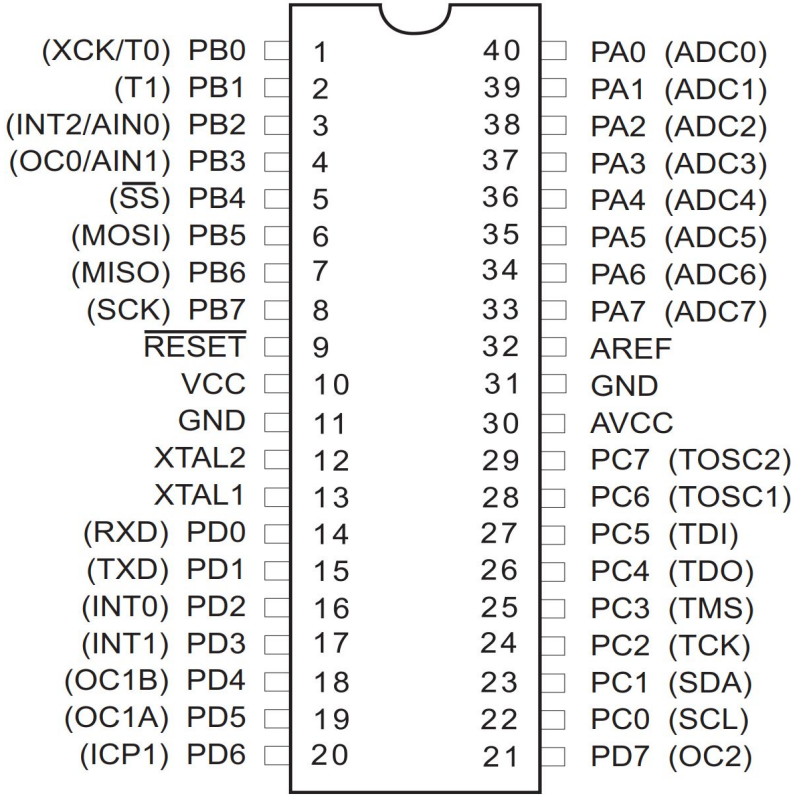
Ports

- General Purpose Input Output (GPIO) interface
  - Send/receive 0 or 1 to/from any kind of device
  - **Control** and **communicate** with the external/physical world

- 4 Ports (A,B,C,D) -> 8 bits long
  - AVR is an 8-bit processor
    - Computations on 8 bits as the basic units of operation (Internal 8-bit ALU)
    - Can you do 32-bit or 64-bit operations?

- In total 32 I/O pins, equivalent in function
  - We can control 32 things at the same time, for example (some of them might be busy)
  - You can use any port (PA, … PD)
  - You can mix the ports and control them at the bit level via software (e.g., when you need 10 bits)

