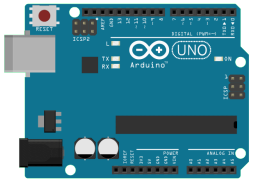
Calcolatori Elettronici T
Ingegneria Informatica

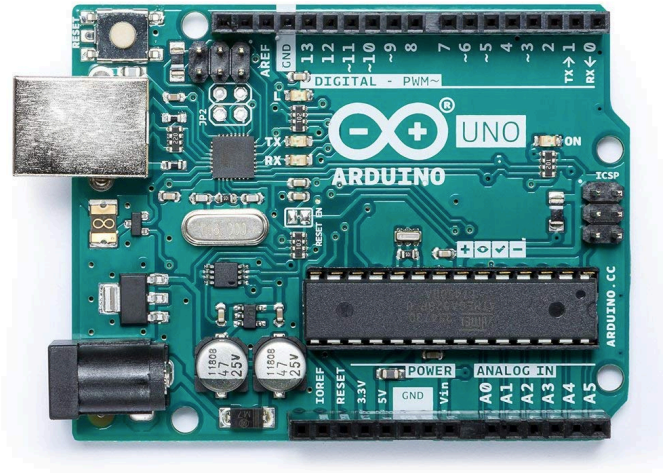
A1 – Introduzione Arduino



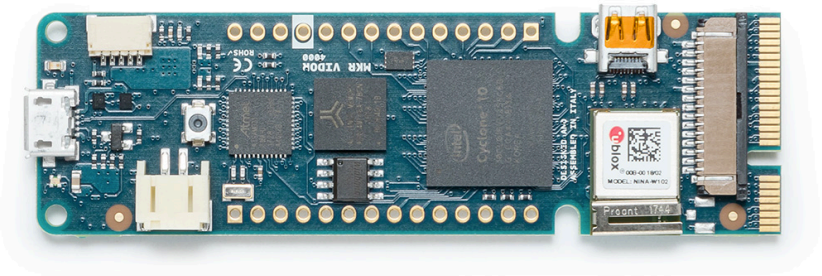
Giovanni Bonaccio, Stefano Mattoccia

Arduino

- Arduino è una piattaforma hardware e software di tipo open-source nata in Italia ma molto diffusa in tutto il mondo https://www.arduino.cc/)
- Esistono varie versioni di Arduino, noi considereremo per i progetti mostrati in questi lucidi **Arduino Uno** (https://store.arduino.cc/arduino-uno-rev3)
- Utilizzato in numerosi progetti, non solo per hobby
- Recentemente è stata proposta anche una versione dotata di FPGA (con ARM M0, in basso a destra)



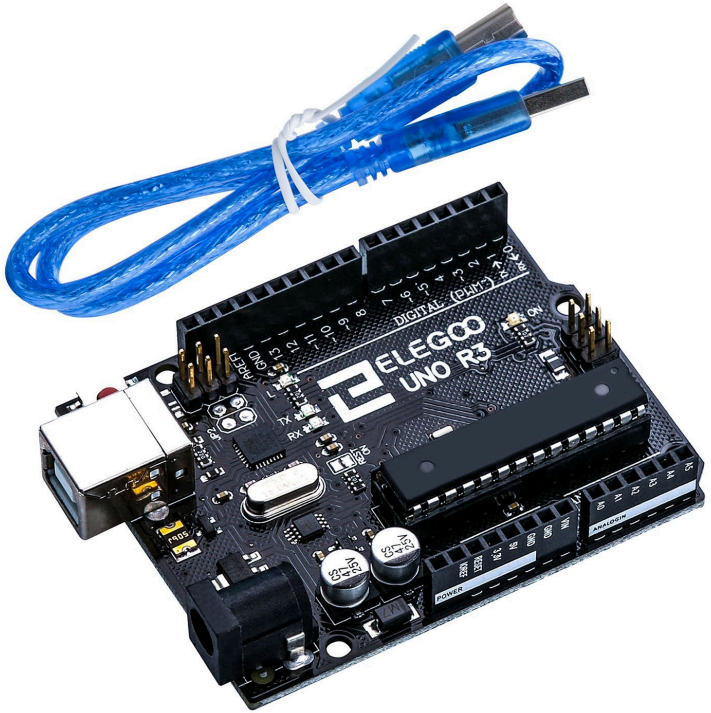
www.arduino.cc/



Arduino MKR VIDOR 4000 (FPGA)

store.arduino.cc/arduino-vidor-4000

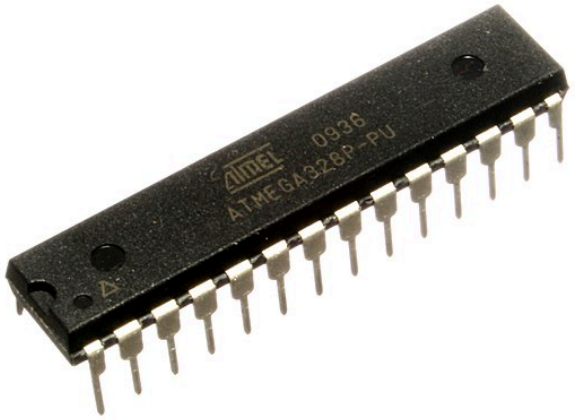
- Le schede ufficiali elencate sul sito ufficiale di Arduino consentono di supportare economicamente le evoluzioni del progetto Arduino
- Esistono però anche versioni compatibili (più o meno ufficiali)
- Una versione compatibile con Arduino Uno sperimentata con gli esempi mostrati in questi lucidi è la scheda Elegoo UNO R3



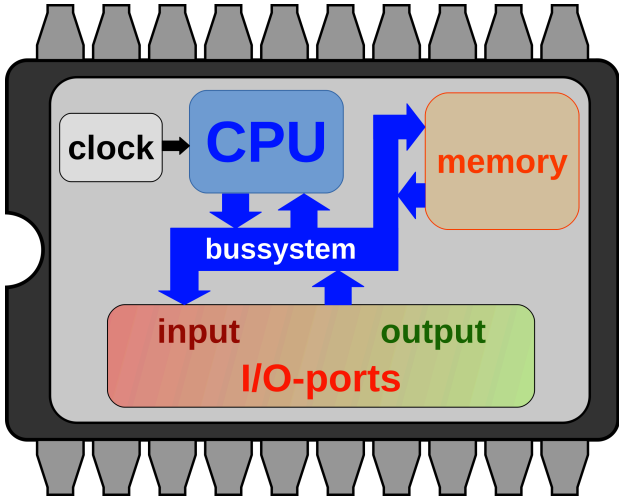
Architettura di Arduino

- Osservando la scheda si nota che Arduino Uno è basato su un solo dispositivo (**microcontrollore**) che integra al suo interno i vari moduli che costituiscono in genere un sistema a microprocessore (**RAM, EPROM, porte di IO,** etc)
- Il microcontrollore è un **ATMEL AVR ATmega 328P**
- L'ATMEL contiene al suo interno un processore **RISC a 8 bit** con frequenza di **clock di 16 MHz**
- Esistono anche versioni con CPU/microcontrollori diversi (e.g., basati su dispositivi RISC a 32 bit ARM M0 o M3) e FPGA
- Esistono anche progetti sperimentali nei quali il microcontrollore è basato su ISA RISC-V
- Indipendentemente dal modello, le porte di I/O e molte altre risorse (memorie di vario tipo, etc) sono integrate all'interno del microcontrollore

