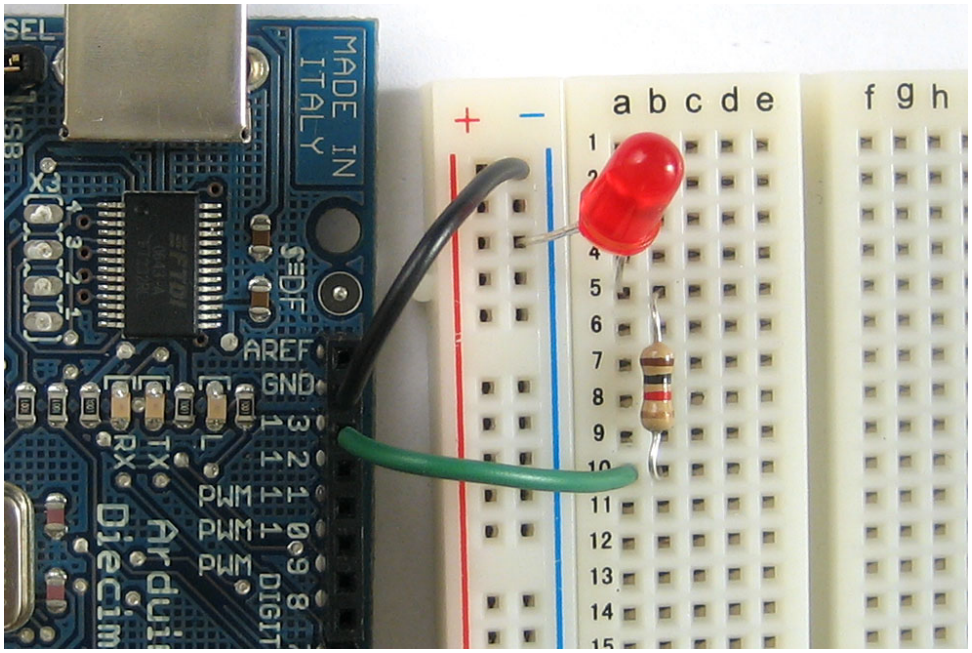
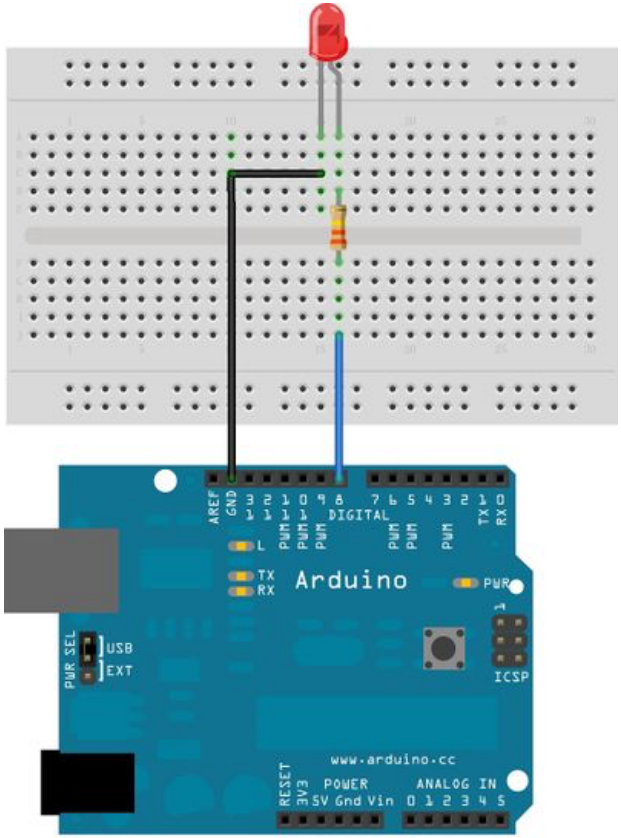
LED & Button & Delay & Interrupt

Using a Led


MADE IN ITALY
AREF
GND
a b c d e
f g h
+
−


AREF
GND
DIGITAL
Arduino
PWR
USB
EXT
ICSP
www.arduino.cc
POWER
ANALOG IN

sketch_apr14a.s

```
#include "avr/io.h”
.global loop
.global setup
setup:
        ser r16
        sts DDRB, r16
        ret
loop:
        sts PORTB, r16

ret
```