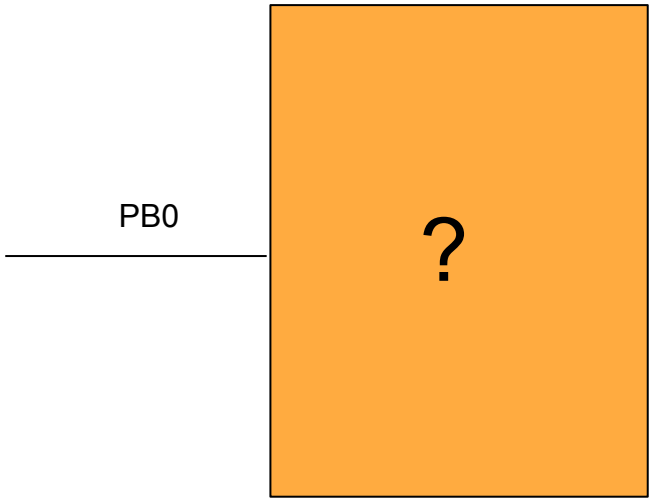
ATmega Ports



GPIO States

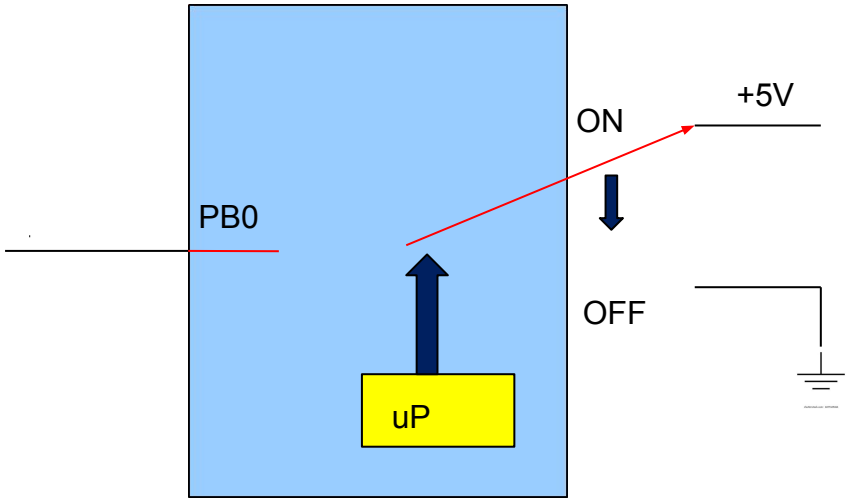
GPIO	Software Levels	Hardware Electric Levels
ON	1	+5V
OFF	0	0V (GND)

GPIO Blackbox



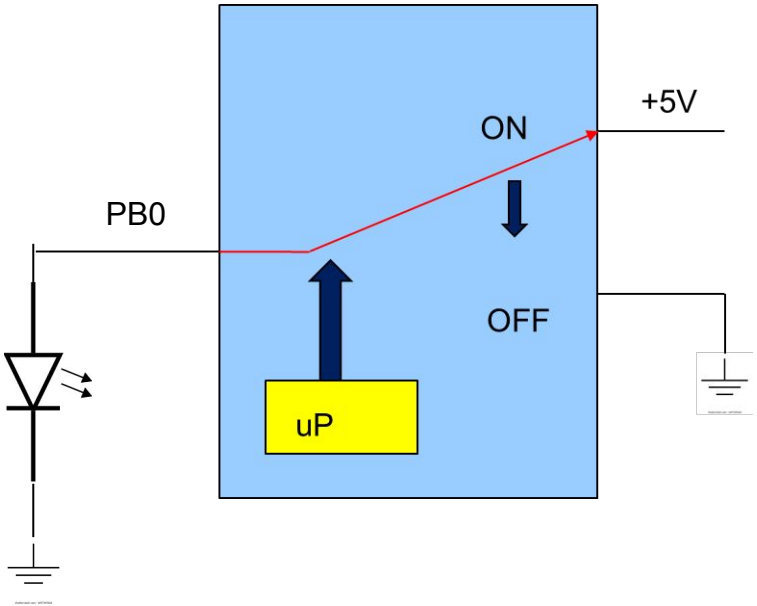
What is the internal working?

GPIO Blackbox and Microprocessor



uP decides the connection of input to output through software

GPIO Blackbox



Final connection with the LED, creating a circuit

Software

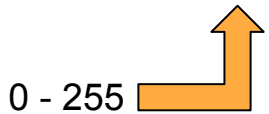
Special Function Register (SFR)

- Allows to control I/Os
- Functions of initializing, reading, writing from memory.
- In addition to being like a variable like operator it mainly has **side-effects**.
  - The side-effect is desirable
  - Each GPIO has 3 SFRs (side-effects).

```
int x;
x = 3;          Write
print(x);       Read
```

SFR Side Effects

- Syntax to call output, input and direction
- Case sensitive
- Port names can be substituted
- These are our programming interfaces
- They are defined as unsigned char in their respective header files.
- E.g.- **unsigned char PORTB**;



- Included in the header files (avr.h)

