ECE375 Lab 2
C -> Assembler -> Machine Code

TA:

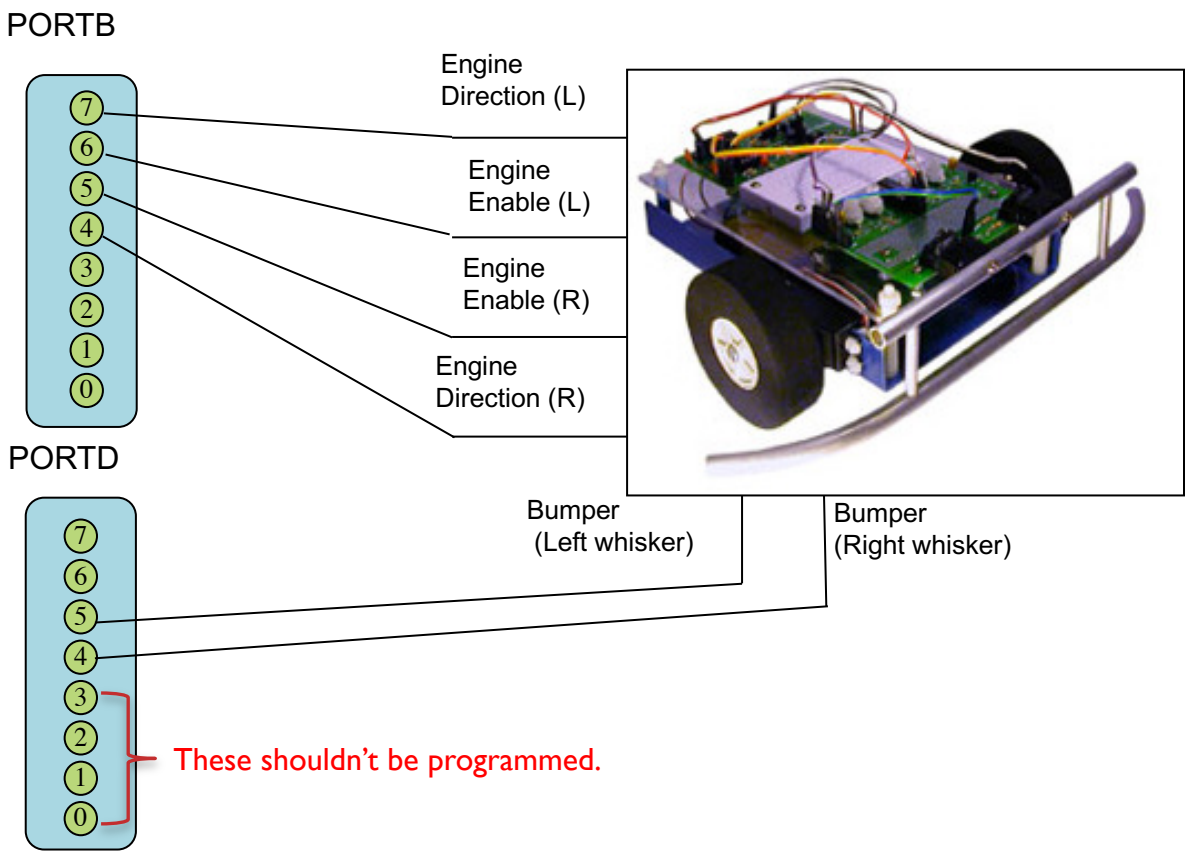
School of Electrical Engineering and Computer Science

Oregon State University

Goal of this Lab

- Understand BumpBot behaviors through LEDs.
- Learn how to control with registers.
- Learn how to avoid switch debouncing.
- Understand the difference between C and Assmebly.