CPU/Microcontrollore ATMEL AVR



ATMEL AVR

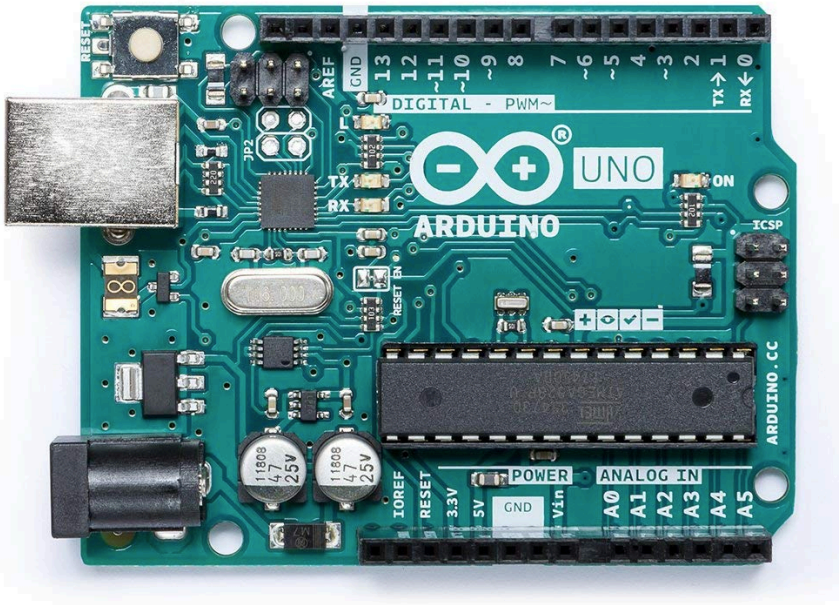


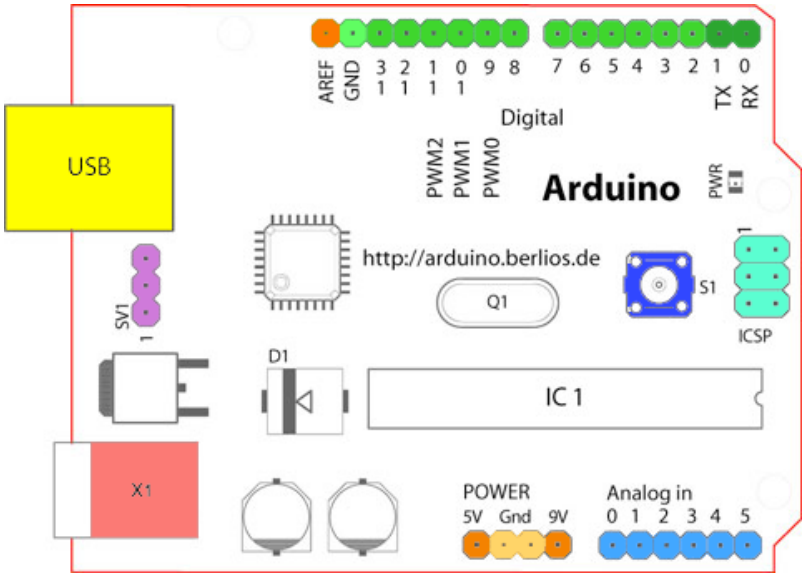
- Il *datasheet* del microcontrollore **ATMEL AVR** è disponibile a questo link:
  https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

- Una dettagliata descrizione dell'ISA **ATMEL AVR** è disponibile a questo link:
  http://www.cse.unsw.edu.au/~cs2121/LectureNotes/week4_notes.pdf

Arduino: porte di I/O

- Una delle caratteristiche fondamentali di Arduino è la possibilità di interagire con l'esterno mediante numerose porte di Input/Output (I/O) digitali e analogiche
- Per noi saranno fondamentali le **porte digitali**
- Ovviamente è utilizzare gli **interrupt**



