LSS 2018-19
Reti Logiche: introduzione ad Arduino

Piero Vicini
(credits: Prof. A. Nigro (UniRoma1) & Prof. P. Visconti (UniSalento))

A.A. 2018-2019

Calcolatori

- Prodotto di una tecnologia estremamente vitale con alto impatto economico e sociale
- Tecnologia pervasiva: calcolo, controllo,....
- ... che rende possibili nuove applicazioni
  - Calcolatori nelle automobili
  - Telefoni cellulari
  - Mappatura del genoma umana
  - Imaging medico
  - WorldWideWeb e motori di ricerca
  - Approccio alla risoluzione di problemi di fisica (biologia, chimica, geologia,...) tramite simulazioni al computer
- .....

Calcolatori e tecnologia

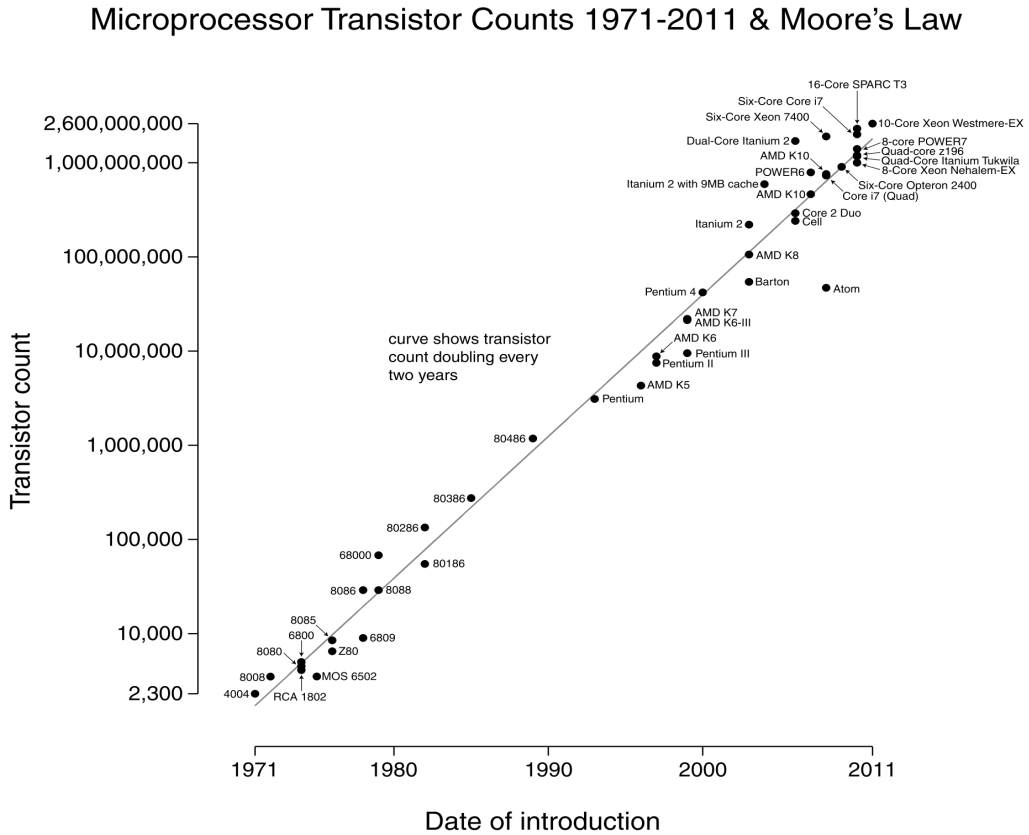
http://www.library.upenn.edu/special/gallery/mauchly/jwmintro.html

1943: ENIAC primo mainframe programmabile
18000 valvole == 5000 transistors

DOWNSIZING AND UPGRADING

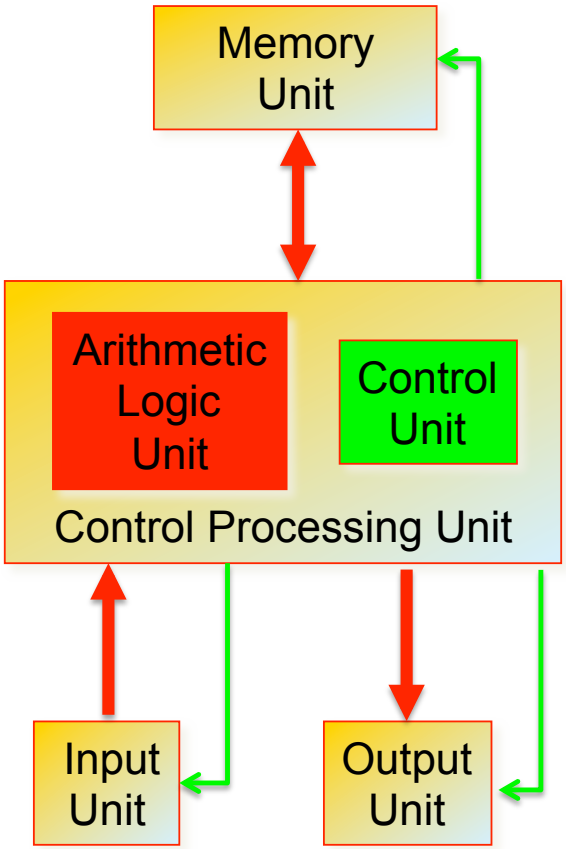
The inception of computing inspired a remarkable race for faster, smaller, lighter, cheaper hardware.

	ENIAC	Intel Core Duo chip
Debut	1946	2006
Performance	5,000 addition problems/sec	21.6 billion ops/sec
Power use	170,000 watts	31 watts max
Weight	28 tons	negligible
Size	80' w x 8' h	90.3 sq. mm.
What's inside	17,840 vacuum tubes	151.6 M transistors
Cost	$487,000	$637



Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

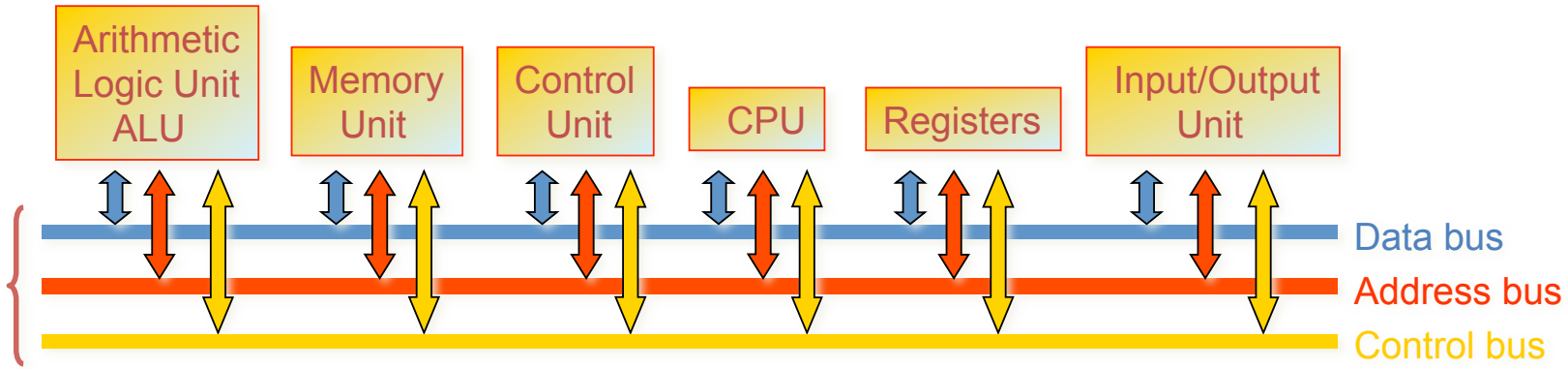
Architettura di un calcolatore



Modello di John von Neumann

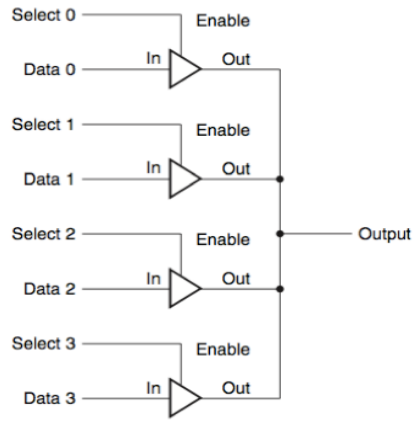
- Un *calcolatore* si distingue da una macchina calcolatrice perche' e' *programmabile i.e.* la sua funzionalita' dipende da un *codice esterno* e non dalla configurazione del sistema
  - Il sommatore basato su AmpOp e' un calcolatore?
- Il suo hardware e' in grado di eseguire diversi compiti eseguendo la sequenza di *istruzioni* contenute in un *programma*.
- Secondo il modello di *Von Neumann* un calcolatore deve essere composto da:
  - *CPU* (Control Processing Unit)
    - Blocco aritmetico (esegue calcoli…)
    - Unita' per controllo e sincronizzazione dei vari componenti
  - Unita' di *I/O* (input/output)
    - Tastiera, mouse, network…
    - Display, stampanti, diffusori audio, network…
  - Unita' di *memoria*
    - *Cache*, RAM, Hard Disk,…

Architettura a bus



- Le unita' funzionali si scambiano informazioni utilizzando strutture condivise: i *BUS*
- Il BUS e' una collezione di linee elettriche con un protocollo di comunicazione che permette di interpretare correttamente e sincronizzare le varie operazioni di trasferimento dati
- Le singole unita' funzionali e/o i loro componenti interni sono univocamente determinati da un *indirizzo*

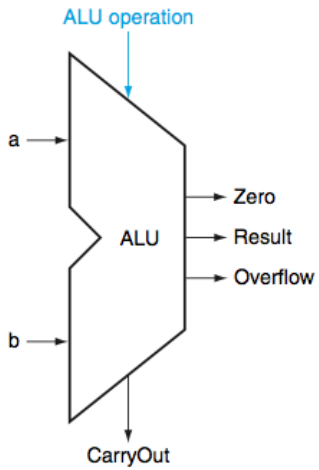
- La *condivisione* di un bus e' possibile grazie a *logiche tri-state* che permettono ad agenti diversi di pilotare lo stesso filo (ovviamente in momenti diversi….)

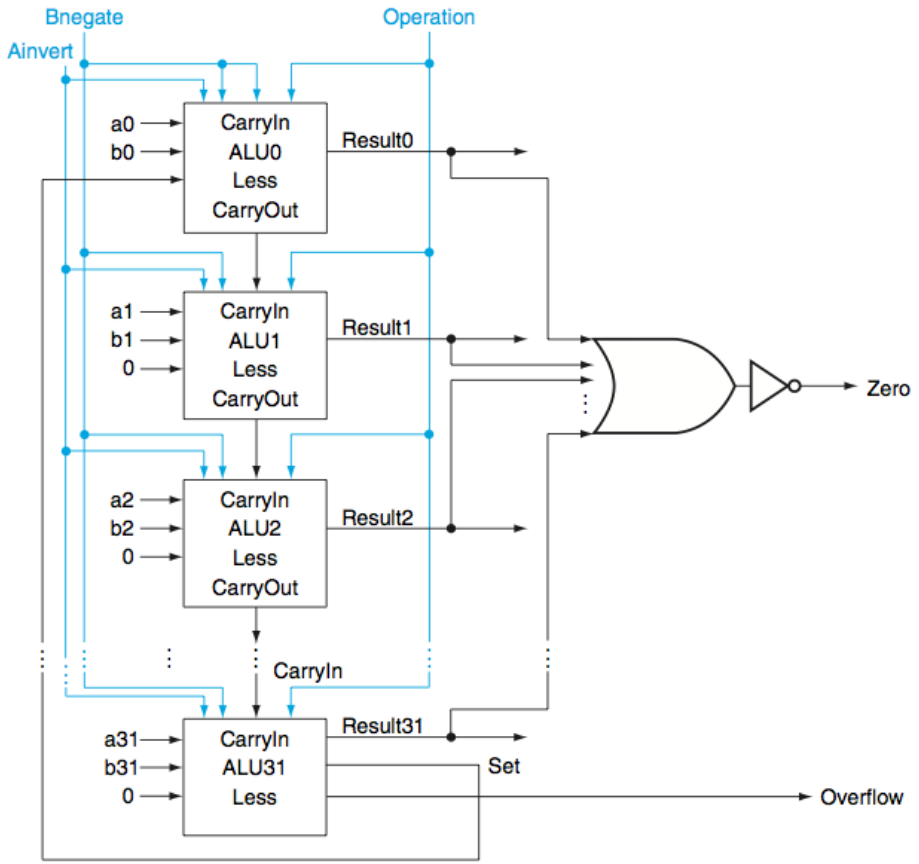


ALU

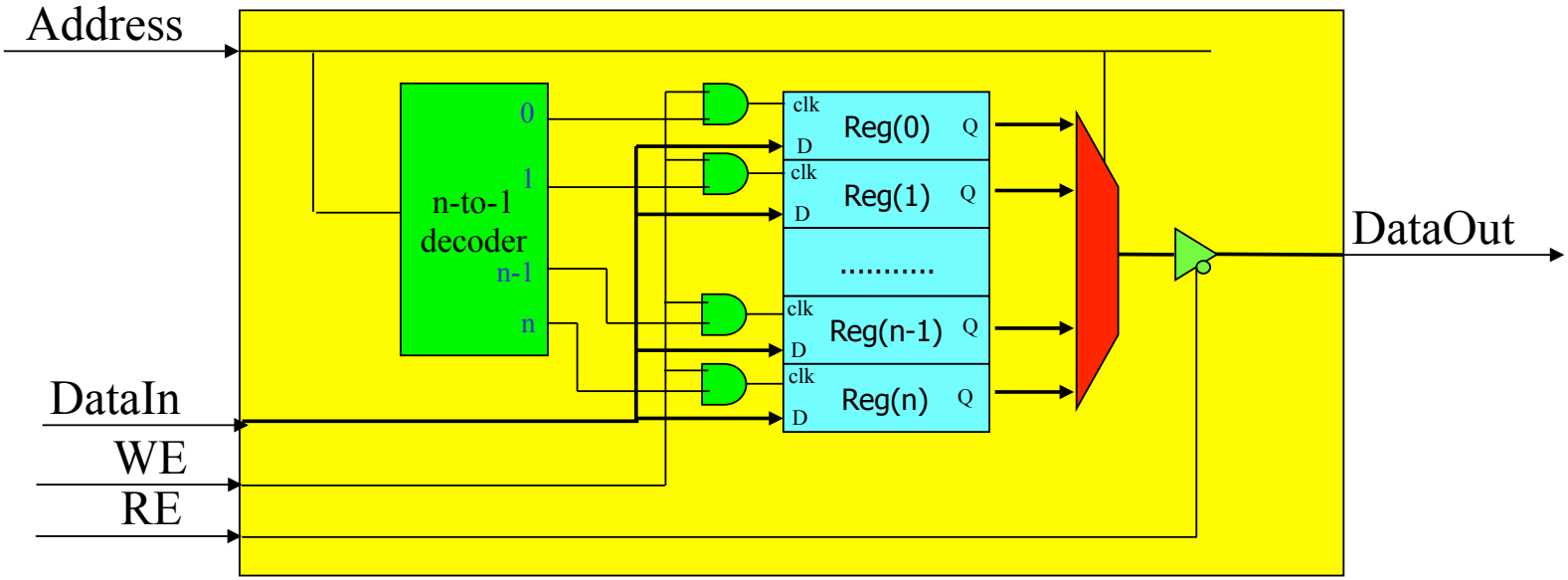
- ALU (*Arithmetic Logic Unit*) esegue operazioni aritmetico/logiche su parole ad n-bit.
  - Somma (*add*) , sottrazione (*sub*), confronto (*slt, beq, bne*), logiche (*and, or*).