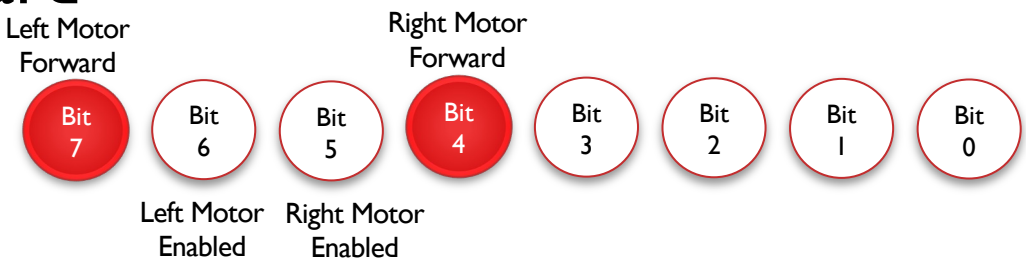
Connection Guides



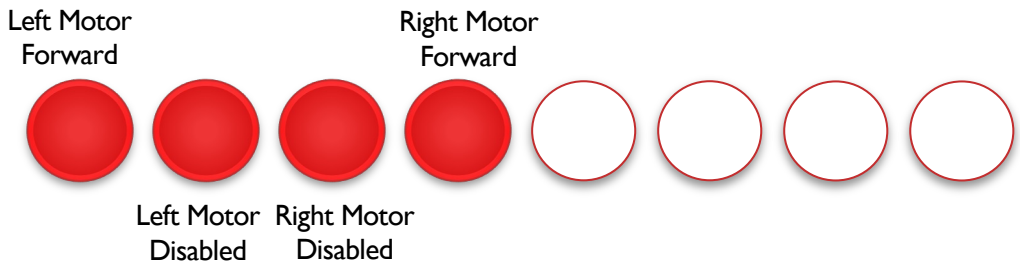
Bumpbot Behaviors

On(1) Off(0)