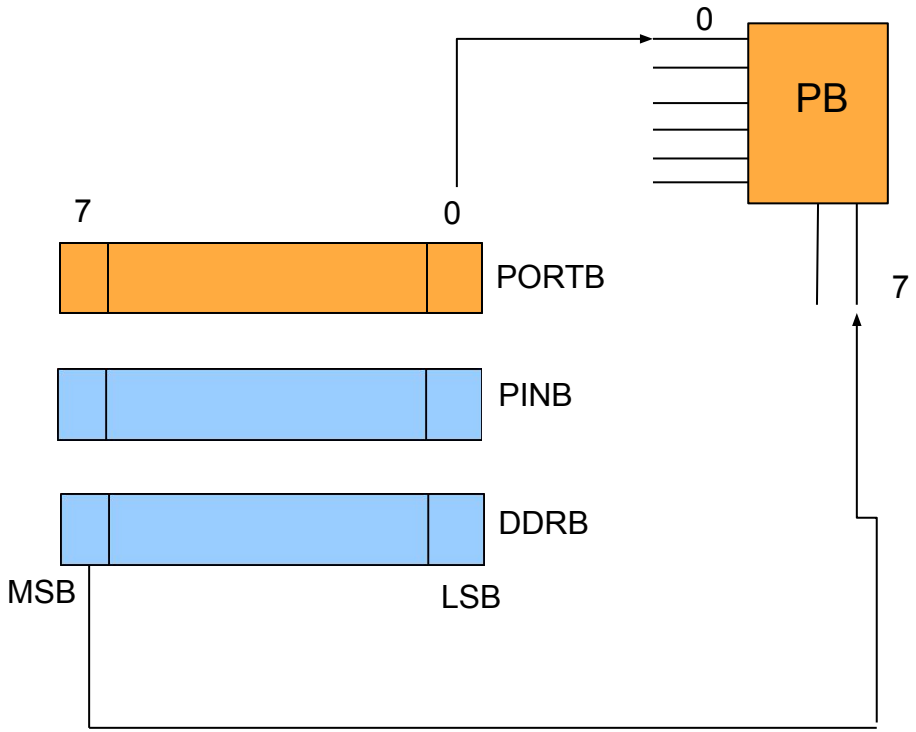
	PORT (e.g., B)	PIN (B)	DDR (B)
Side Effects	Output	Input	Direction

Data Types

C DATA TYPES (in AVR)	AVR(SIZES)
char	8 bit
short	10 bits
int	16 bits
long	32 bits
long long	32 bits