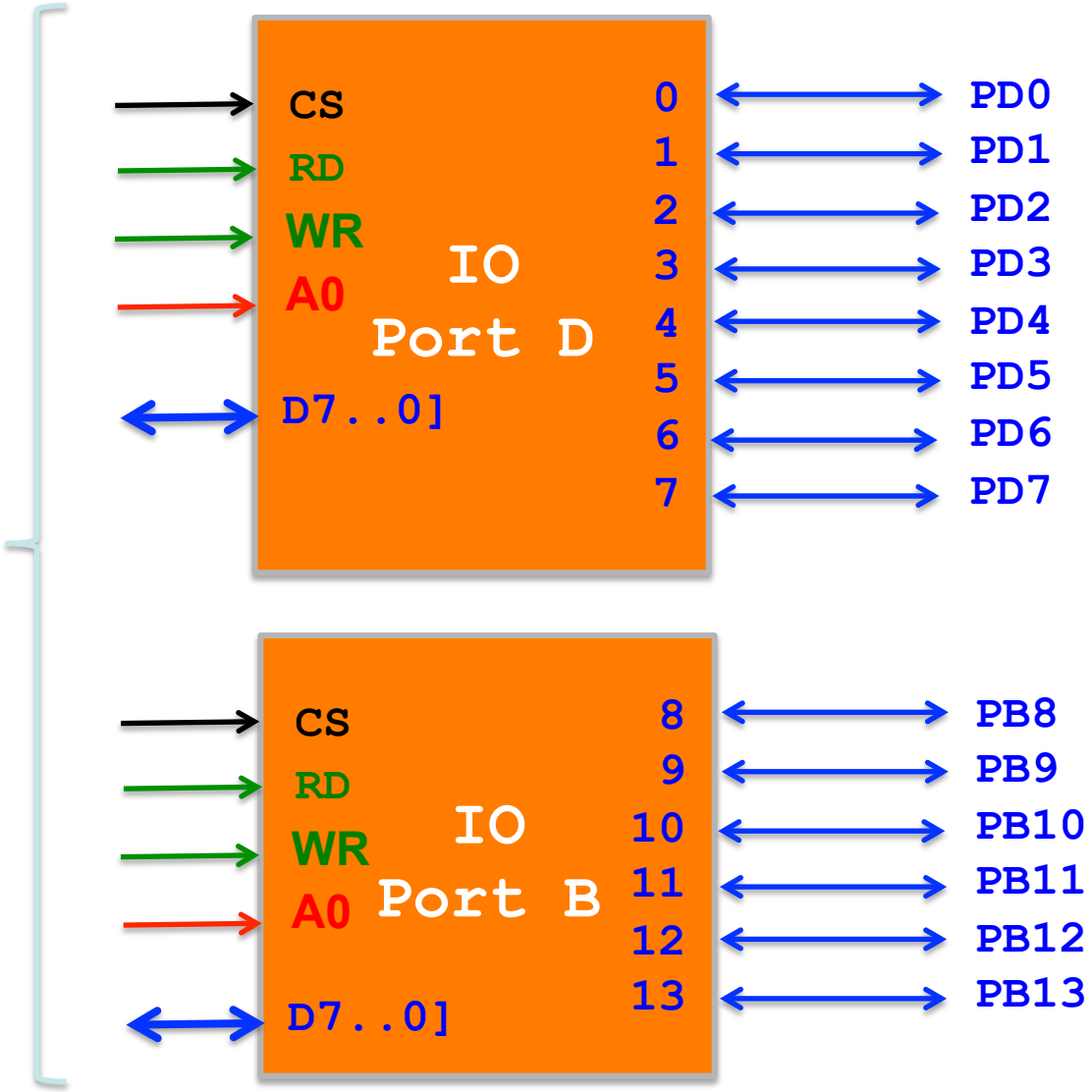


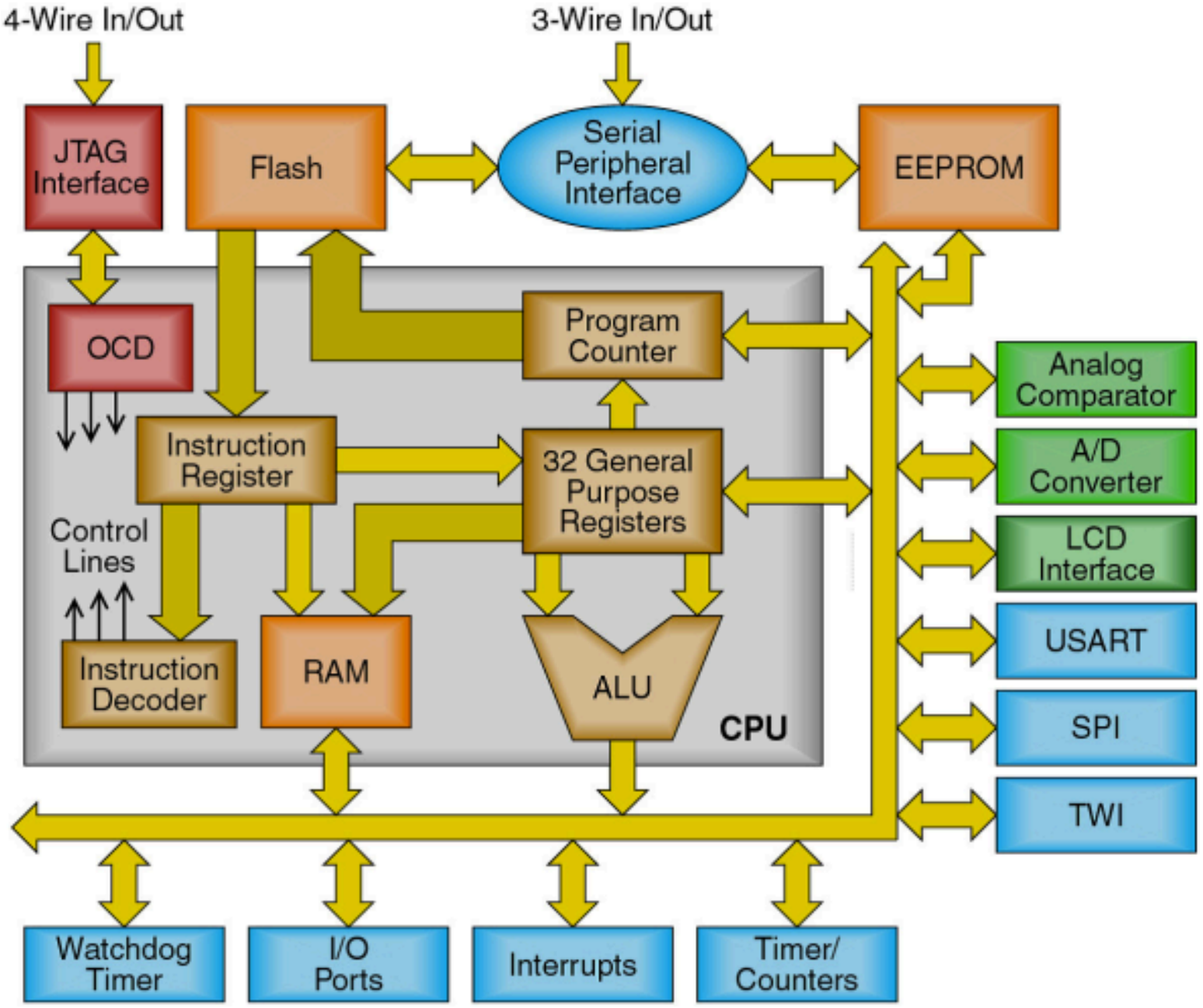
https://www.arduino.cc/en/Reference/Board

Porte di I/O digitali

Arduino dispone di due porte, B e D, per input e output di segnali digitali. Entrambe le porte sono programmabili

I segnali utilizzati per la connessione alla CPU sono diversi ma codificano le stesse informazioni indicate a destra (i.e., CS, RD, WR, indirizzi e dati)





https://nishantnath.wordpress.com/2012/03/23/introduction-to-atmega-microcontrollers/