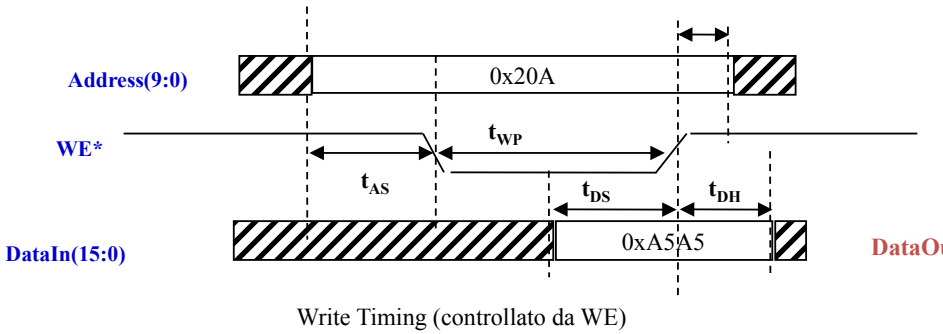
- *1-bit ALU* e' l'unita' elementare composta da un *Full Adder* (FA) (che abbiamo gia' visto), un blocco per esecuzione operazioni logiche e da un circuito di controllo.
- (*Operation*) seleziona l'operazione da eseguire



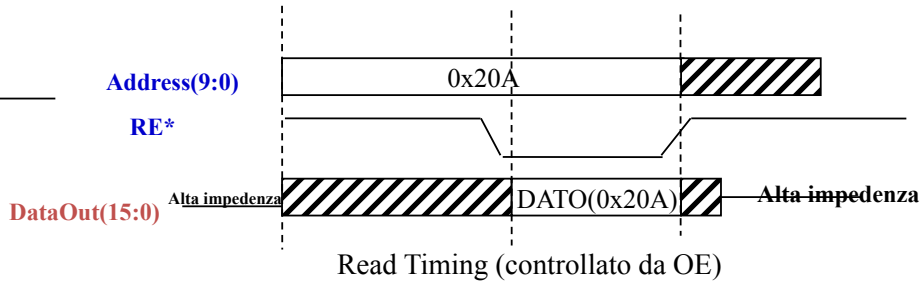


Memoria RAM (Random Access Memory)





Write Timing (controllato da WE)



Read Timing (controllato da OE)

Livelli di astrazione di un'architettura di calcolo

- Un architettura di calcolo puo' essere scomposta in diversi *livelli di astrazione*
- Questa descrizione definisce interfacce chiare tra funzionalita' diverse nascondendo i dettagli del singolo livello
    - Un cambiamento di un componente di un certo livello non comporta (non dovrebbe comportare…) cambiamenti negli altri livelli
- L'*astrazione* cresce dai livelli hardware fino al livello applicativo

Il linguaggio dei calcolatori: *Instruction Set*

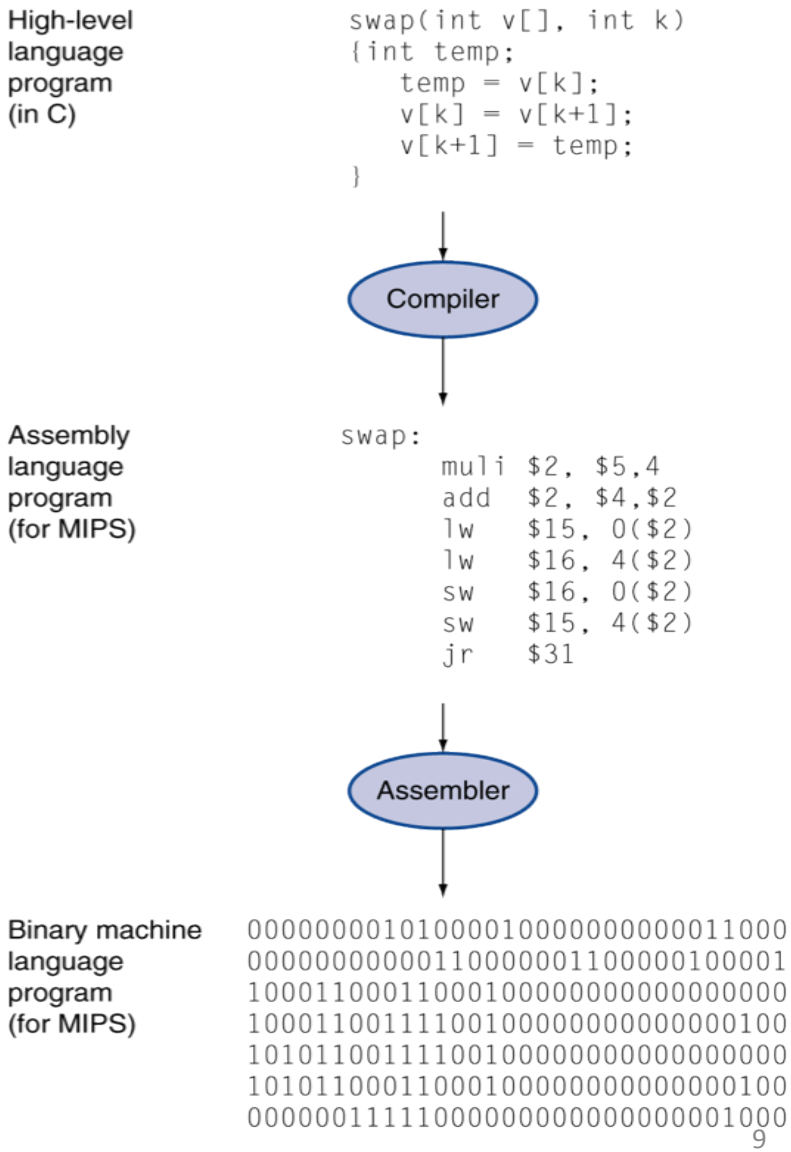
- *L'Instruction Set* (**IS**) e' l'insieme delle istruzioni del processore i.e. il suo "vocabolario"
- Riflette l'architettura interna del processore
- Ogni processore ha il suo IS specifico *(http: //en.wikipedia.org/wiki/List_of_instruction_sets)*
- Esistono varie categorie di IS che si differenziano per la loro struttura
  - ***CISC*** (Complex Instruction Set Computer), **RISC** (Reduced....), **VLIW** ma anche cose piu' esotiche quali **ZISC** e **NISC**…
- Comunque i vari IS hanno molti aspetti in comune…
  - Istruzioni per operazioni aritmetiche (`add, sub, inc, cp,...`)
  - Istruzioni per operazioni aritmetico/logiche (`and, or, shift, rotate,...`)
  - Istruzioni per trasferimento dati tra registri e memoria (`load, store,...`)
  - Istruzioni per *salti condizionati e incondizionati* (`jump, call,...`)
  - Istruzioni per operazioni di I/O(`in, out,...`)
  - Istruzioni per gestione della CPU (`nop, halt,...`)