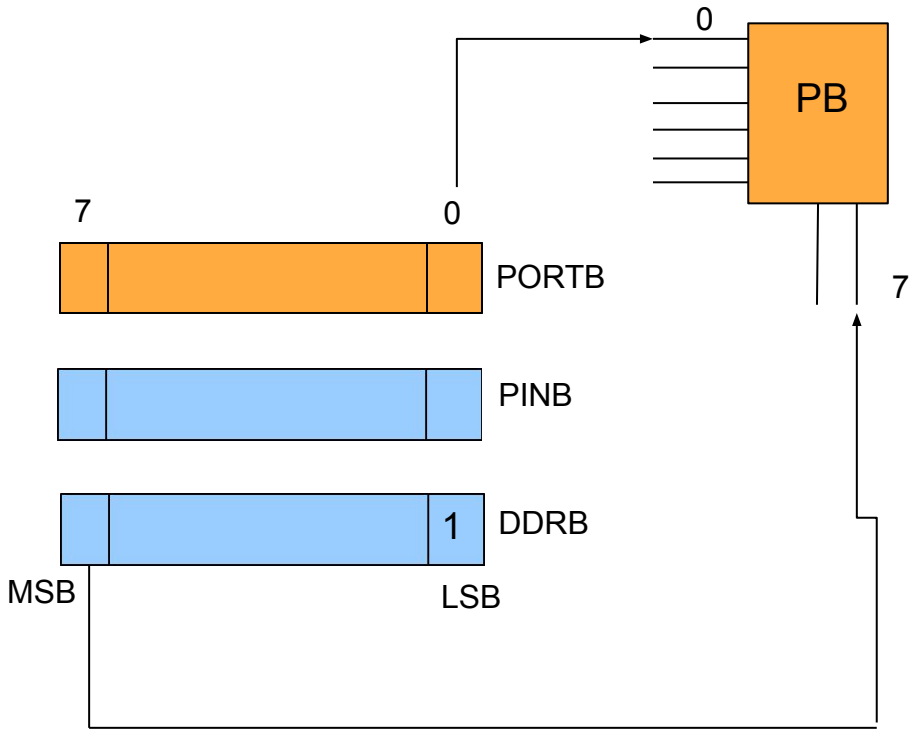
Software to Physical Pin Layout

- Each SFR has a data register through which you can control the physical pins as shown in the figure
- You can read and write to these like any variables



Data Direction Register (DDR) Functionality

- By changing the LSB in the DDR register we can make the I/O pin an output (1) or input (0)
- Here in the given figure the pin 0 of PORT B is initialized as output



Program Example

```
#include "avr.h"

main(){

DDRB = 1;   00000001
PORTB = 1

while(1){
}
```

// Include proper header file

// Initialize PIN 0 of PORT B as Output
// Initialize PIN 0 of PORT B as HIGH / ON

// Infinitely running loop

LED Blinking

```
#include "avr.h"

main(){

DDRB = 1;   00000001
PORTB = 1

while(1){

    PORTB = 1;
    wait();
    PORTB = 0;
    wait();
}
}
```

```
void wait(void){
int i;

for(i=0; i < 10000; i++);
}
```

Instruction timing!

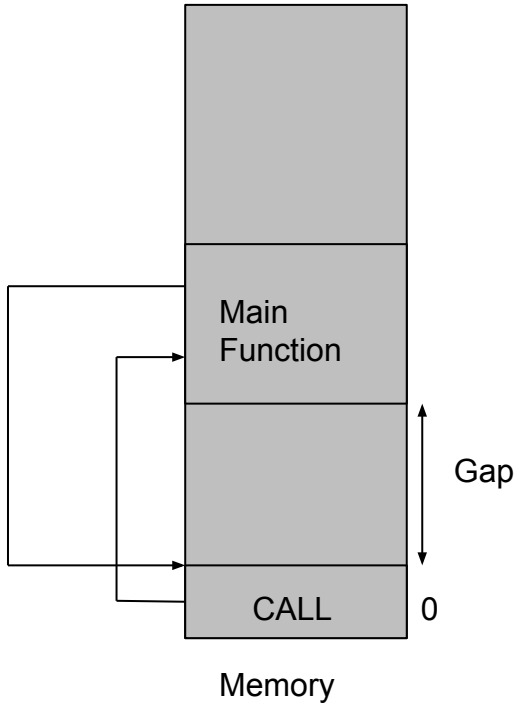
Volatile Variable

- Sometimes the compiler may remove the delay loop when optimizing the code as the delay loop isn't doing anything but waiting
- Trick is to use the keyword volatile while initializing the variable to tell that its needed in our program