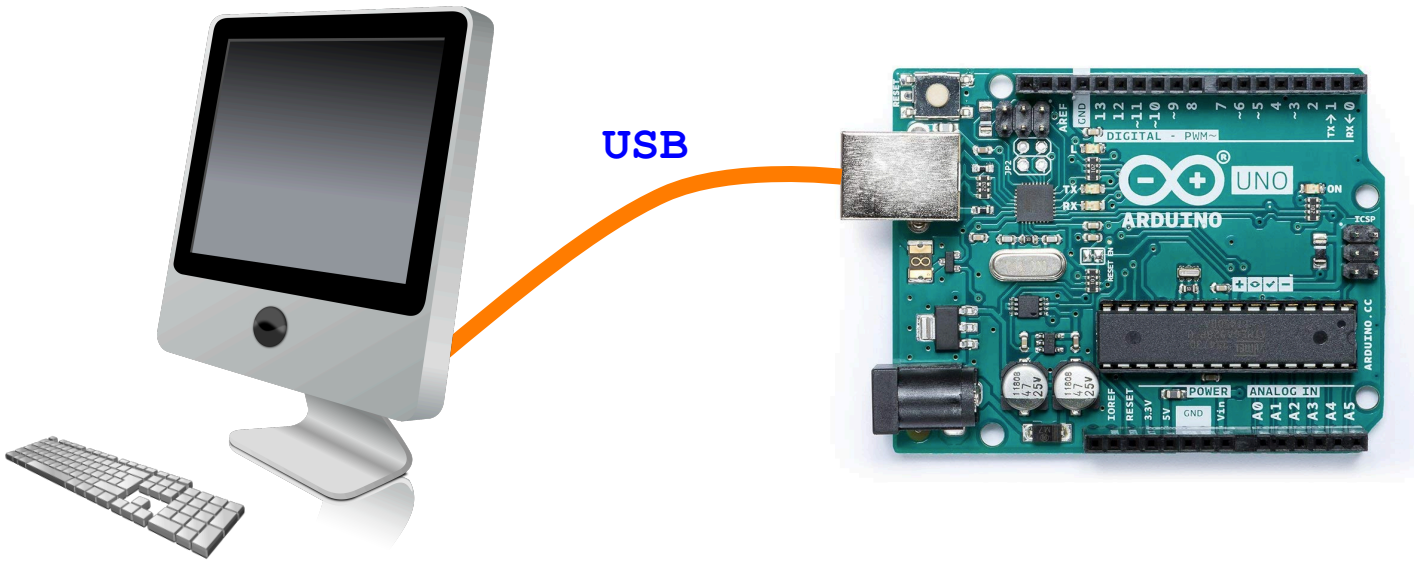
Strumenti di sviluppo software

- Uno dei motivi dell'ampia diffusione di Arduino consiste nella disponibilità di strumenti software per lo sviluppo

- Esistono tre modalità di programmazione:
  - **Sketch**
  - **C/C++**
  - **Assembler**
- Per noi la modalità di programmazione ideale sarebbe quella basata sul linguaggio Assembler
- Tuttavia, questo implicherebbe studiare un nuovo ISA
- Per questo ragione, gli esempi saranno mostrati in C

Macchina virtuale (VM) pre-configurata

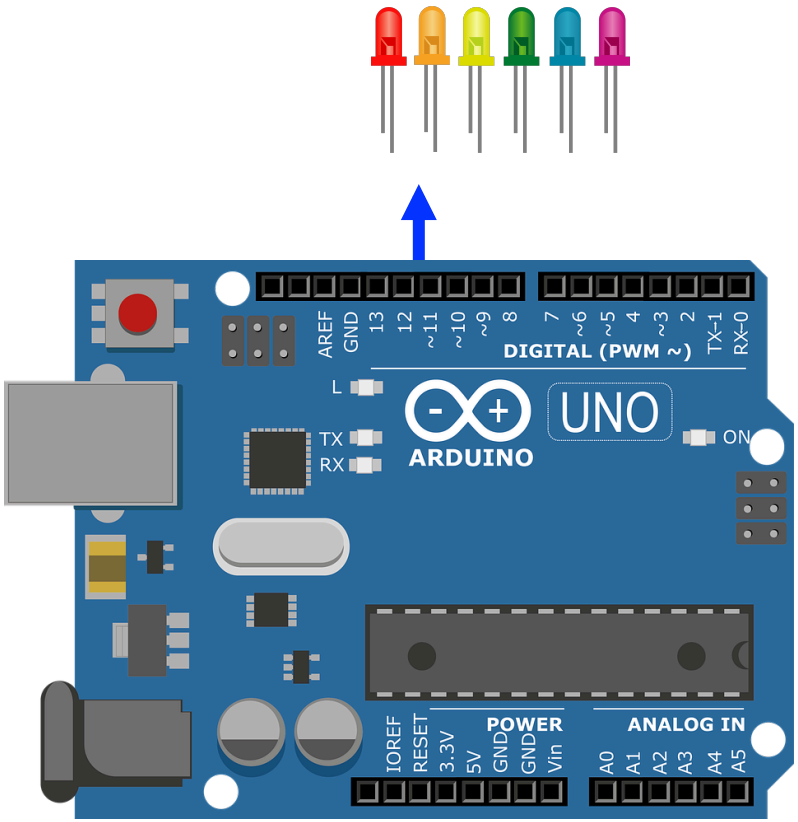
- Lo **sviluppo** software avviene **su PC** e poi il **codice eseguibile è trasferito (via USB) sulla scheda Arduino.** Questo processo si chiama ***cross-compilazione***
- Per facilitare l'utilizzo agli interessati è disponibile sul sito del corso una **VM Linux Ubuntu 16.04 a 64 bit** (VirtualBox) già pre-configurata per lo sviluppo software (**Mac, Linux** e **Windows**)
- Il compilatore utilizzato è una variante di *gcc* per il microcontrollore ATMEL



Lampeggio di un led

Vediamo con un semplice esempio come far lampeggiare un led con la scheda Arduino.

E' un esempio semplice, **ma come si procede?**



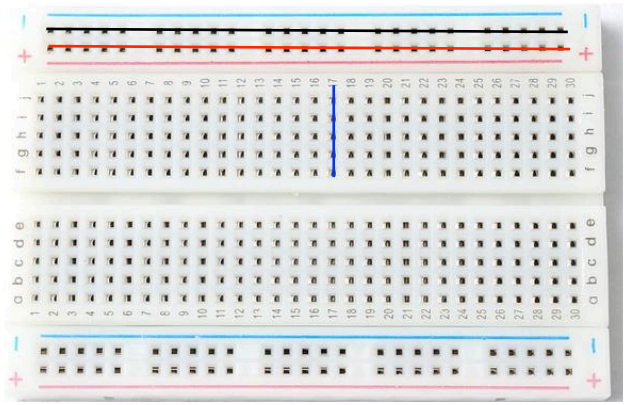
Sviluppo software: procedimento

Per ogni progetto occorrerà:

- Effettuare gli opportuni collegamenti hardware
- Implementare il **main** che ne permetta il funzionamento richiamando delle **librerie relative alle periferiche**
- Il codice delle librerie, sviluppate per queste esercitazioni, è disponibile all'interno della macchina virtuale