Calculating a delay of 1 s

```
#include "avr/io.h"
.text
.global setup
.global loop
.global timer
.global again


setup:
       ldi r16, 0b00100000
       sts DDRB, r16

loop:
       ldi r17, 0b00100000
       sts PORTB, r17
       call timer
       ldi r17, 0b00000000
       sts PORTB, r17
       call timer
       jmp loop
```

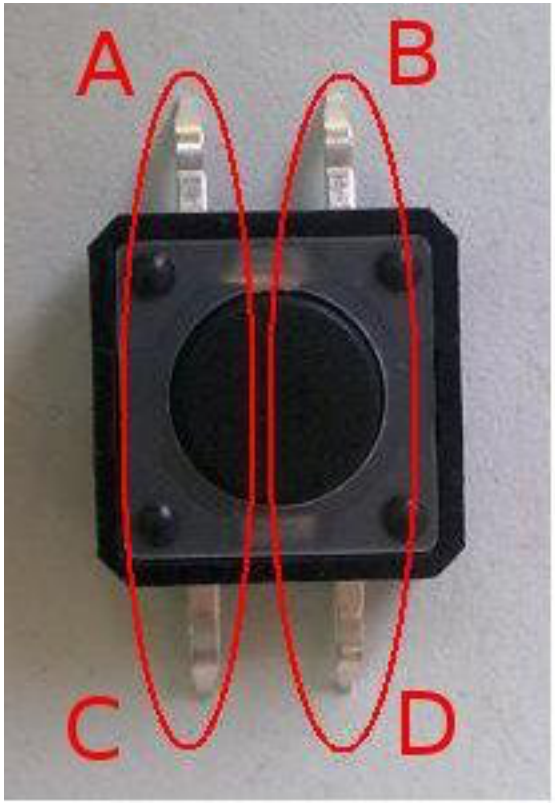
```
timer:
LDI R17, 32
L1:  LDI R18, 200
L2:  LDI R19, 250
L3:
        NOP
        NOP
        DEC R19
        BRNE L3
        DEC R18
        BRNE L2
        DEC R17
        BRNE L1

ret
```

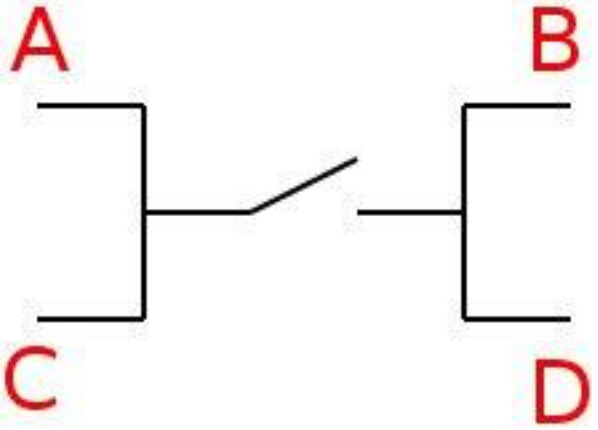
Stack

```
#define __SFR_OFFSET 0

#include "avr/io.h"
#include <avr/portpins.h>


.text
.global setup
.global loop
```

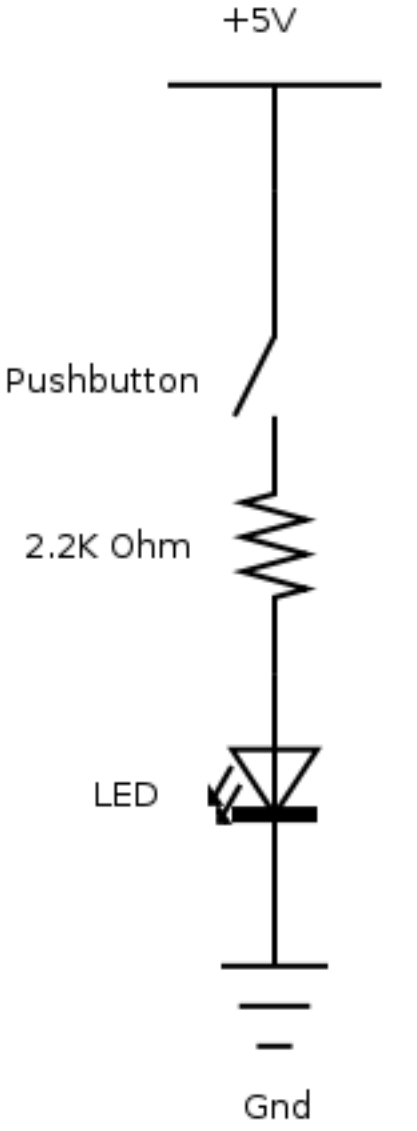
```
setup:

        ldi r22, 0x20
        sts 0x0100, r22 ; load a value to memory address 0100

         ldi r17, 0x00
         ldi r18, 0x01
         mov ZH, r17
         mov ZL, r18   ; Z point to memory addres 0100

         ldi r16, hi8(RAMEND)
         sts SPH, r16
         ldi r16, lo8(RAMEND)
         sts SPL, r16          ; create the stack


         ld r19, Z
; load the value from the memory address that Z register points

         push r19    ;push the value to the stack
         pop r19   ;pop the value from the stack

         LD r28, Z+
; Load the value from the memory address that Z register points, and increase
the pointer


loop:

       jmp loop
```