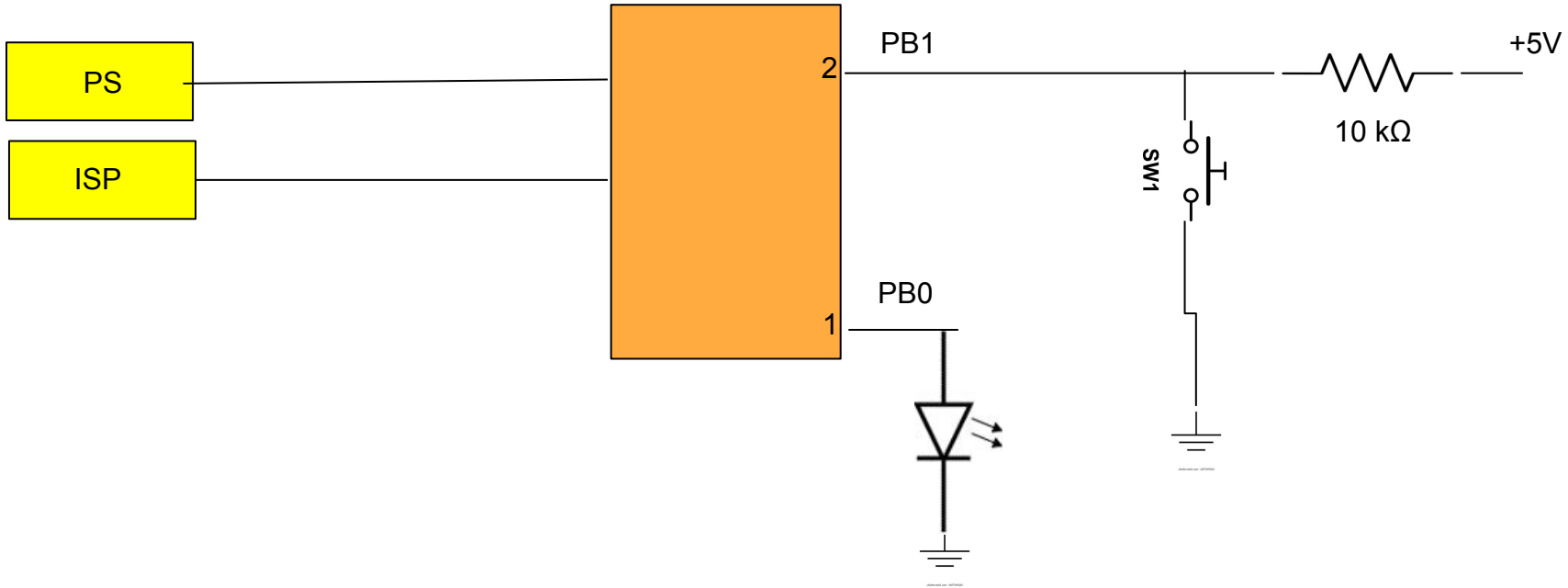
```
void wait(void){
volatile int i;

for(i=0; i < 10000; i++);
}
```

Program in Memory

- The program starts from the 0 location of a memory where there is a program call
- The call jumps the program counter to the main program which then returns to the position after the program call
- There is a gap between the program call and the main program