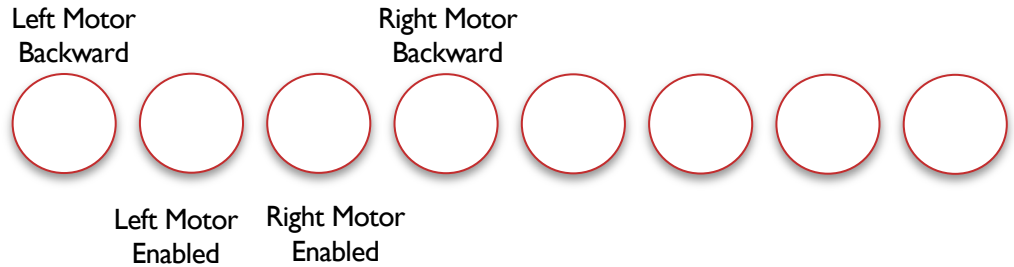
- Forward



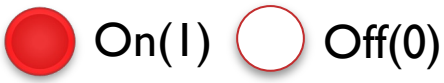
- Halt



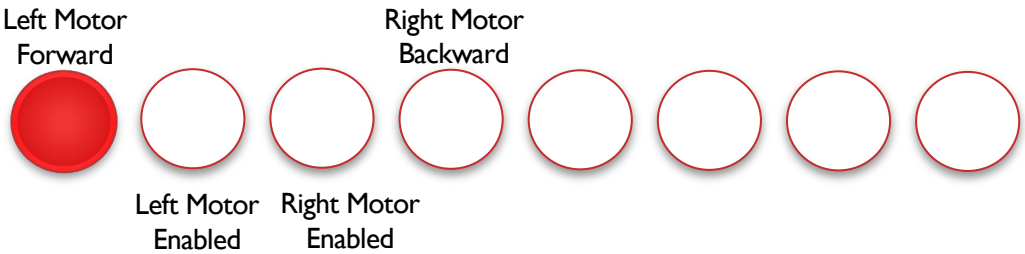
- Backward



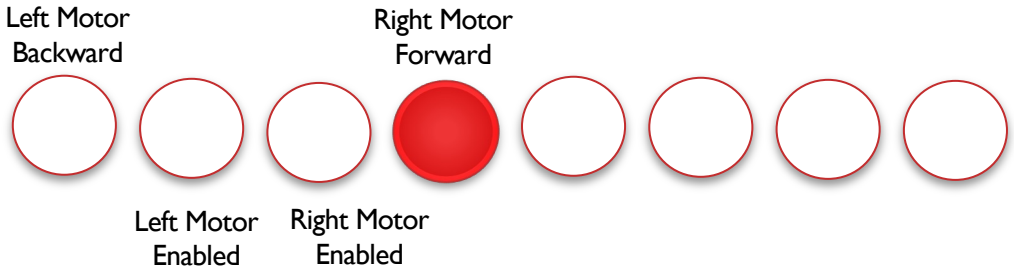
Bumpbot Behaviors



- Turn Right



- Turn Left



Controlling Registers

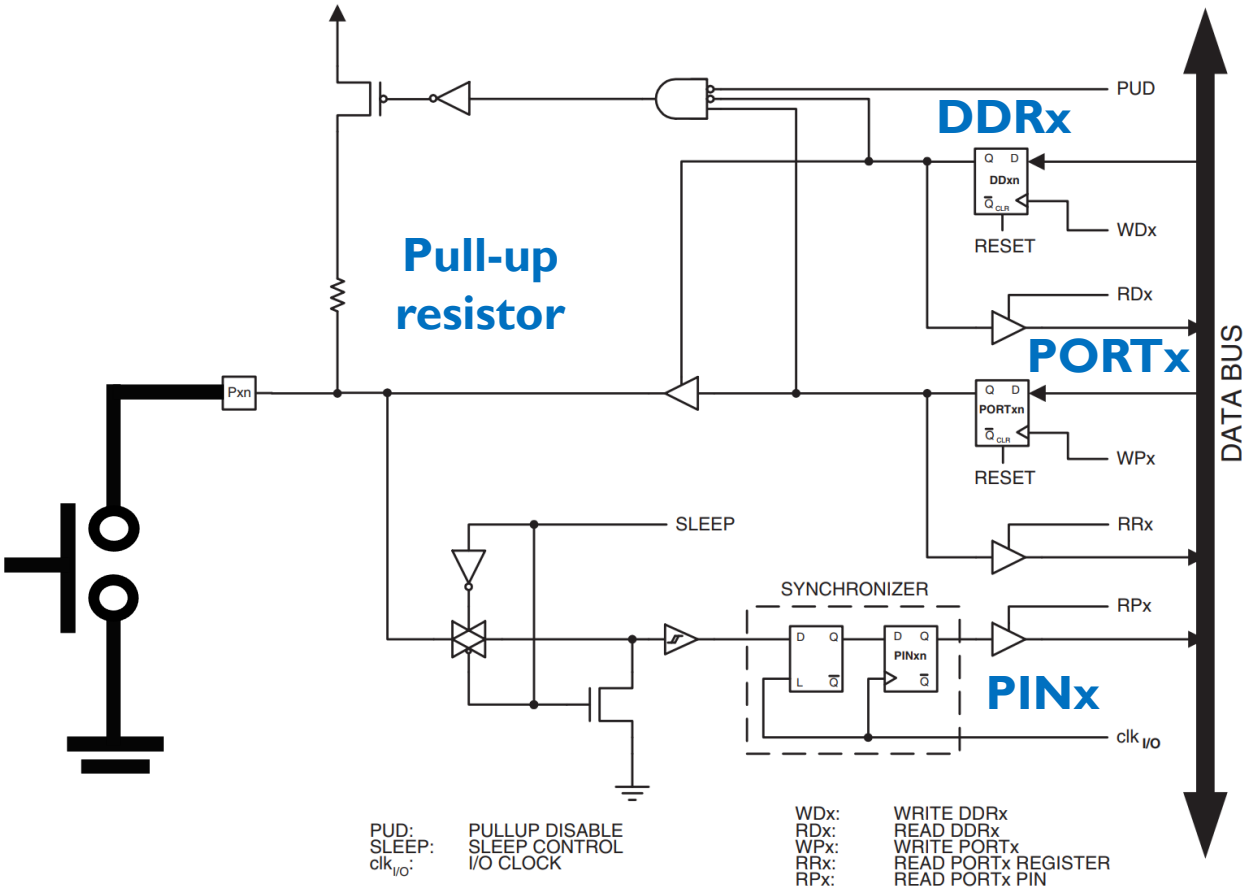
- Register Types
  - DDRx is a Data Direction Register for port x
  - PORTx is a Port Output Register for port x
  - PINx is a Port Input Register for port x

- Output Port Settings
  - DDRB = 0b11111111 ; set bit 0-7 as output
  - PORTB = 0b11110000 ; assign output data to bit 4-7
    ; LEDs are turned on

- Input Port Settings
  - DDRD = 0b00000000 ; set 7-0 bits as inputs
  - PORTD = 0b11111111 ; enable pull up resistors
  - IN mpr, PIND ; Read input data to mpr