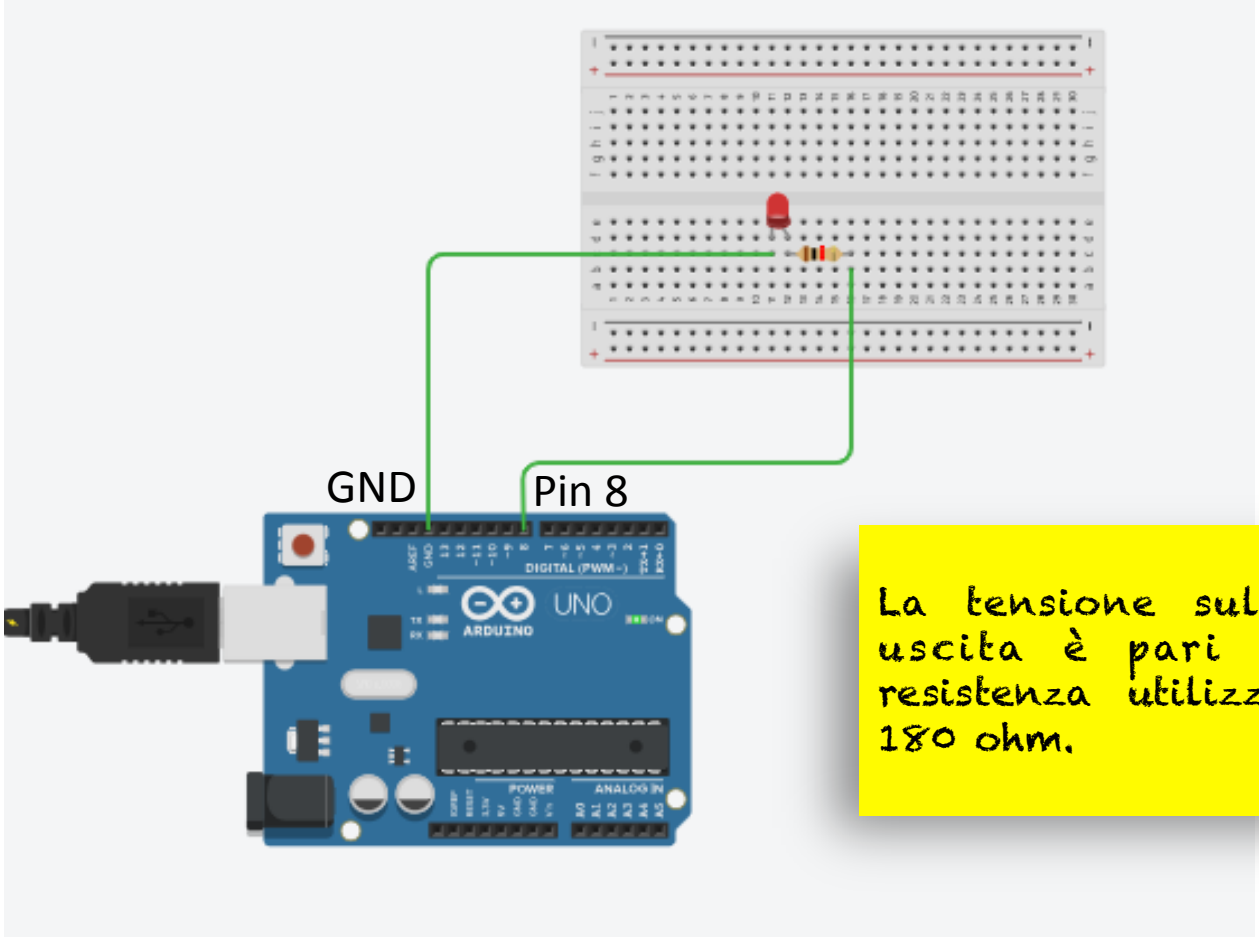
I componenti utilizzati saranno:

- **Led** (e una resistenza)
- **Breadboard** *(a destra)*



Collegamenti hardware

Effettuare i collegamenti con i **pin** avendo cura di utilizzare una **resistenza**, per evitare di bruciare il Led a causa dell'eccessiva corrente



La tensione sul pin8 di uscita è pari a 5V. La resistenza utilizzata è da 180 ohm.

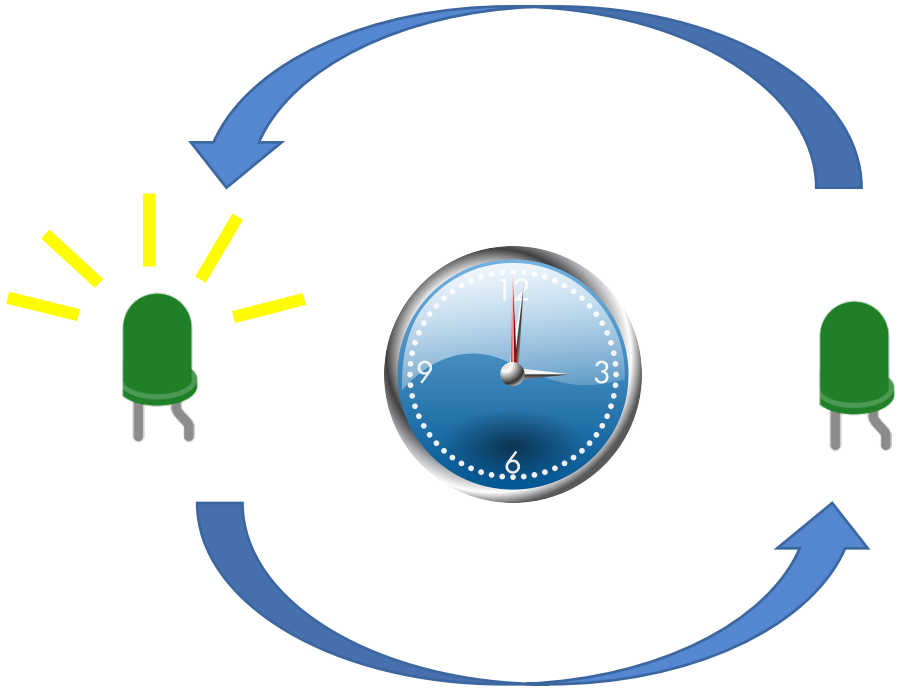
https://www.tinkercad.com

Codice

```
#include "lampeggioLED.h"

void main(void)
{
    //  Output: Bit 0 della porta b (pin 8)
    //  Imposta pin 8 come output
    port_setup_set_data_direction_output_bits_port_B(0b00000001);

while(1)
    {
        // alterna lo stato del bit 0 della porta b
        // accende il LED
        port_set_high_bits_port_B(0b00000001);
        // ritardo di 1000 ms
        _delay_ms(1000);
        // spegne il LED
        port_set_low_bits_port_B(0b00000000);
        // ritardo di 1000 ms
        _delay_ms(1000);
    }
}
```