Il linguaggio dei calcolatori: *il programma*

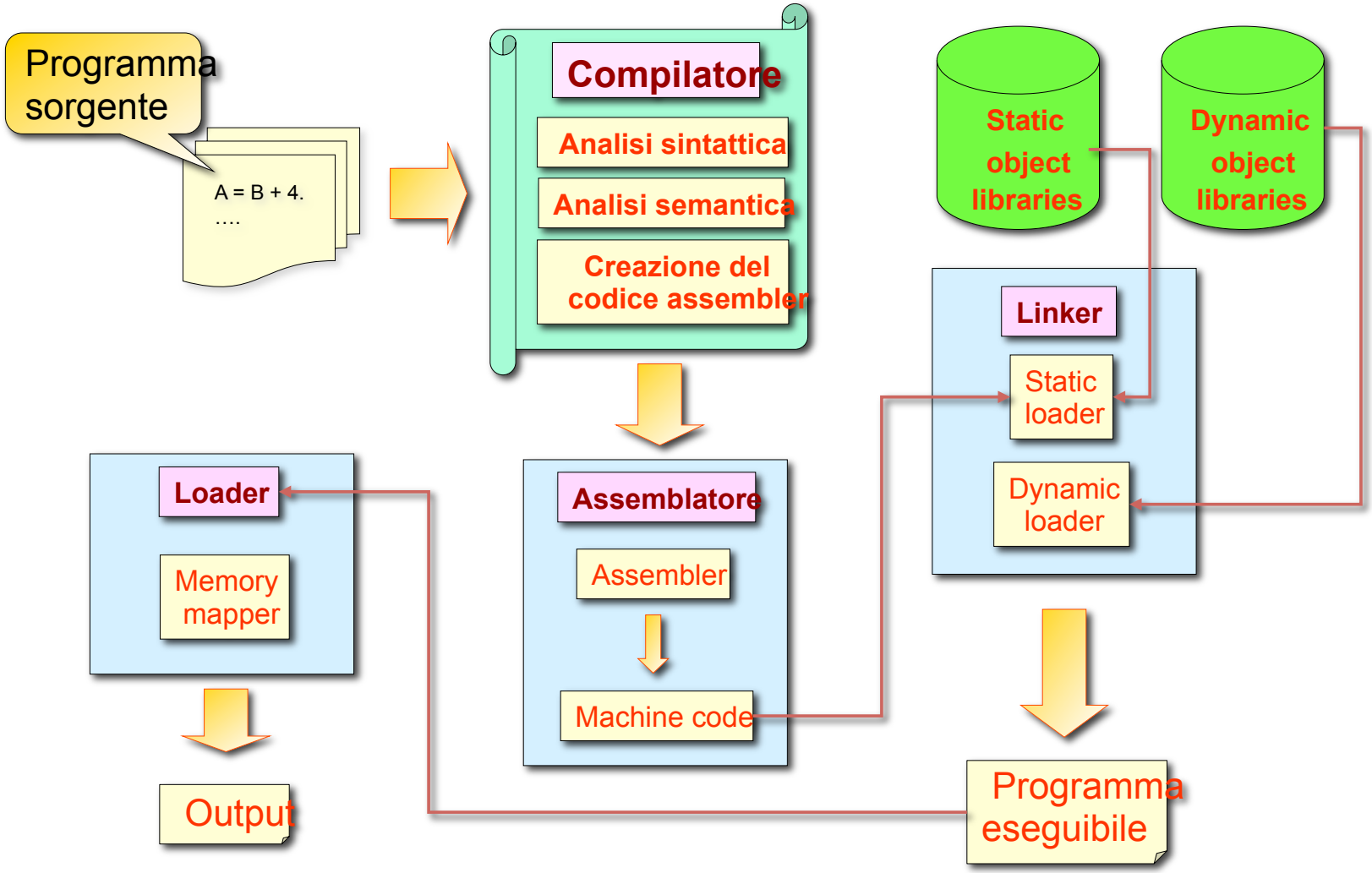
- Un *programma* e' una sequenza opportuna di istruzioni che devono essere eseguite in ordine per completare una determinata operazione.
- Il programma risiede in memoria ed ogni istruzione per essere correttamente eseguita e sincronizzata deve avere un formato noto al processore e un protocollo di lettura definito.
- Ogni istruzione deve essere poi decodificata dalla CPU prima di essere eseguita
- L'esecuzione di un programma e' una iterazione (dalla prima all'ultima istruzione) del cosiddetto ciclo di fetch-execute
  1. Preleva (Fetch) la prossima istruzione da eseguire dalla memoria
  2. Decodifica l'istruzione da eseguire (detta OPCODE)
  3. Legge gli eventuali operandi dalla memoria
  4. Esegue (Execute) le istruzioni ed immagazzina i risultati

Uno sguardo alla gerarchia del codice

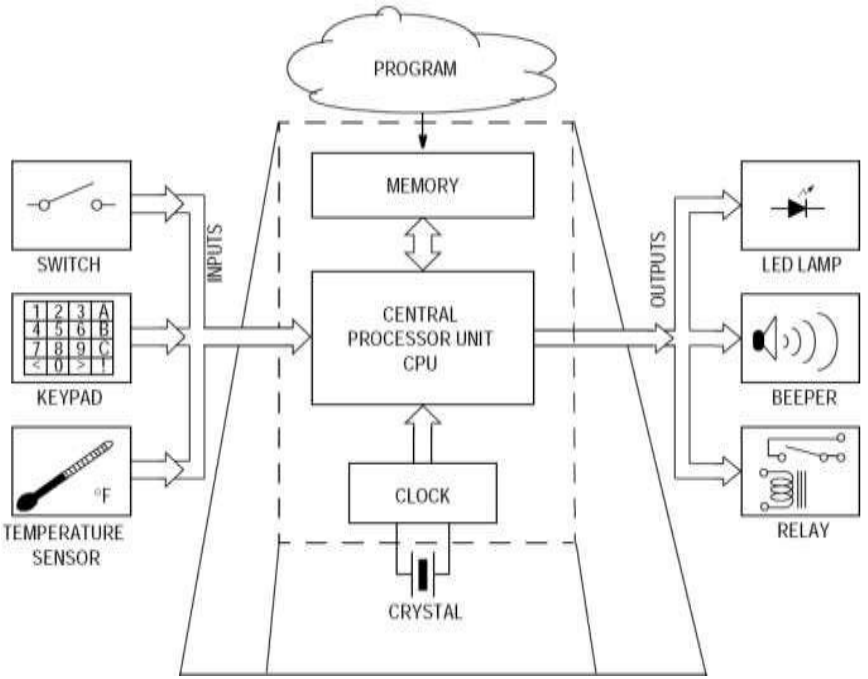
- Linguaggio di alto livello
  - "User friendly and intelligible"
  - Assicura produttività e (a volte...) portabilita' tra diverse piattaforme
  - Strumenti software (Compilatore) traducono in assembler (o anche codice eseguibile)
- Linguaggio Assembler
  - Rappresentazione testuale, mnemonica delle istruzioni di un computer
  - Strumenti SW (*Assembler*) traducono "assembly code" nel linguaggio dell'Hardware
- Rappresentazione Hardware
  - Linguaggio Macchina dove le istruzioni ed i dati sono rappresentati da stringhe di bit