AVR Ports

Figure 30. General Digital I/O[1]

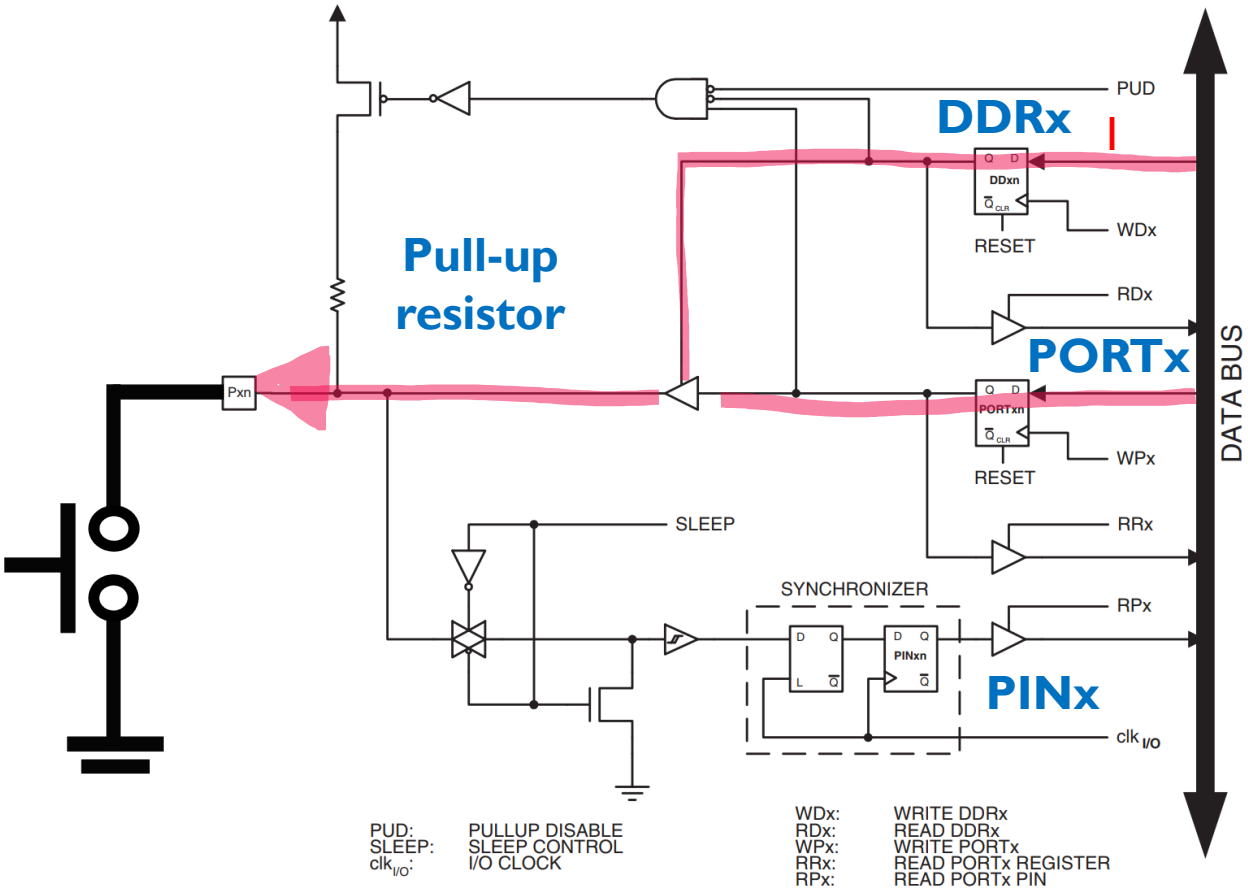


Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Atmega128 Datasheet 66p

AVR Ports – Configure output

Figure 30. General Digital I/O[1]