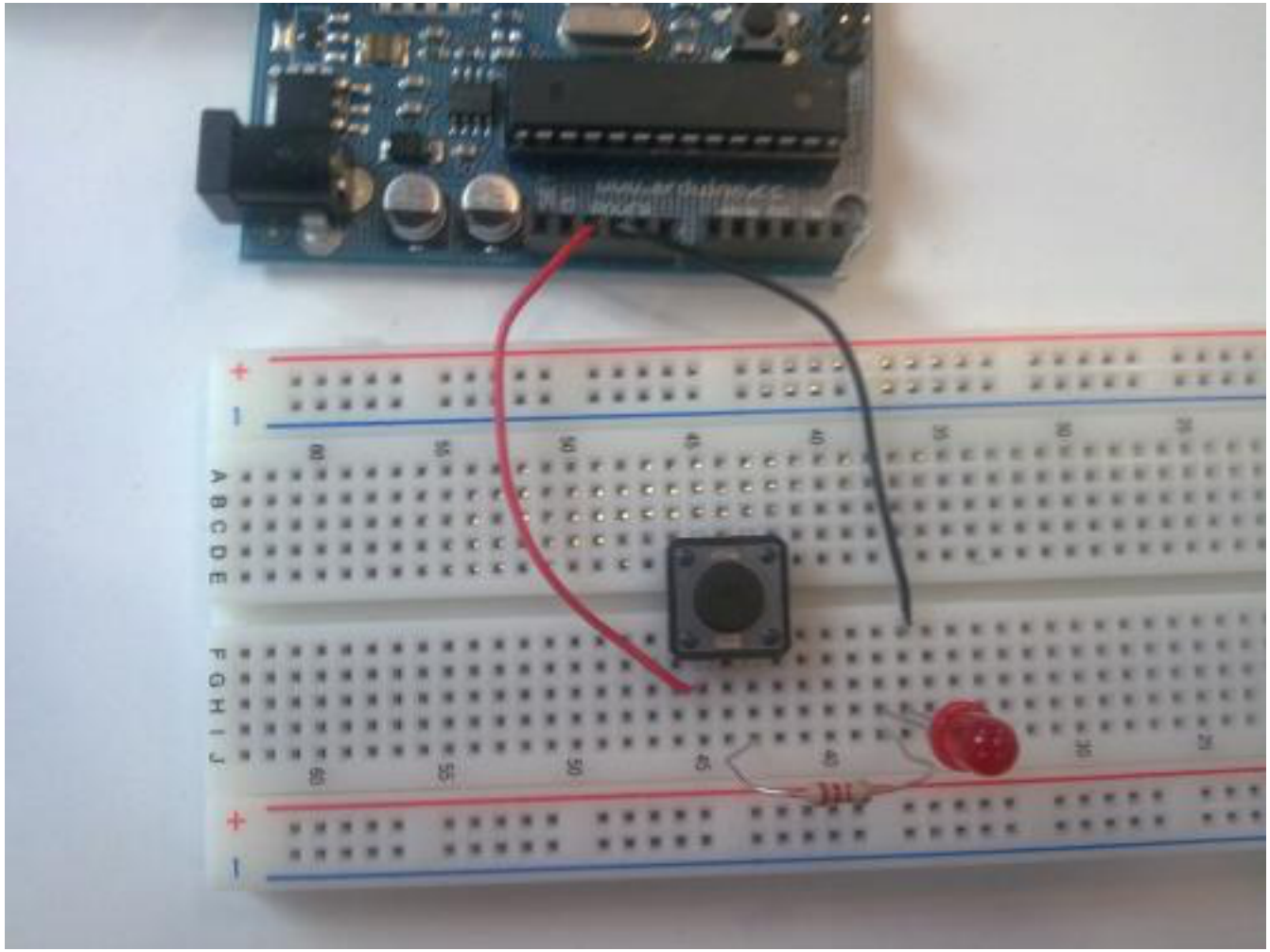
Button

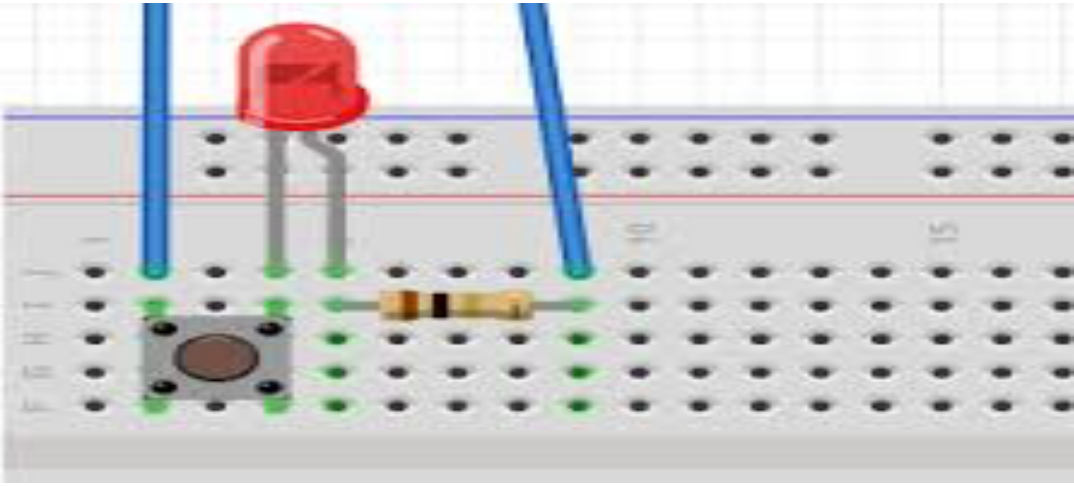


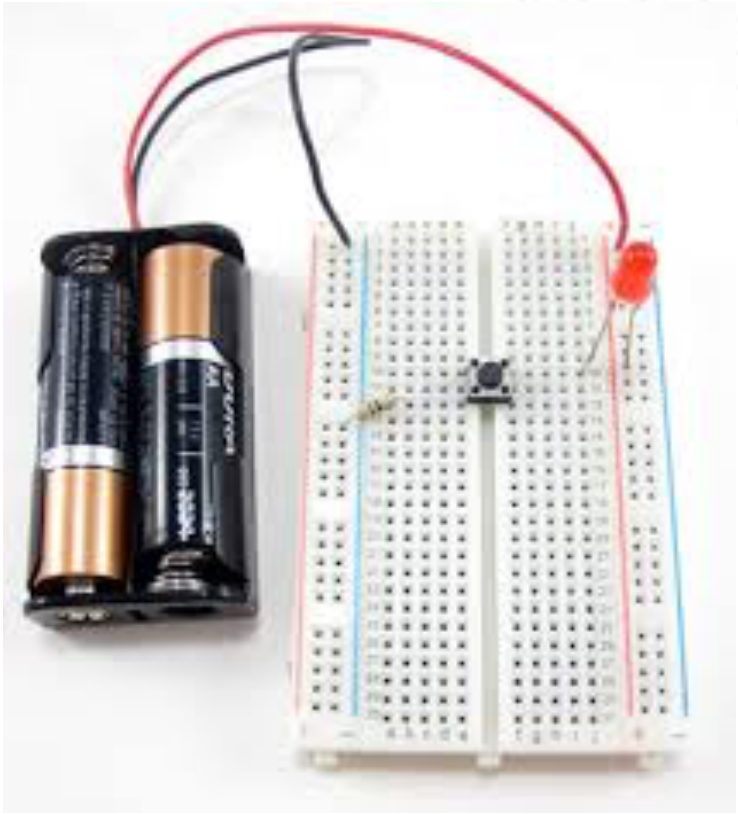


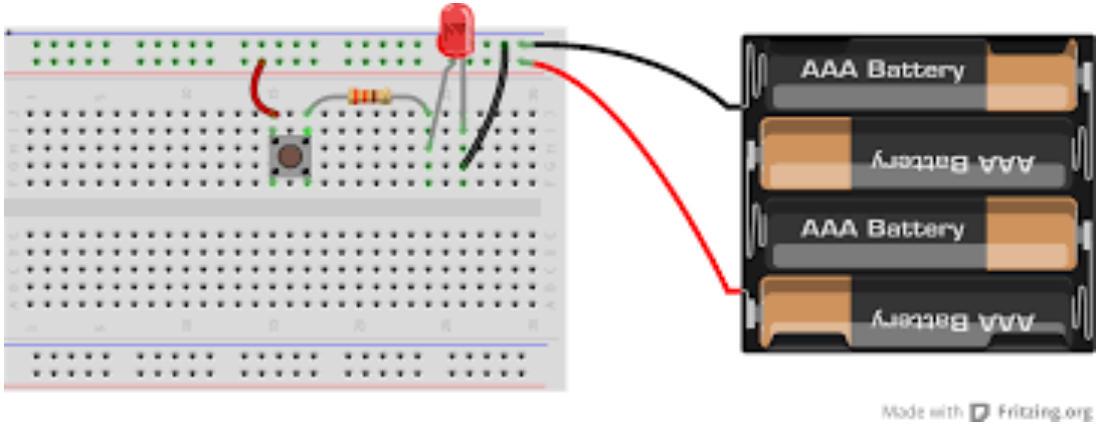
Button and LED – no asm control



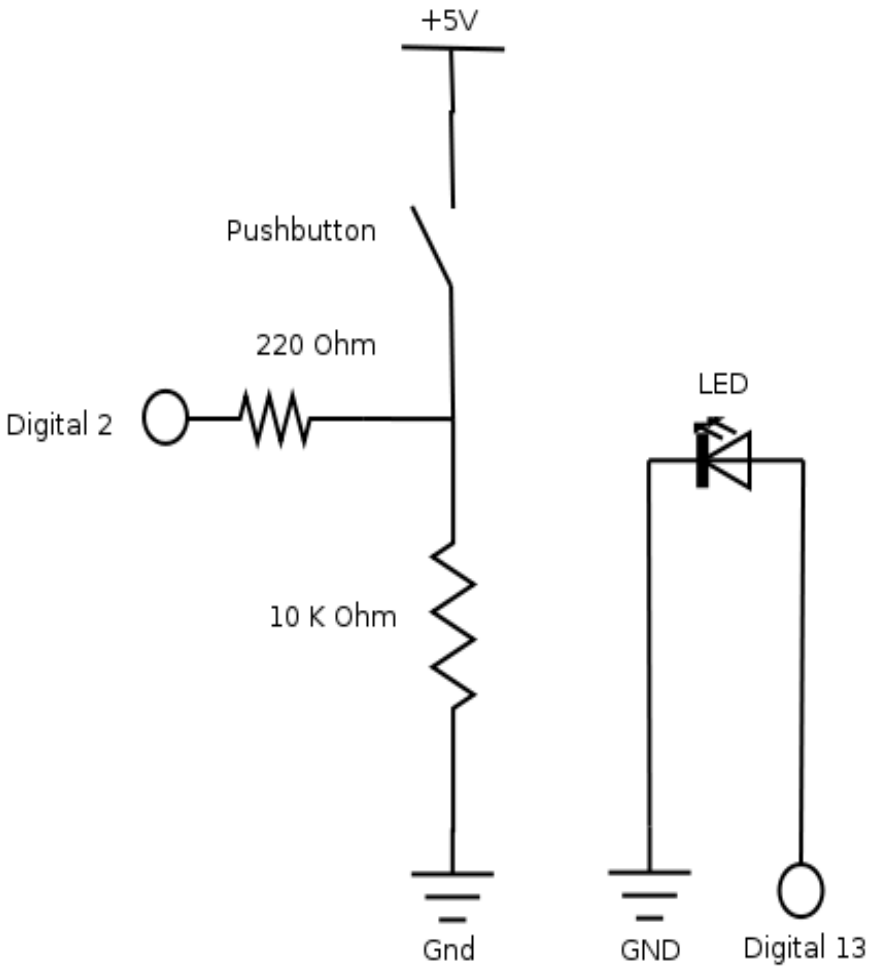