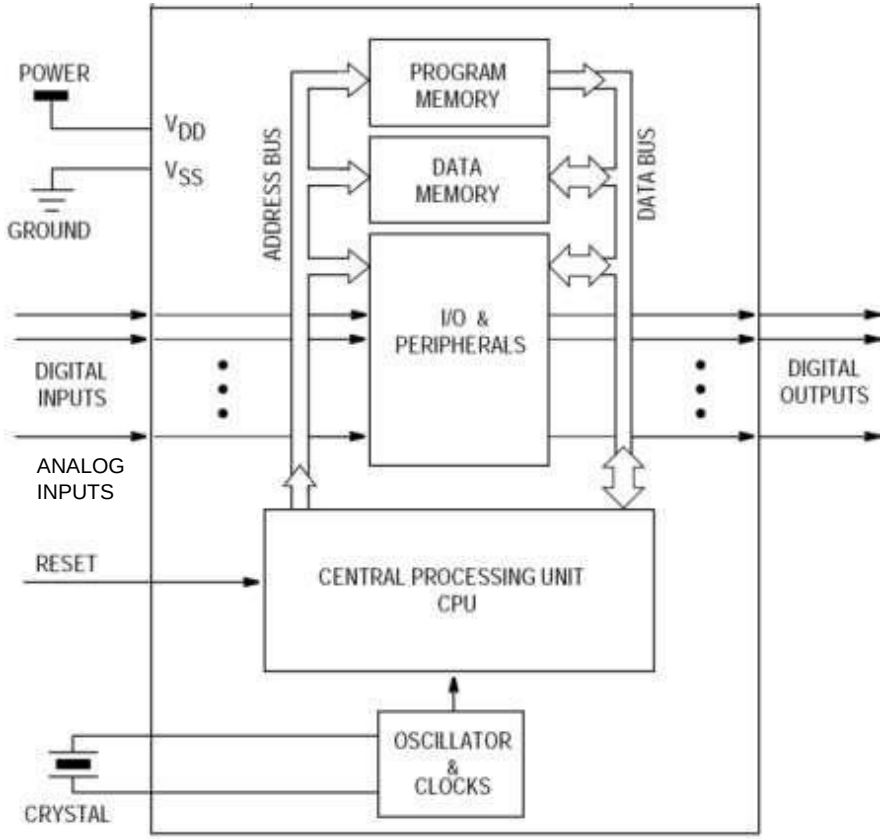


Ciclo completo di produzione di un programma eseguibile



Microcontrollore



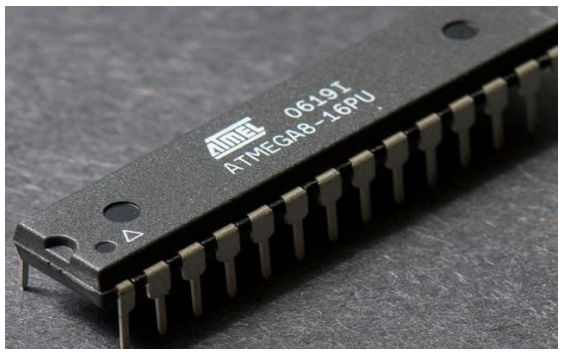


In elettronica digitale il microcontrollore o microcontroller o MCU (MicroController Unit) è un **dispositivo elettronico integrato** su singolo chip, nato come evoluzione alternativa al Microprocessore ed utilizzato generalmente in sistemi embedded ovvero per applicazioni specifiche di controllo digitale.

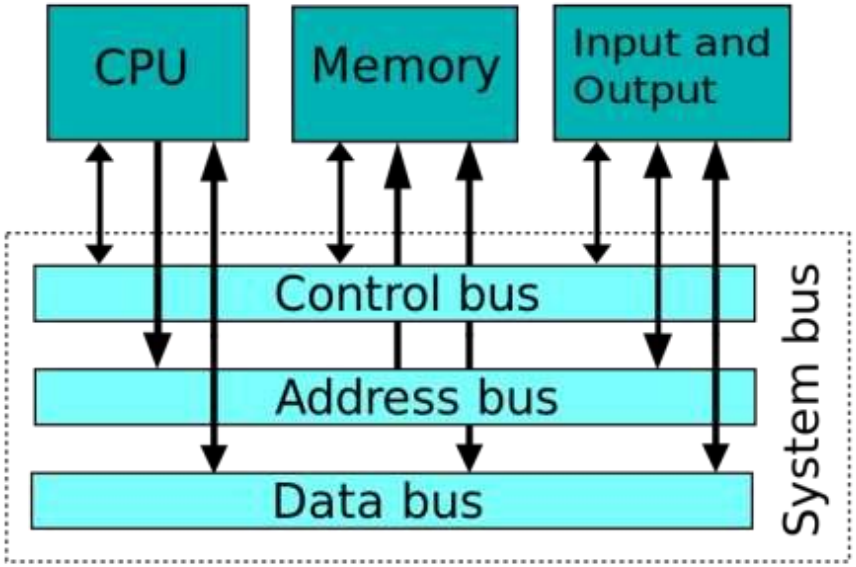
http://it.wikipedia.org/wiki/Microcontrollore

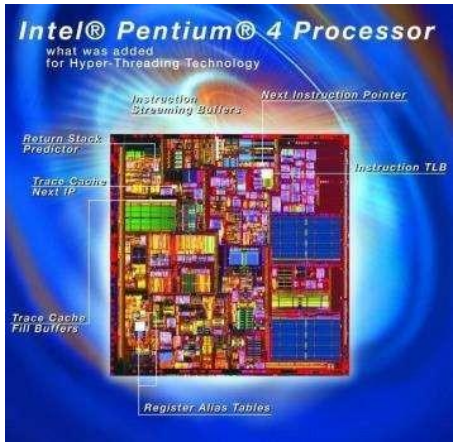
AVR (**A**lf and **V**egard **R**ISC processor)

- Sviluppati da Atmel nel 1996
- Famiglia di Microcontrollori RISC (**reduced instruction set computer**)
  - Istruzioni a lunghezza fissa, accesso alla memoria di tipo load-store con 32 registri general-purpose
  - Pipeline a due stadi per velocizzare l'esecuzione
  - Esecuzione della maggior parte delle istruzioni in un solo ciclo di clock
  - Fino a 12 volte più veloce di una architettura standard CISC
- Architettura Harvard



Architettura Von Neumann



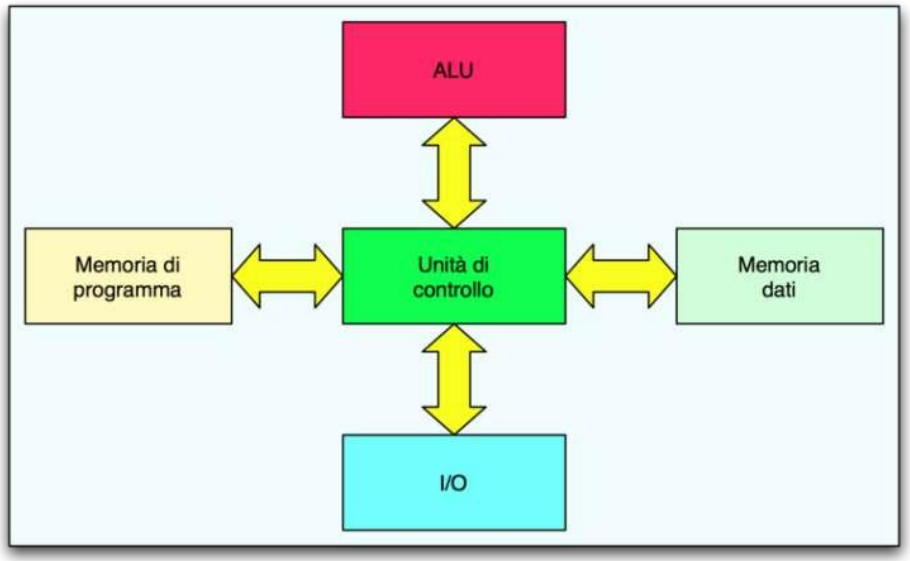


Nella **macchina di von Neumann**:

- dati e istruzioni memorizzati in un'unica memoria che permette lettura e scrittura;
- la memoria è costituita da celle uguali, indirizzate dalla loro posizione;
- le istruzioni vengono eseguite in modo sequenziale.

http://it.wikipedia.org/wiki/Architettura_di_von_Neumann

Architettura Harvard



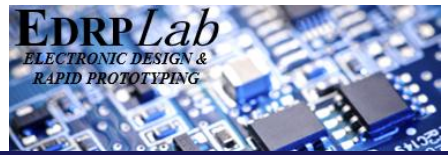
Surveyor SRV-1 Blackfin Robot

- L'architettura di **Von Neumann** si contrappone all'**architettura Harvard** nella quale invece i dati del programma e le istruzioni del programma sono memorizzati in spazi di memoria distinti.
- Nello specifico i microcontrollori Atmel AVR utilizzano una **memoria flash** per la memorizzazione del programma, mentre una **memoria SRAM** (*Static RAM*) per la memorizzazione dei dati

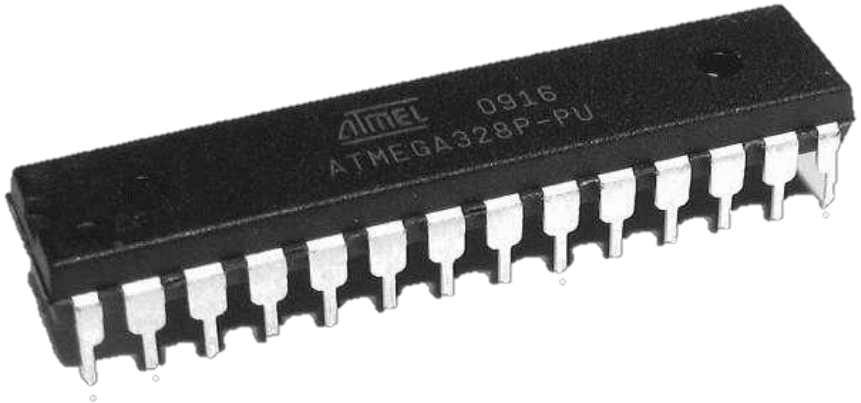
http://it.wikipedia.org/wiki/Architettura_Harvard