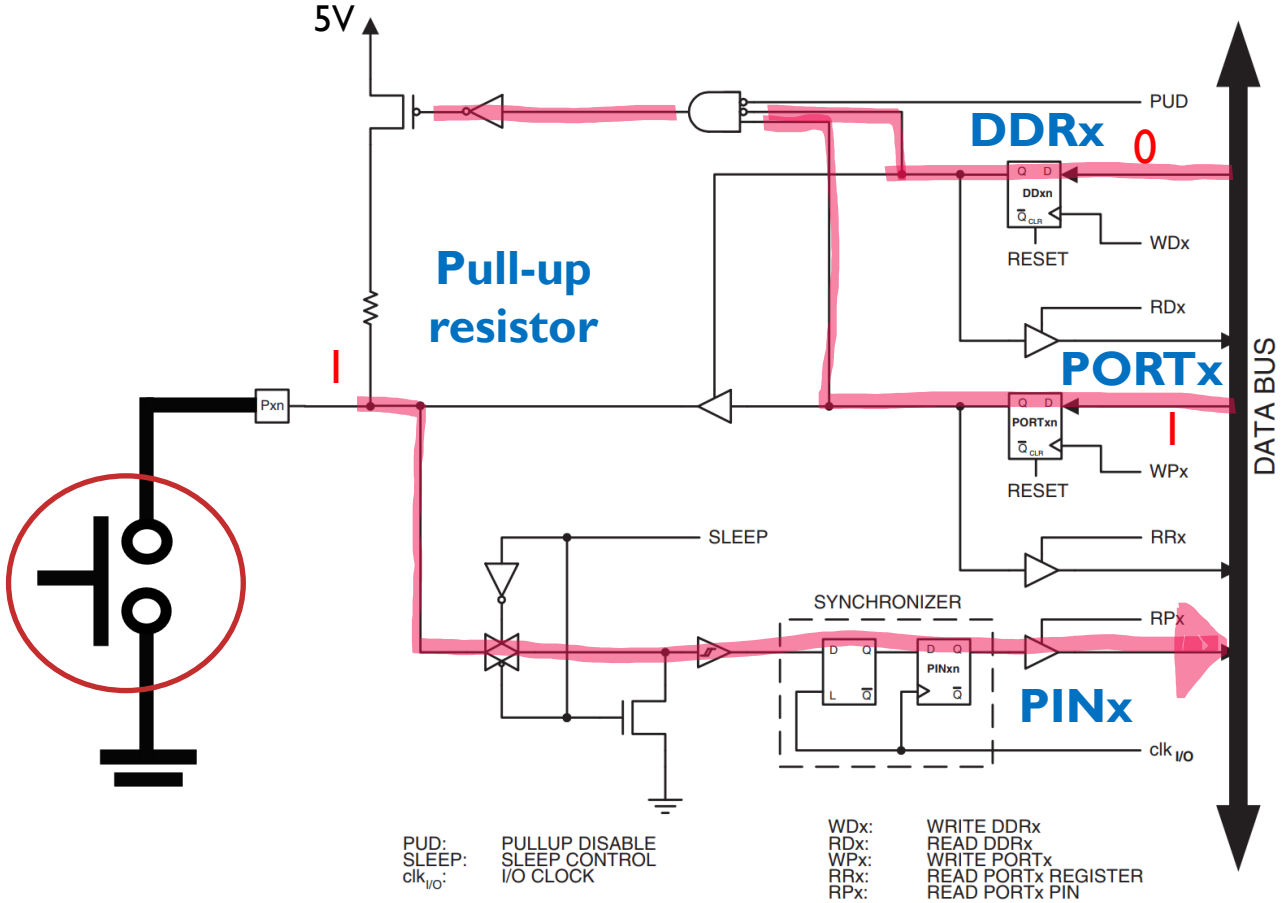


Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Atmega128 Datasheet 66p

AVR Ports – Configure input

Figure 30. General Digital I/O[1]