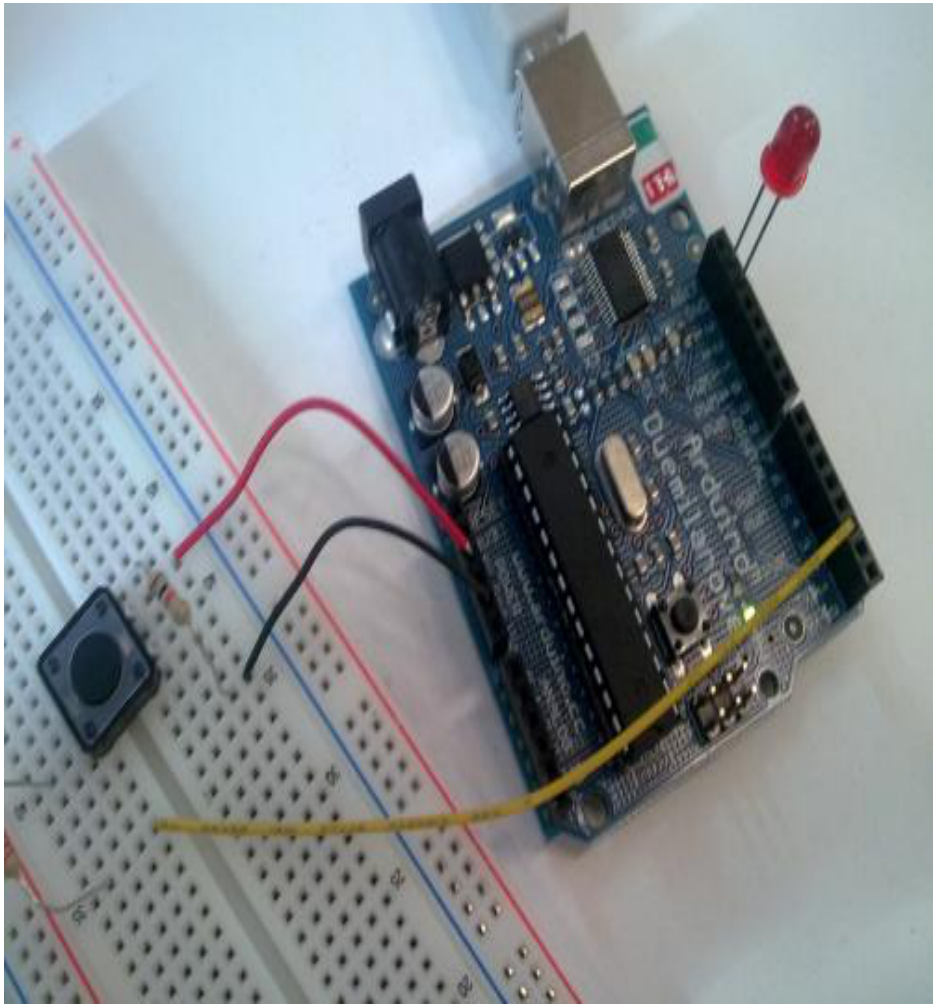




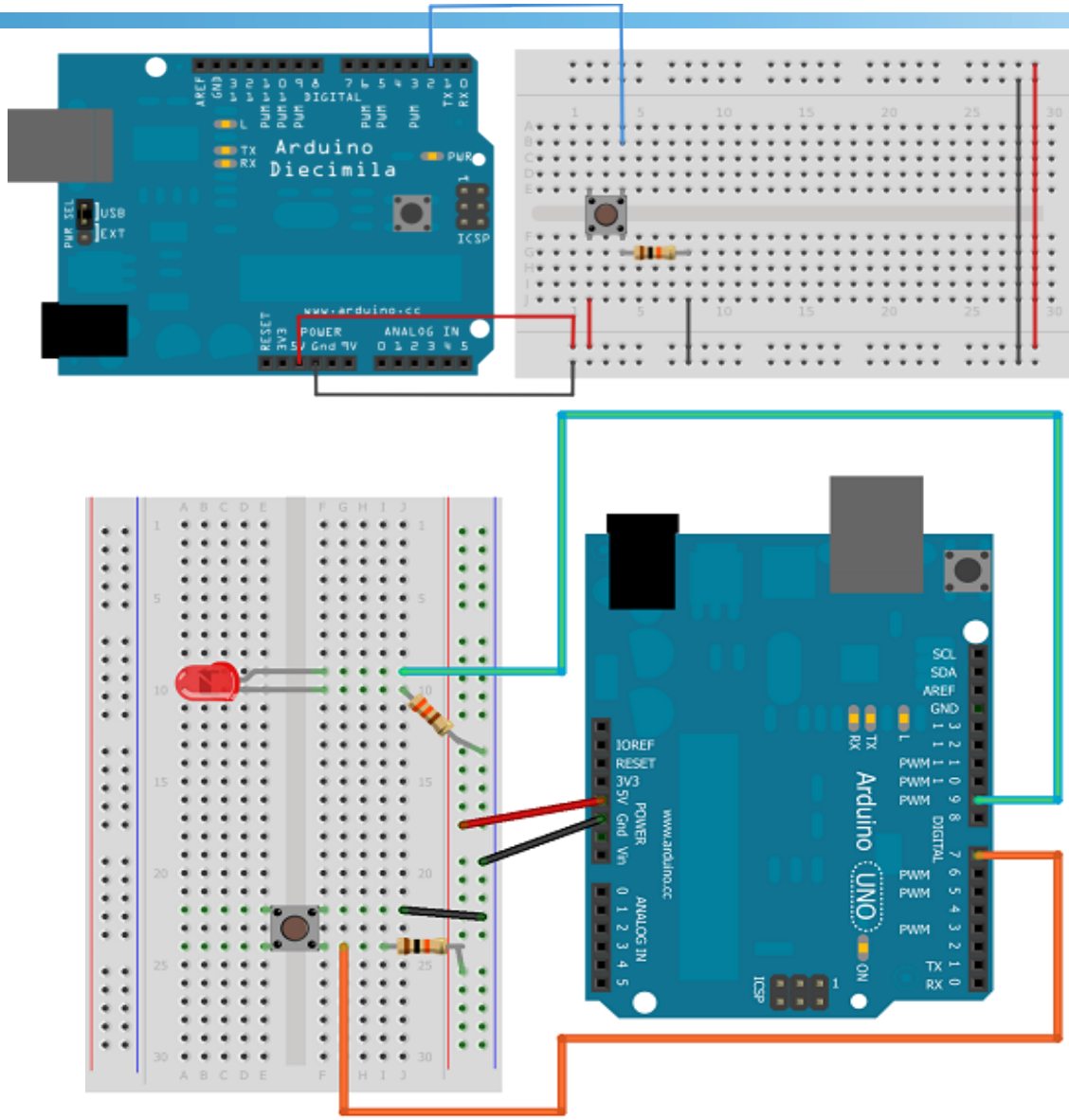

AAA Battery
AAA Battery
AAA Battery
AAA Battery
Made with Fritzing.org

Button and LED –asm control





Button and LED –asm control



LED – Button using C

```
int ledPin = 13; // choose the pin for the LED
int inPin = 7;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
    pinMode(ledPin, OUTPUT);  // declare LED as output
    pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
    val = digitalRead(inPin);  // read input value
    if (val == HIGH) {         // check if the input is HIGH
(button released)
        digitalWrite(ledPin, LOW);  // turn LED OFF
    } else {
        digitalWrite(ledPin, HIGH);  // turn LED ON
    }
}
```