AVR Microcontroller (caratteristiche)

- Memoria Flash programmabile (almeno 10.000 volte) , RAM, EEPROM interne (scrivibile almeno 100.000 volte)
- Sistema di programmazione interno (ISP)
- Varietà di periferiche: I/O digitali, ADC, Timer, UART, RTC timer, pulse width modulator (PWM)…
- Clock fino a 20MHz
- Ampia gamma di tensioni di funzionamento: da 1.8 V a 6.0 V.
- Package variabile da 8 pin fino a 64 pin
- *Watchdog* con oscillatore interno autonomo
- POR (Power On Reset)
- Famiglie
  - ATtiny25-45-85, ATtiny24-44-84, ATtiny2313-4313 ...
  - ATmega88, ATmega168, ATmega328P ...
  - XMega (sigle che iniziano con "ATXMega")

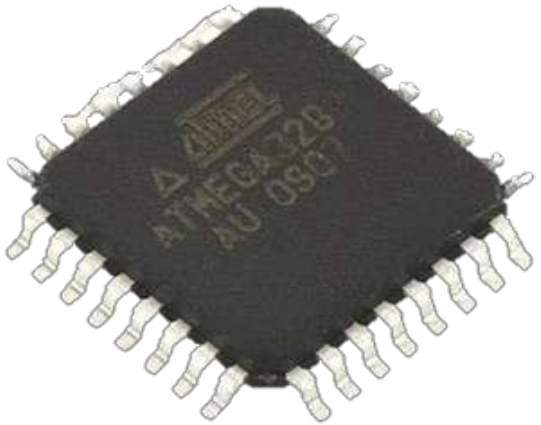


Atmel ATmega328P



Versione PDIP
Plastic Dual In-line Package

Versione SMD
Surface-Mount Device



Valori massimi

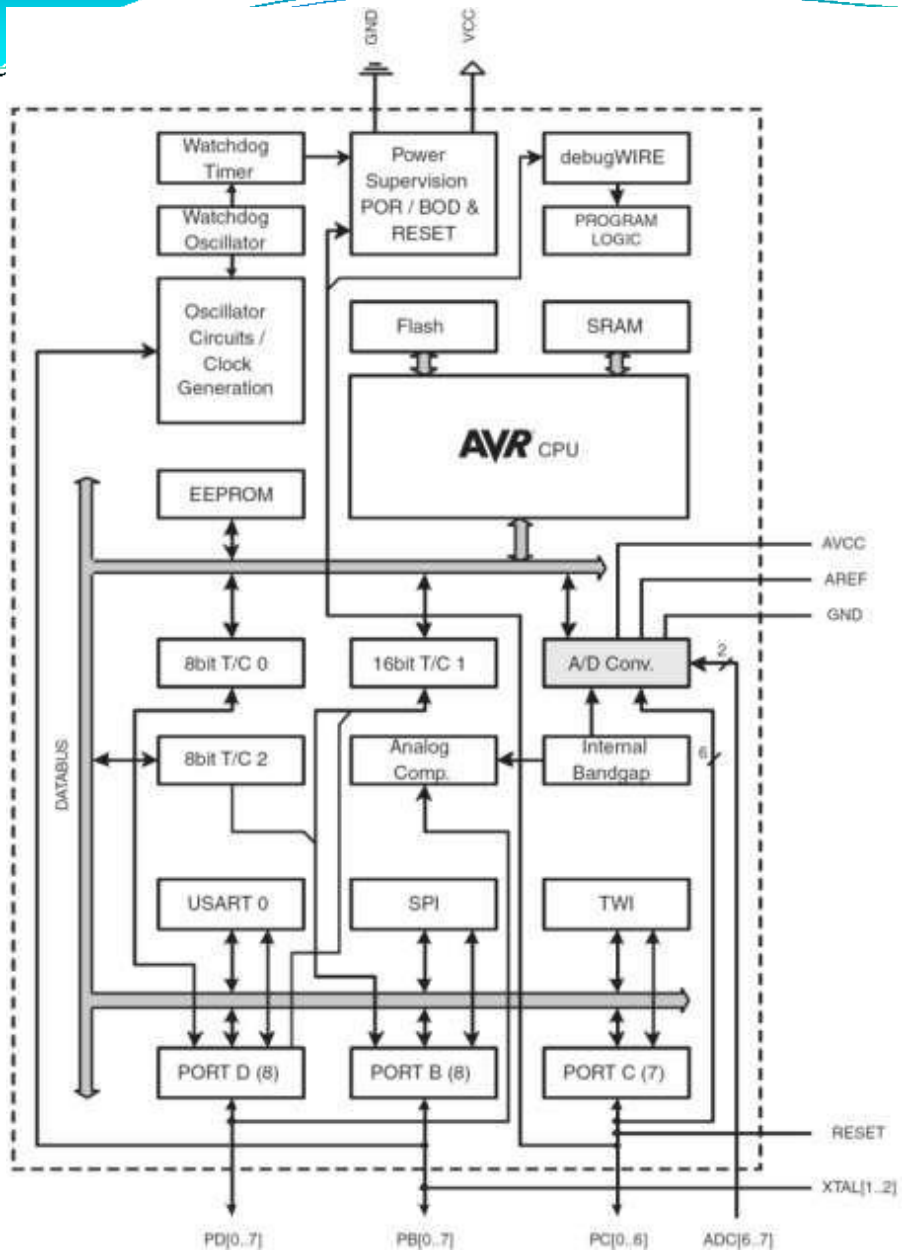
28.1 Absolute Maximum Ratings*

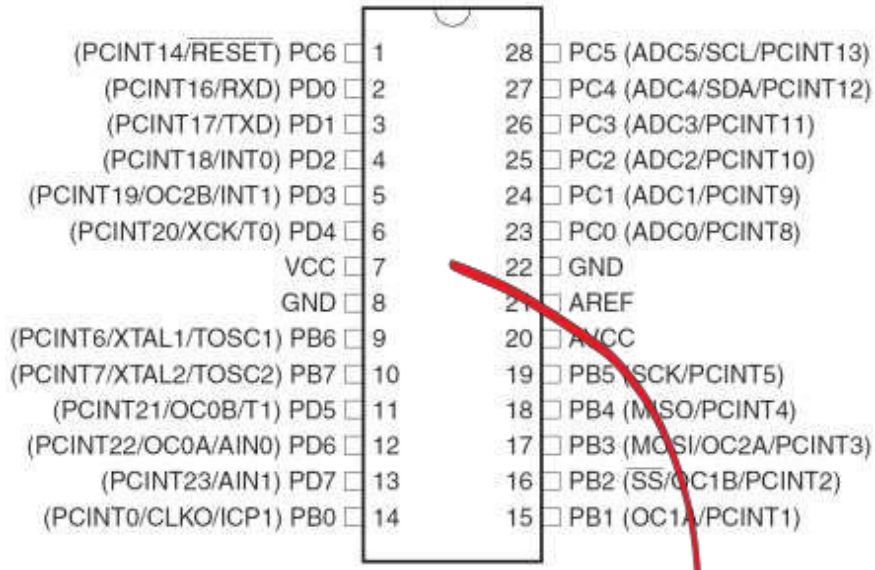
*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf

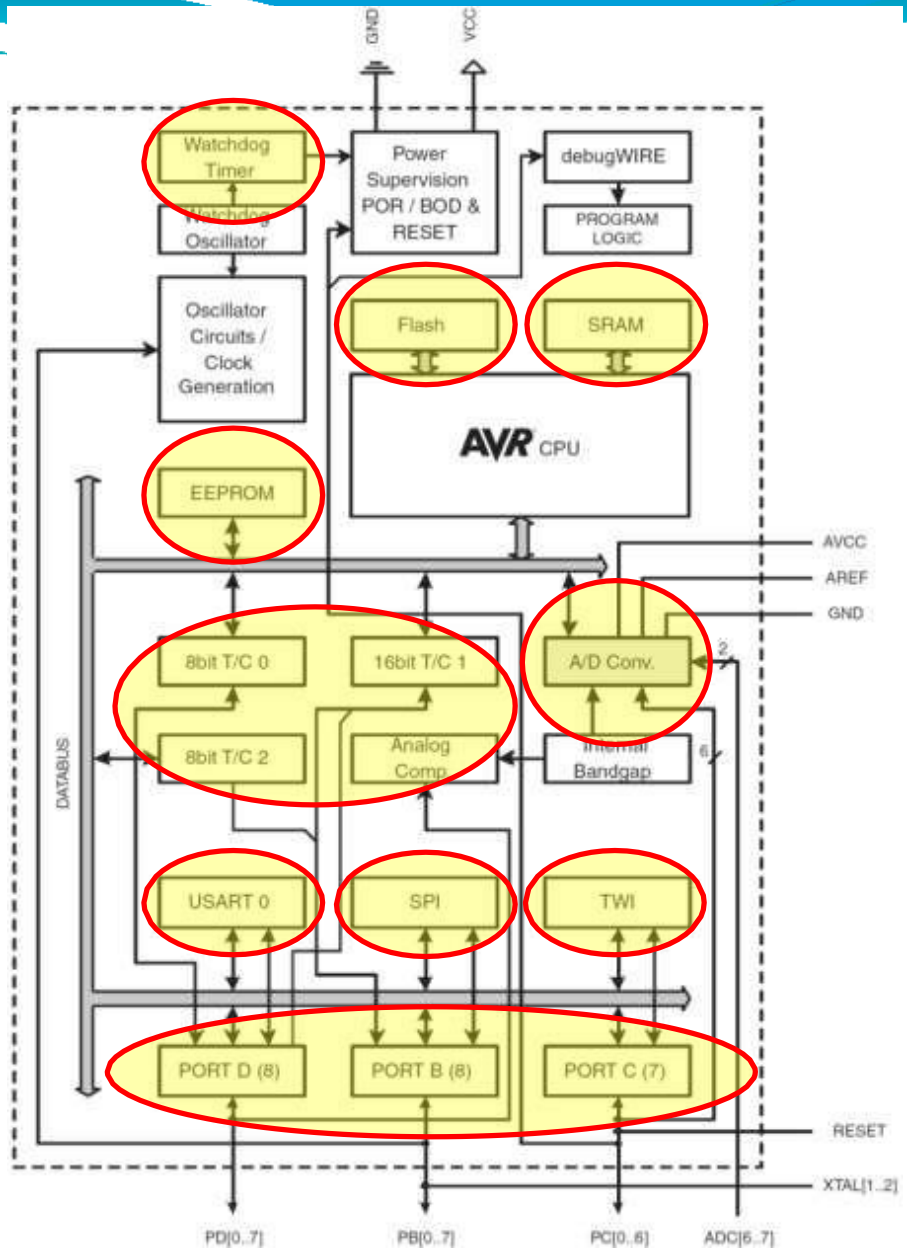
Schema a blocchi - Architettura interna ATmega328





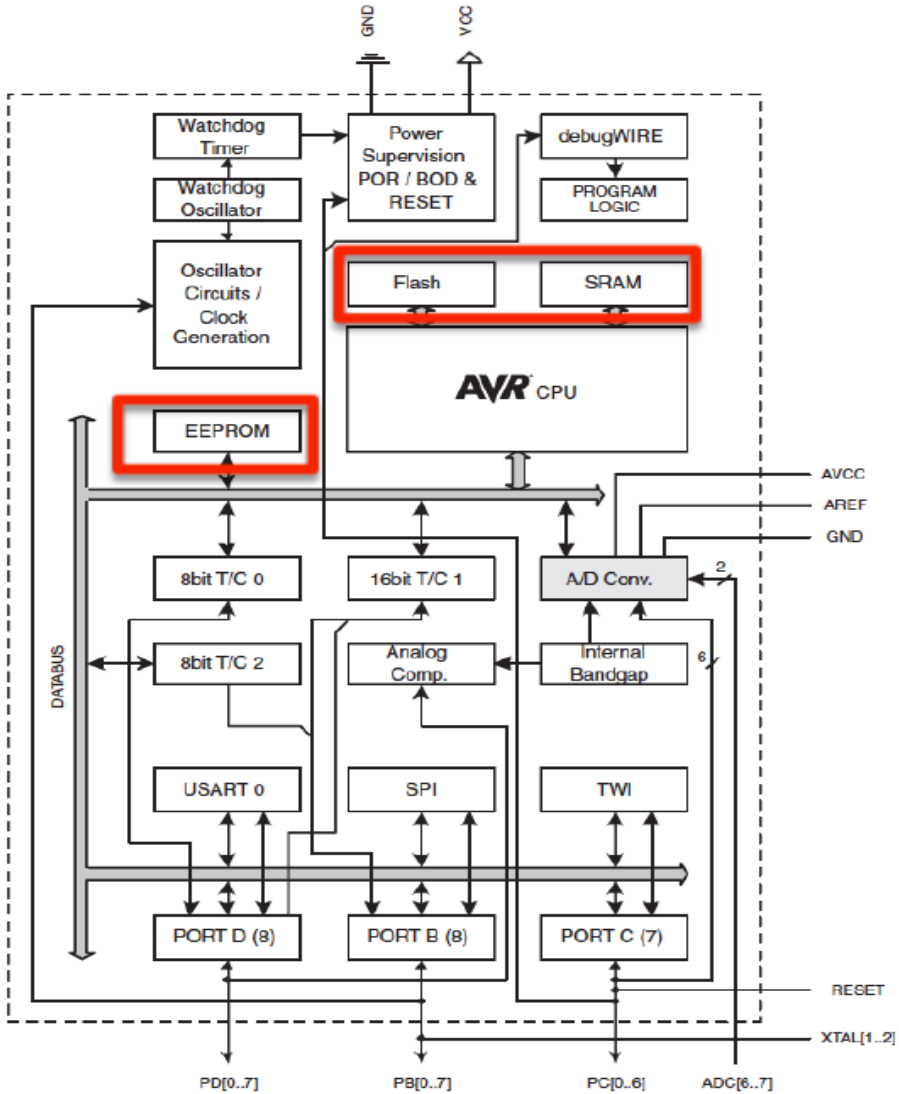
Periferiche interne

- Memoria Flash (memoria Programma)
- Memoria SRAM (memoria dati)
- Memoria EEPROM (memoria dati)
- WatchDog Timer
- Interfaccia Seriale
- Interfaccia SPI
- Interfaccia I2C
- Convertitore Analogico –Digitale
- Timers
- Porte/Pin



Memorie integrate

- Architettura Harvard
- Flash-program memory
  - 32kB
- SRAM-data memory
  - 2KB
- EEPROM
  - Per dati da conservare anche in caso di rimozione dell'alimentazione.
  - Presenza su I/O bus