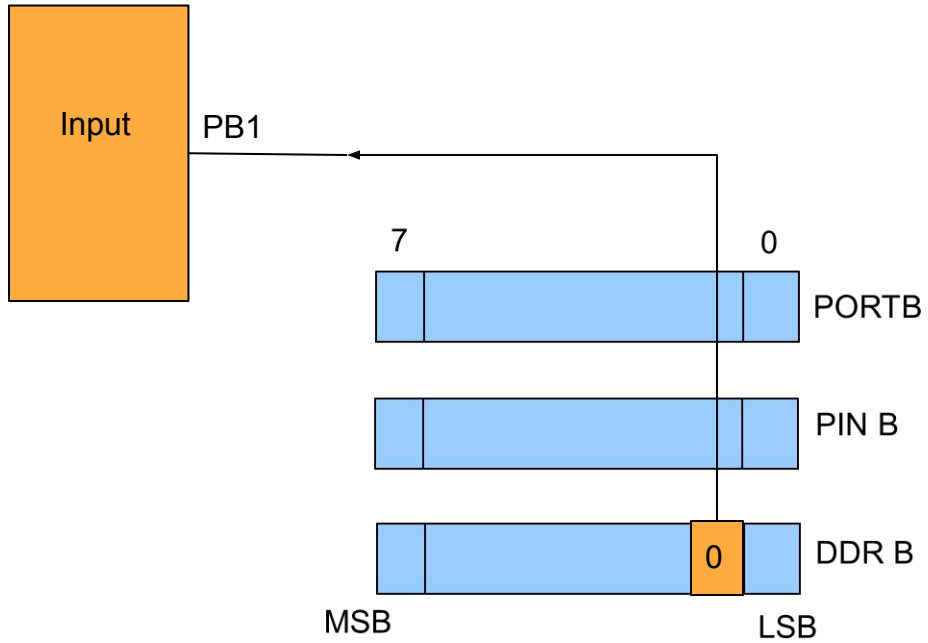
Final Layout of the Circuit



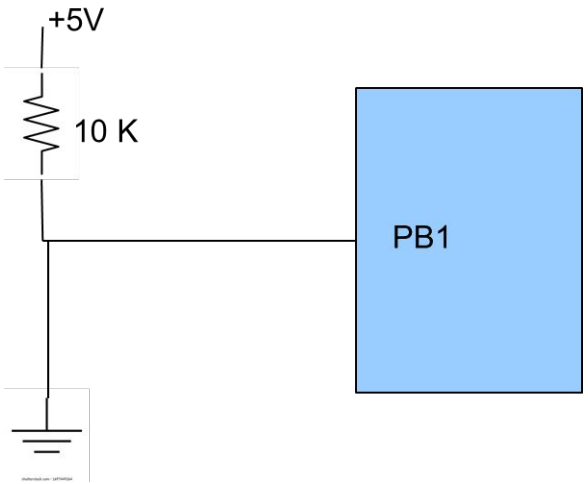
Input as a Push Button

- As we connected the LED to PIN 0 let's connect the push button to PIN 1
- As it should be initialized as input, we make the PIN 1 of DDRB equal to 0
- For input initialization the PORT register is useless and the value of the input should be initialized in the PINB register

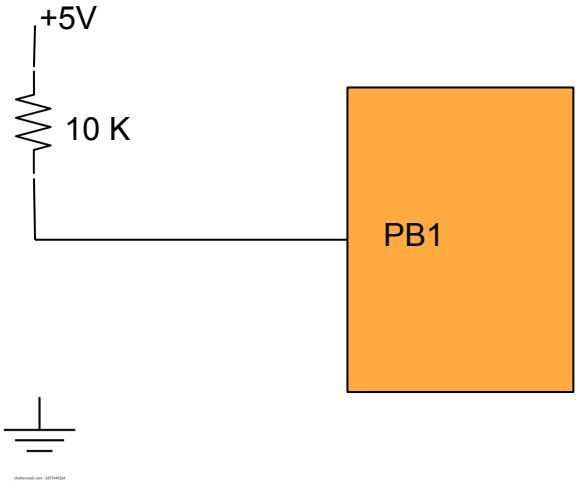
Example: if (PINB == 3) …



Button Layout



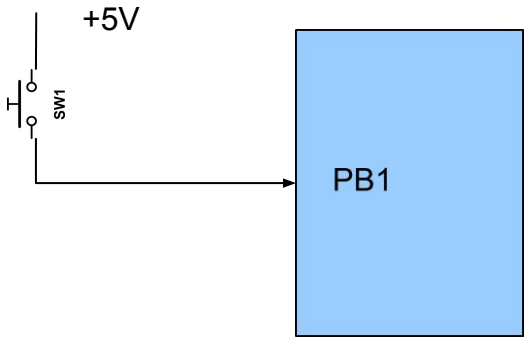
Switch Pressed



Switch Left idle

Undefined State

Leaving Switch idle thus has undefined value



Processing on Pins

In order to check for selective bits, make the rest zero and do bitwise AND

```
main(){
.
.
.
/* Set the DDRB properly
for LED and Push Button!*/
  for (;;){
    …
    //  LED ON
    …
    //  LED OFF
   }
```

Bit twiddling

```
      00000010
PINB  ??????x?
--------------------
      000000x0
```

```
for (;;){
    …
    if (PINB & 2 ) {
        …
    }
}
```

See you next time :)

Q & A