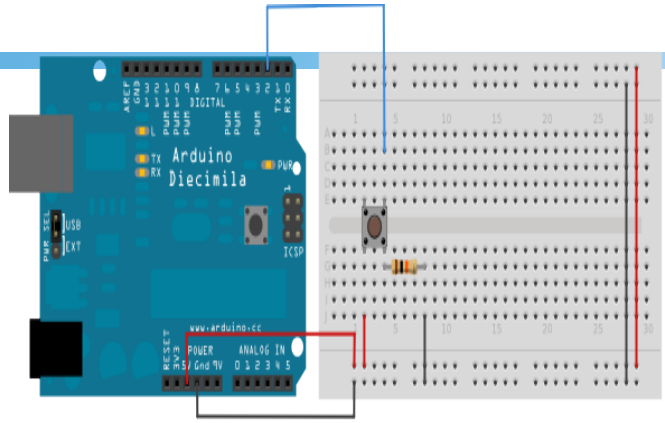
LED – Button using assembly

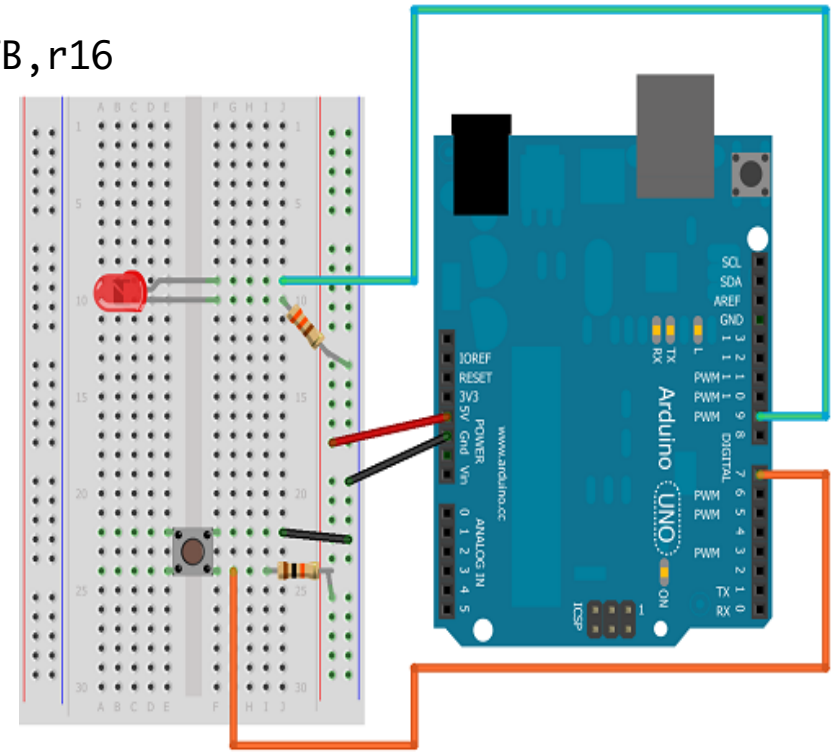
```
#include "avr/io.h"
.text
.global setup
.global loop
.global one
.global two

setup:
        ldi r16, 0b11111111
        sts DDRB, r16
        ldi r17, 0b00000000
        sts DDRD, r16

loop:
one:     lds r20, PORTD
         cpi r20, 0x00
         breq two
         sts   PORTB, r17
        jmp  one
two:  sts PORTB,r16
jmp  one
```

LED – Button using assembly

```
#define __SFR_OFFSET 0

#include "avr/io.h"
#include <avr/portpins.h>


.text
.global setup
.global loop
.global one
.global two


setup:
  ldi r16,0b11111111
  ldi r17, 0b00000000
  ldi r18,0b0100000

   out DDRB, r16
   out DDRD, r17
```

```
loop:
  out PORTB, r18

 one:

      sbis PIND,7
      brne two
      out   PORTB, r17
      jmp  one
two:  out   PORTB,r16

jmp  one
```





```
one:     lds r20, PORTD
         cpi r20, 0x00
         breq two
```

AVR Interrupts

- An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention
- An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

- Basically can be divided into internal and external interrupts

- Hardware is used to recognize interrupts

AVR Interrupts

- To enable an interrupt, two control bits must be set
  - the **Global Interrupt Enable** bit (I bit) in the Status Register
    - Using **sei** instruction
  - the **enable bit for that interrupt**

- To disable all maskable interrupts, reset the I bit in SREG
  - Using **cli** instruction

- Priority of interrupts is used to handle multiple simultaneous interrupts

Set Global Interrupt Flag - sei

- Sets the global interrupt flag (I) in SREG. The instruction following SEI will be executed before any pending interrupts.

```