Libreria lampeggioLED.h

Per questi esperimenti non è risultato necessario la dichiarazione di ulteriori variabili o metodi, perciò la libreria definita per questo progetto, **lampeggioLED.h**, non fa altro che richiamare un'altra libreria denominata **common_commands.h.**



Libreria common_commands.h 1/2

- Dichiara i metodi comuni a tutti i progetti
- Richiama le librerie di sistema per il compilatore del microcontrollore **AVR**
- Definisce il tipo di dato **BYTE**
- Consente di definire degli **interrupt handler** (che riprenderemo più avanti per altri progetti)

La libreria definisce le seguenti funzioni:

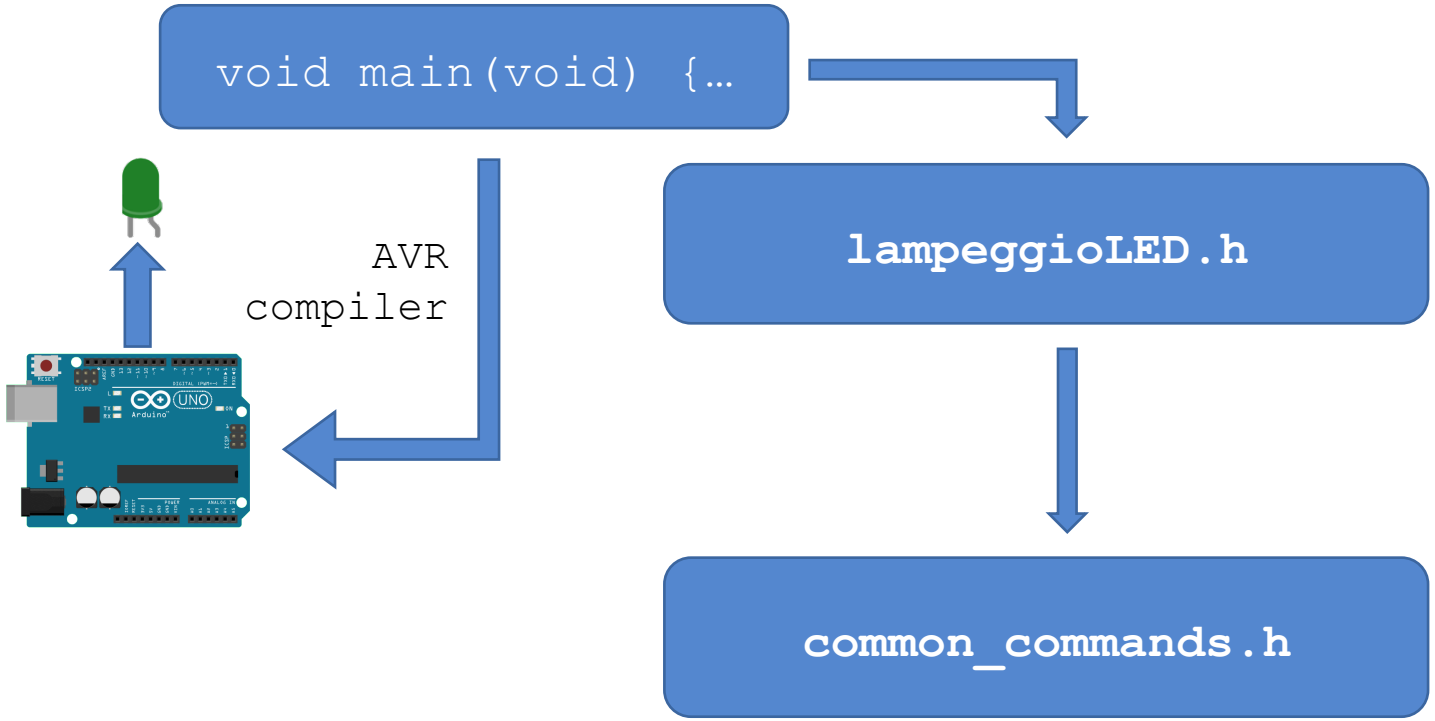
- **void port_setup_set_data_direction_input_bits_port_X (BYTE code)**: consente di impostare i pin in input della trasmissione dei dati per la porta X
- **void port_setup_set_data_direction_output_bits_port_X (BYTE code)**: consente di impostare i pin in input della trasmissione dei dati per la porta X

Libreria common_commands.h 2/2

- **void port_set_low_bits_port_X(BYTE code)**
  Assegna il valore logico LOW (0) ai bit della porta X. Gli altri valori sono lasciati inalterati
- **void port_set_high_bits_port_X(BYTE code)**
  Assegna il valore logico HIGH (1) ai bit della porta X. Gli altri valori sono lasciati inalterati
- **void port_set_all_bits_port_X(BYTE code)**
  void port_set_all_bits_data_direction_port_B(BYTE code)
  Stesso compito delle funzioni precedenti, la differenza consiste nel mancato utilizzo delle operazioni AND (**&**) e OR (**|**). Pertanto, tutti i bit sono impostati con il valore **BYTE code** sovrascrivendo tutti i valori.

La libreria definisce anche funzioni per la **gestione degli interrupt** che non saranno utilizzate per questi primi progetti e discusse in seguito

Panoramica



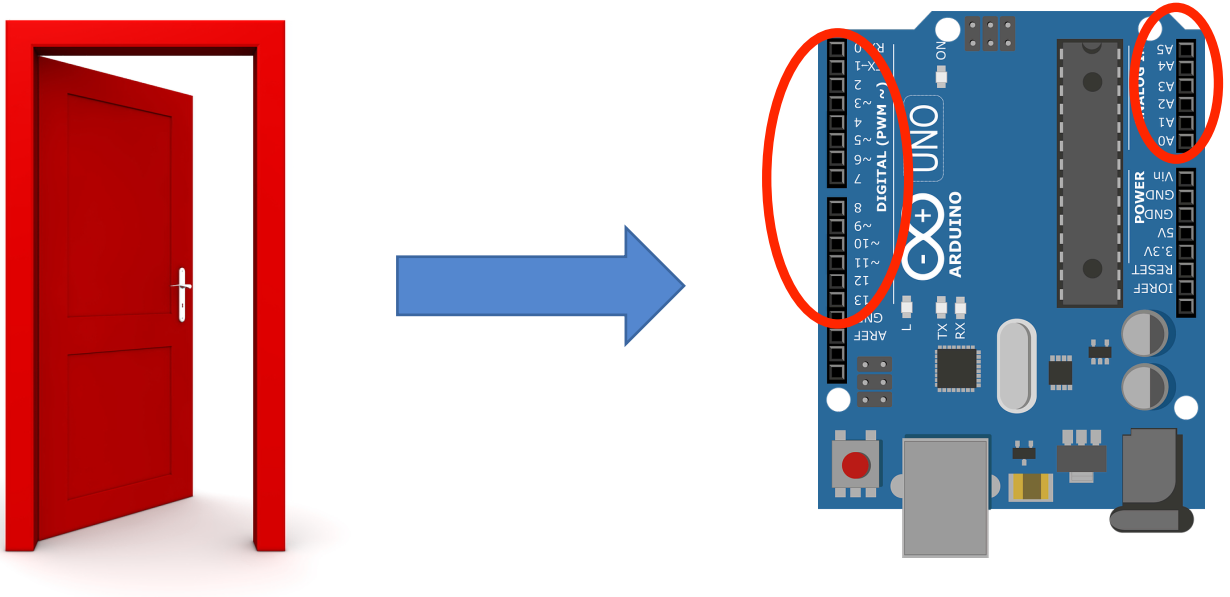
La libreria **common_commands.h** contiene le funzioni comuni a tutti i progetti