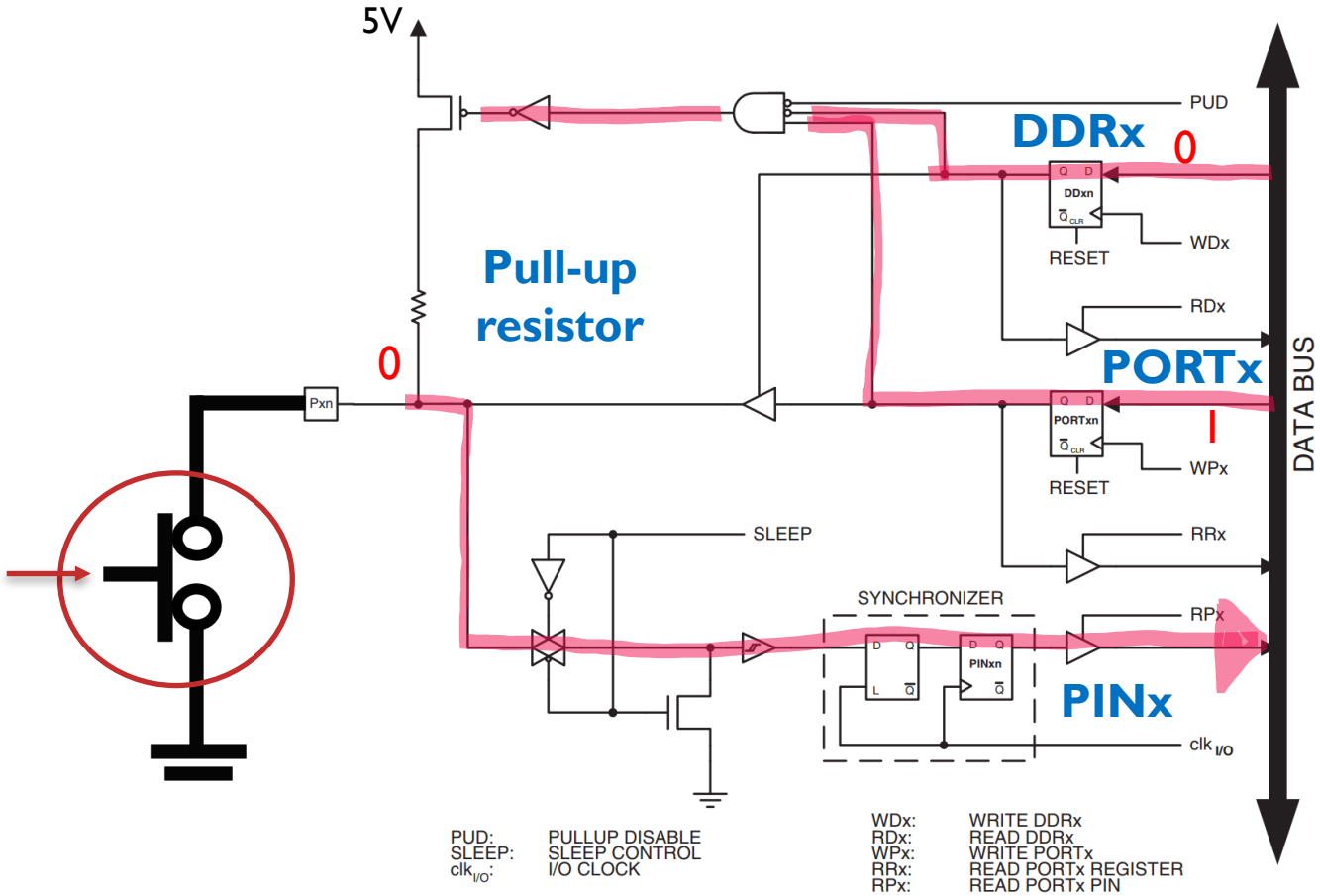


Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Atmega128 Datasheet 66p

AVR Ports – Configure input

Figure 30. General Digital I/O[1]