sei          ; set global interrupt enable
sleep        ; enter sleep state, waiting for an interrupt
```

Clear Global Interrupt Flag - cli

- Clears the Global interrupt flag in SREG. Interrupts will be immediately disabled.

```
in r18, SREG        ; store SREG value
cli                 ; disable interrupts
                    ; do something very important here out
SREG, r18           ; restore SREG value
```

Interrupt Register

- External Interrupt Control Register A (EICRA)

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- External Interrupt Mask Register (EIMSK)

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

External Interrupt Mask Register (**EIMSK**)

Interrupts

- The interrupt execution response for all the enabled AVR interrupts is basically five clock cycles minimum.
  - For saving the Program Counter (2 clock cycles)
  - For jumping to the interrupt routine (3 clock cycles)
- The priority of an interrupt is based on the position of its vector in the program memory
- The lower the address the higher is the priority level.
- RESET has the highest priority

LED – Interrupts

```
#include <avr/interrupt.h>

.text
.global setup
.global loop
.global INT0_vect

setup:
        clr r20
// value indicate the pin13 status
        clr r16
        sts DDRD, r16
// set portD as input
        ser r16

        sts DDRB, r16
// set portB as output
        ldi r16, 0b00100000
        sts PORTB, r16
```

```
sei
ldi r16, 0b00000001
sts EIMSK, r16
// set interrupt mask as INT0
ldi r16, 0b00000011
sts EICRA, r16
// set the EICRA to response to
rising edge of INT0

loop:
        rjmp loop


INT0_vect:
    lds r20, PORTB
  cpi r20, 0x00
  breq lightup
    clr r16
  sts PORTB, r16
  reti
lightup:
    ldi r16, 0b00100000
    sts PORTB, r16
reti
```