Tutte le librerie utilizzate sono incluse nella VM disponibile sul sito del corso

Pin mapping

Per accedere ai pin e assegnare quindi loro un valore logico, utilizziamo le **porte di I/O:**

- **Porta B**: comprende i pin digitali da 8 a 13
- **Porta C**: comprende i bit analogici A0-A5
- **Porta D**: comprende i pin digitali 0 al 7



Porte e registri

Ogni porta permette di manipolare i relativi pin attraverso opportuni **registri** mappati a opportuni indirizzi di memoria

In Arduino Uno sono presenti tre tipi di registri. Prendendo come riferimento la porta B, essi sono:

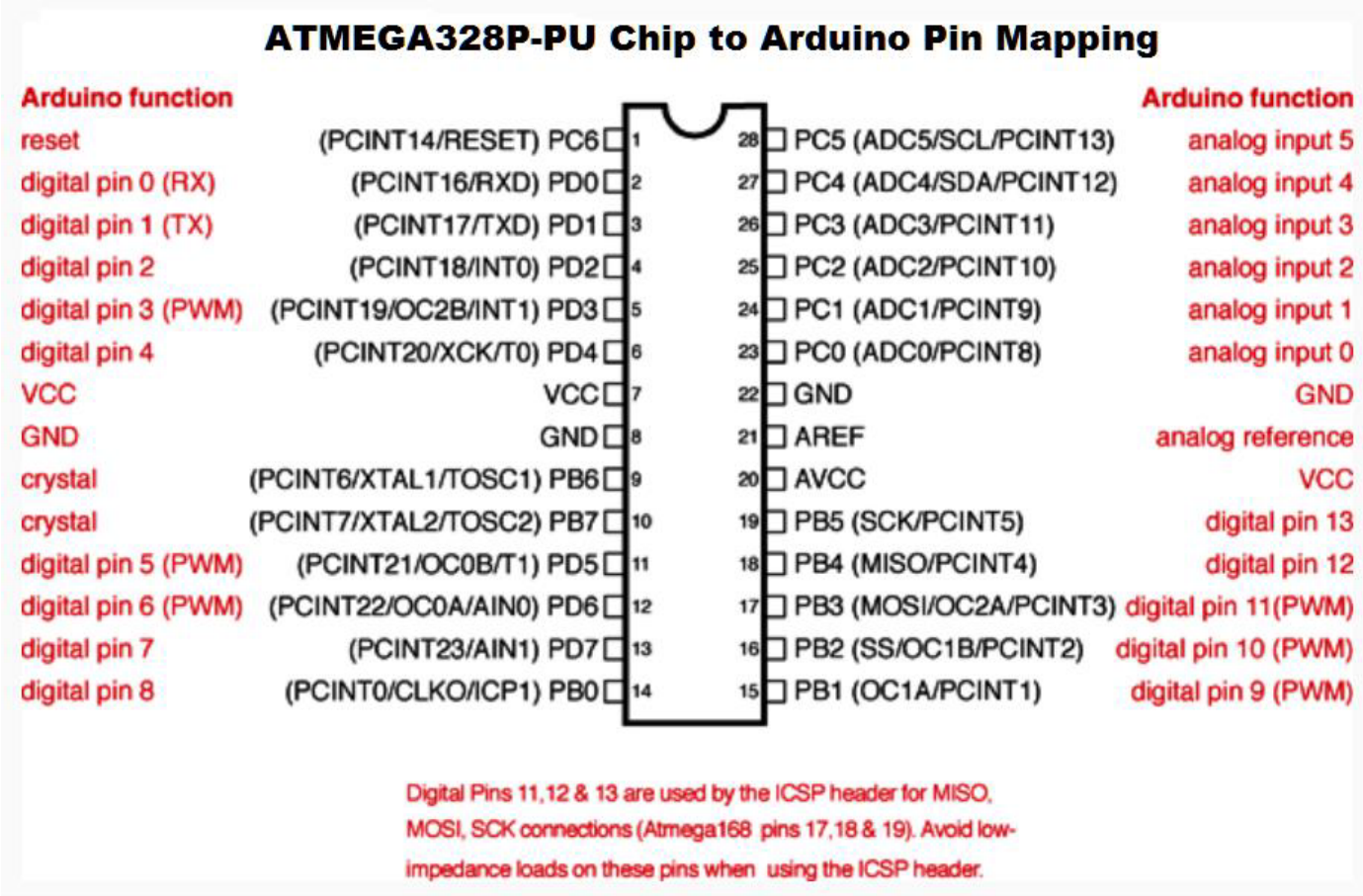
- **DDRB**: registro che permette di settare un pin come input o output

- **PORTB**: registro che permette di assegnare un valore a un pin in output

- **PINB**: registro che permette di leggere il valore di un pin in input. Su questo registro è permessa la sola lettura e non la scrittura

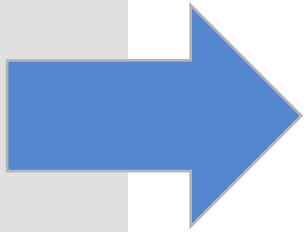
Pin mapping

- È possibile comunicare con i registri con un parallelismo di 8 bit. Ogni bit di una porta si riferisce a un pin di Arduino
- Consideriamo ad esempio il pin 9: esso corrisponde a PD7, ovvero bit 7 della Porta D.