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Atmega128 Datasheet 66p

Switch Debouncing

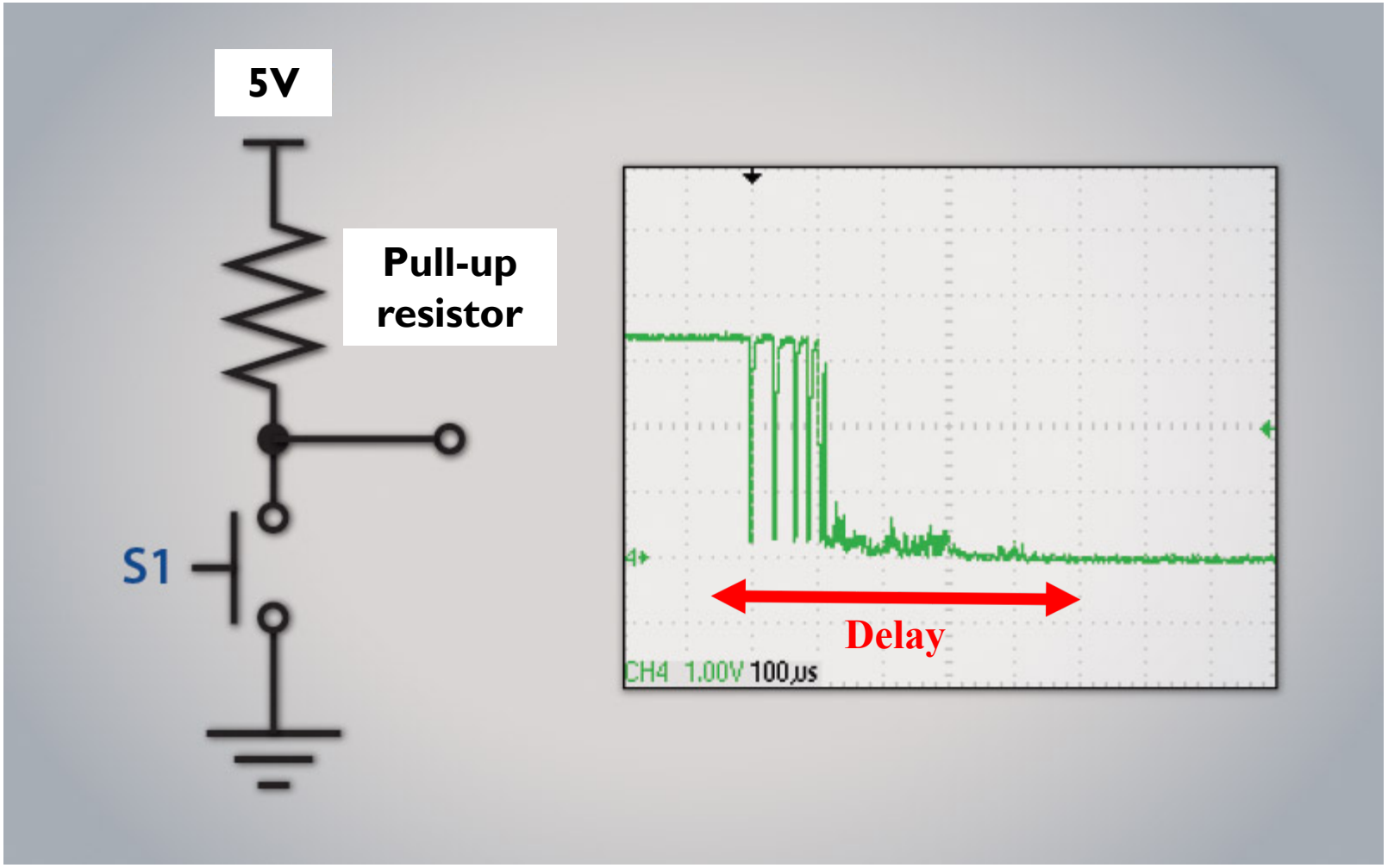


Image from google

C vs Assembly

- In C

```
DDRB = 0b11110000        // set bit 7-4 as output
PORTB = 0b01100000       // turn on LEDs connected to bit 6-5
```

- In Assembly

```
LDI     mpr, 0b11110000
OUT     DDRB, mpr        ; set bit 7-4 as output
LDI     mpr, 0b01100000
OUT     PORTB, mpr       ; turn on LEDs connected to bit 6-5
```

C vs Assembly

- In C

```
uint8_t mpr = PIND & 0b00110000;        //extract only 4,5th bit
If (mpr == 0b00100000)                  //check if the right whisker is hit
{
    BotAction();                        //call BotAction
}
```

- In Assembly

```
IN      mpr,    PIND        ;read input values
ANDI    mpr,    0b00110000  ;extract only 4,5th bit
CPI     mpr,    0b00100000  ;check if right whisker is hit
BRNE    NEXT                ;if no, go to NEXT
RCALL   BotAction           ; if yes, call BotAction
NEXT:
```

Check-off Lists

- Correct LED behaviors based on the switch buttons.
  - LEDs represent the Bot behaviors
  - Switch buttons represent whisker hits.

Compilation for Mac and Ubuntu users

1. Install avr-gcc toolchain.

2. Download Makefile from lab webpage.

3. Open the file with a text editor, set PRG variable to a file name of your source code without the file extension(.c)
   - e.g., PRG = DanceBot

Questions?