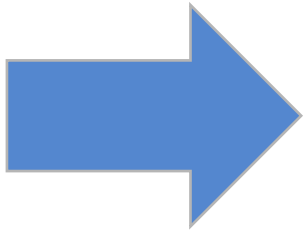


Codice 1/2

```
#include "lampeggioLED.h"

void main(void)
{
     //  Output: Bit 0 della porta b (pin 8)
     //  setup del pin 8 come output
    port_setup_set_data_direction_output_bits_port_B(0b00000001);


    while(1)
    {
         //  alterna lo stato del bit 0 della porta b
         // accende il LED
        port_set_high_bits_port_B(0b00000001);

         // ritardo di 1000 ms
        _delay_ms(1000);

         // spegne il LED
        port_set_low_bits_port_B(0b00000000);

         // ritardo di 1000 ms
        _delay_ms(1000);
    }

}
```



Codice 2/2



```
void port_setup_set_data_direction_output_bits_port_B(BYTE code)
{
     DDRB = DDRB | code;
}

void port_set_high_bits_port_B(BYTE code)
{
     PORTB = PORTB | code;
}


void port_set_low_bits_port_B(BYTE code)
{
     PORTB = PORTB & code;
}
```

Da notare l'utilizzo di operatori **AND(&)** e **OR(|)** che permettono cambiare i bit desiderati lasciando inalterati gli altri

Domanda

Come potrebbe essere realizzata (in software e hardware) la funzione **_delay(1000)**? E **_delay(K)**, con K intero?

Compilazione ed esecuzione del codice

In Linux, da un terminale è necessario per prima cosa compilare il *codice.c*:

avr-gcc -g -Os -mmcu=atmega32 -c codice.c

-Os indica al compilatore di minimizzare la dimensione del codice generato

-g indica di inserire informazioni di debug

Successivamente (*linking*) si crea il file *codice.elf*:

avr-gcc -g -mmcu=atmega32 -o codice.elf codice.o

Infine, il codice può essere trasferito ed eseguito in Arduino mediante:

avr-objcopy -j .text -j .data -O ihex codice.elf codice.hex

Codice macchina/assembler e makefile

E' possibile visualizzare il codice macchina/assembler generato dal compilatore avr-gcc mediante il seguente comando:

avr-objdump -h -S codice.elf > codice.txt

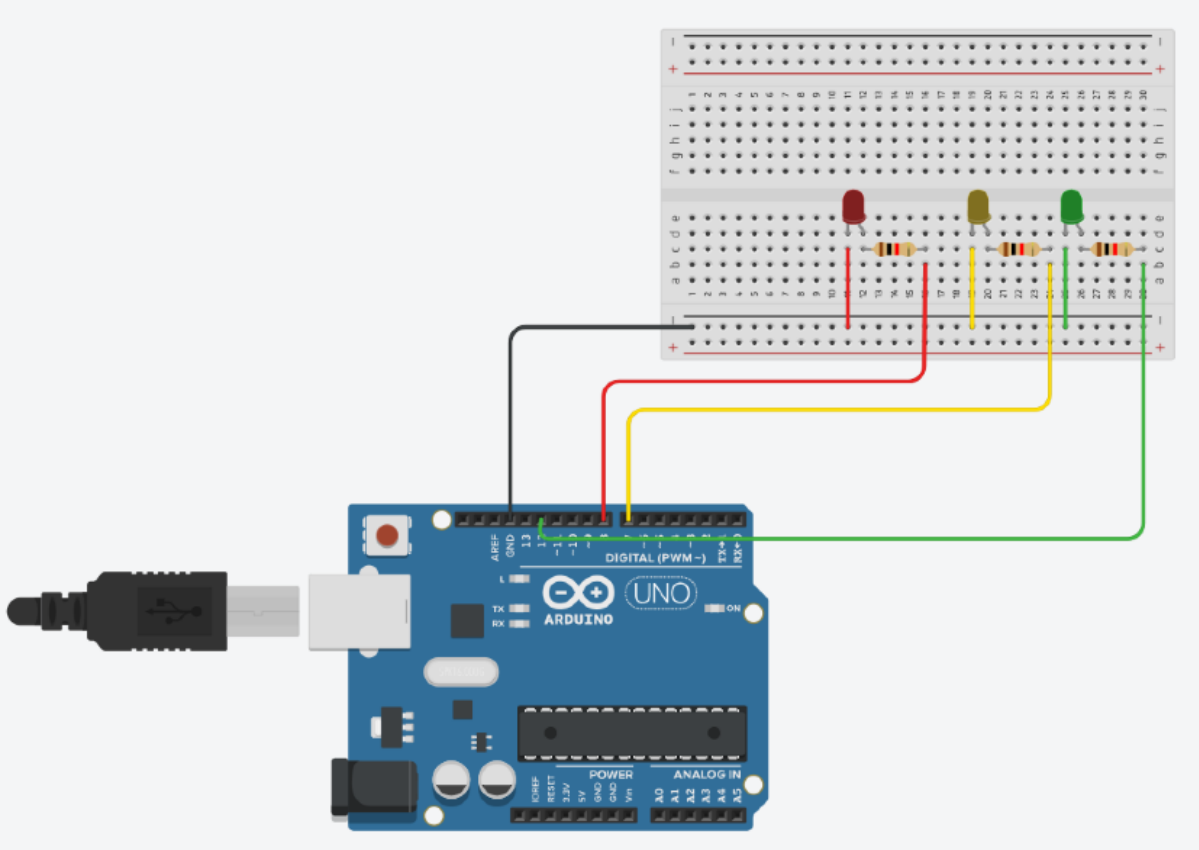
Per evitare di digitare continuamente questi comandi è possibile creare un ***makefile*** che automatizza il procedimento descritto in precedenza. Il comando *make* è molto utile e ampiamente utilizzato.

Un esempio di *makefile* per il compilatore avr-gcc può essere creato come mostrato in questo link:

gist.github.com/SkypLabs/c32ce01c695620e033443d407544f21e

Semaforo: connessioni

In questo caso desideriamo gestire le fasi di un semaforo. Il progetto è diverso ma adottiamo lo stesso principio utilizzato nell'esempio precedente.



Semaforo: codice

```
void main(void)
{
    //  Output:
    // Bit 0 della porta B (pin 8) [LED ROSSO]
    // Bit 7 della porta D (pin 7) [LED GIALLO]
    // Bit 4 della porta B (pin 12) [LED VERDE]

    // setup del bit 0 e 4 della porta b, bit 7 della porta D come OUTPUT
    port_setup_set_data_direction_output_bits_port_B(0b00010001);
    port_setup_set_data_direction_output_bits_port_D(0b10000000);

    while(1)
    {
        //  alternanza led
        port_set_high_bits_port_B(0b00000001);
        _delay_ms(5000);
        port_set_low_bits_port_B(0b11111110);
        port_set_high_bits_port_B(0b00010000);
        _delay_ms(5000);
        port_set_low_bits_port_B(0b11101111);
        port_set_high_bits_port_D(0b10000000);
        _delay_ms(2000);
        port_set_low_bits_port_D(0b01111111);
    }
}
```