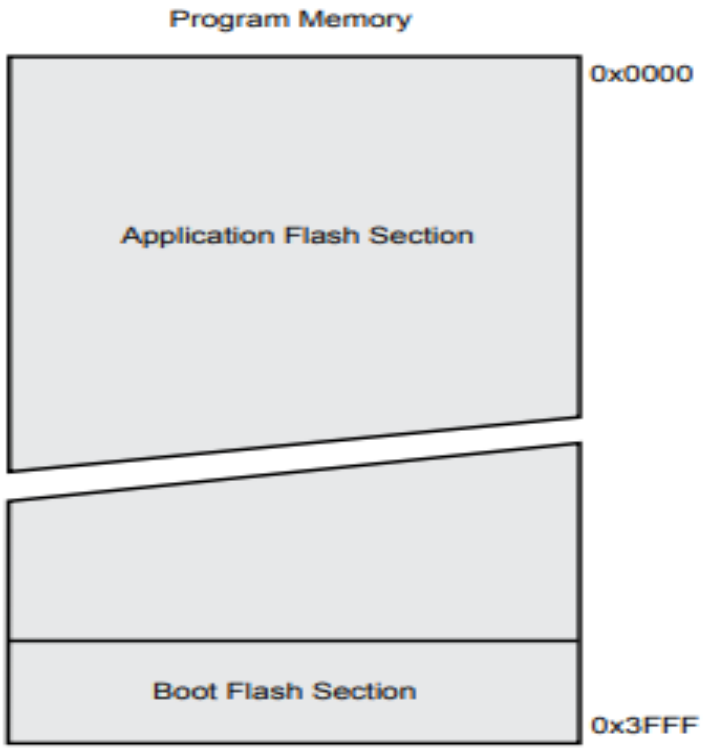


Memorie integrate

- **Flash memory (32KB) (indirizzi da 15bit)**
  - **Utilizzata per la memoria programma-sola lettura**
  - **Non volatile**
  - **Alloca i dati nella flash utilizzando l l'attributo PROGMEM**
- **SRAM (2KB)**
  - **Valori temporanei, stack,etc…**
  - **Memoria volatile**
  - **Dimensione limitata**
- **EEPROM (1KB)**
  - **Dati a lungo termine**

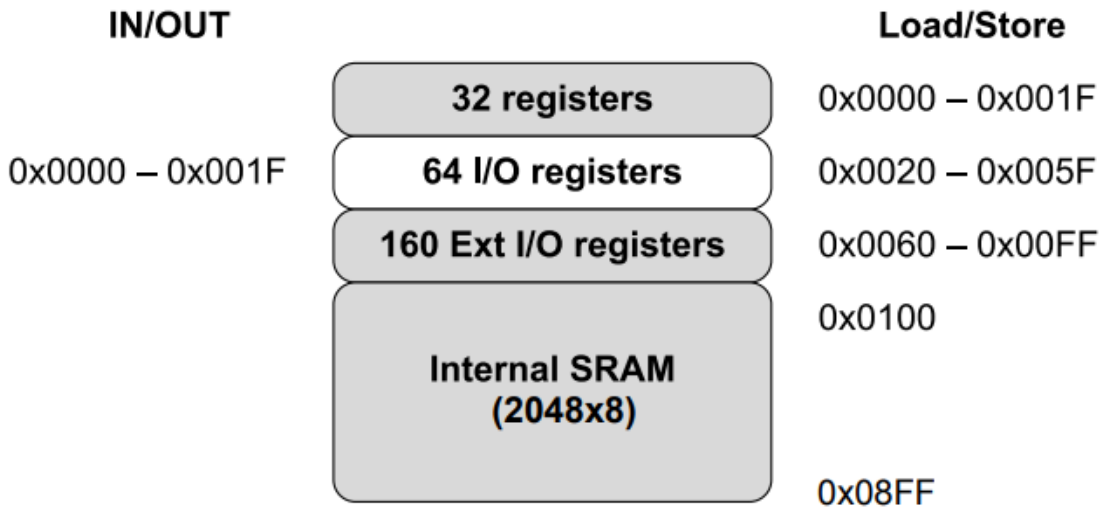
FLASH PROGRAM MEMORY

Visto che tutte le istruzioni AVR sono a 16bit o 32bit, la memoria flash per la memorizzazione del programma è organizzato in 16K locazioni x 16 bit. Per motivi di sicurezza del software la memora flash è divisa in due sezioni: la sezione destinata al **bootloader** e la sezione destinata alla memoria del programma nel dispositivo. Il **program counter (PC)** del ATmega328/P è largo 14bits, in modo da indirizzare le 16K locazioni di memoria programma.



SRAM Data Memory

Le 2303 locazioni di memoria dati più basse indirizzano sia i Register File, la memoria I/O, la memoria I/O estesa e i dati della SRAM. Le prime 32 locazioni indirizzano i Register File, le successive 64 locazioni indirizzano la memoria I/O , dopo 160 locazioni della memoria I/O estesa, e i successivi 2K locazioni indirizzano i dati della SRAM.



Ci sono 5 metodi di indirizzamento:

- **Diretto;**
- **Indiretto con spostamento;**
- **Indiretto;**
- **Indiretto con pre-incremento;**
- **Indiretto con post-incremento**

SRAM Data Memory

General Purpose Register

- 32 8-bit GP registers
- Part of SRAM memory space

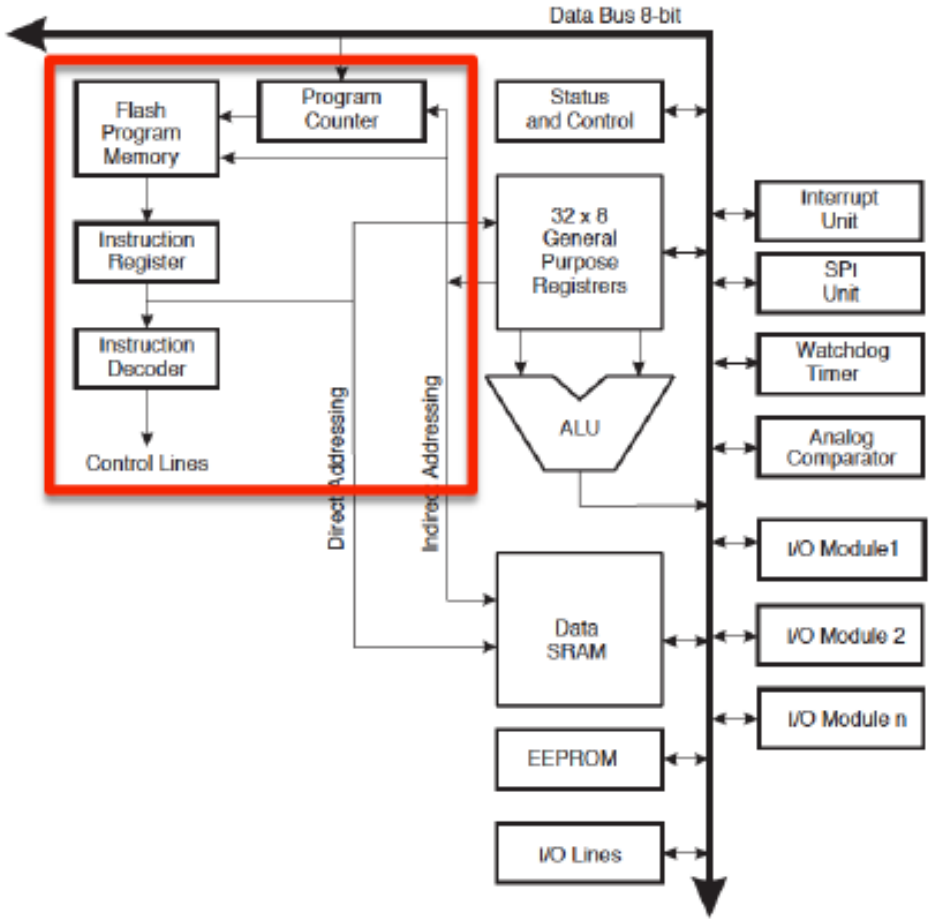
Figure 6-2. AVR CPU General Purpose Working Registers

	7 … 0	Addr.	
General Purpose Working Registers	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
	R14	0x0E	
	R15	0x0F	
	R16	0x10	
	R17	0x11	
	…		
	R26	0x1A	X-register Low Byte
	R27	0x1B	X-register High Byte
	R28	0x1C	Y-register Low Byte
	R29	0x1D	Y-register High Byte
	R30	0x1E	Z-register Low Byte
	R31	0x1F	Z-register High Byte

AVR CPU

- Instruction Fetch and Decode

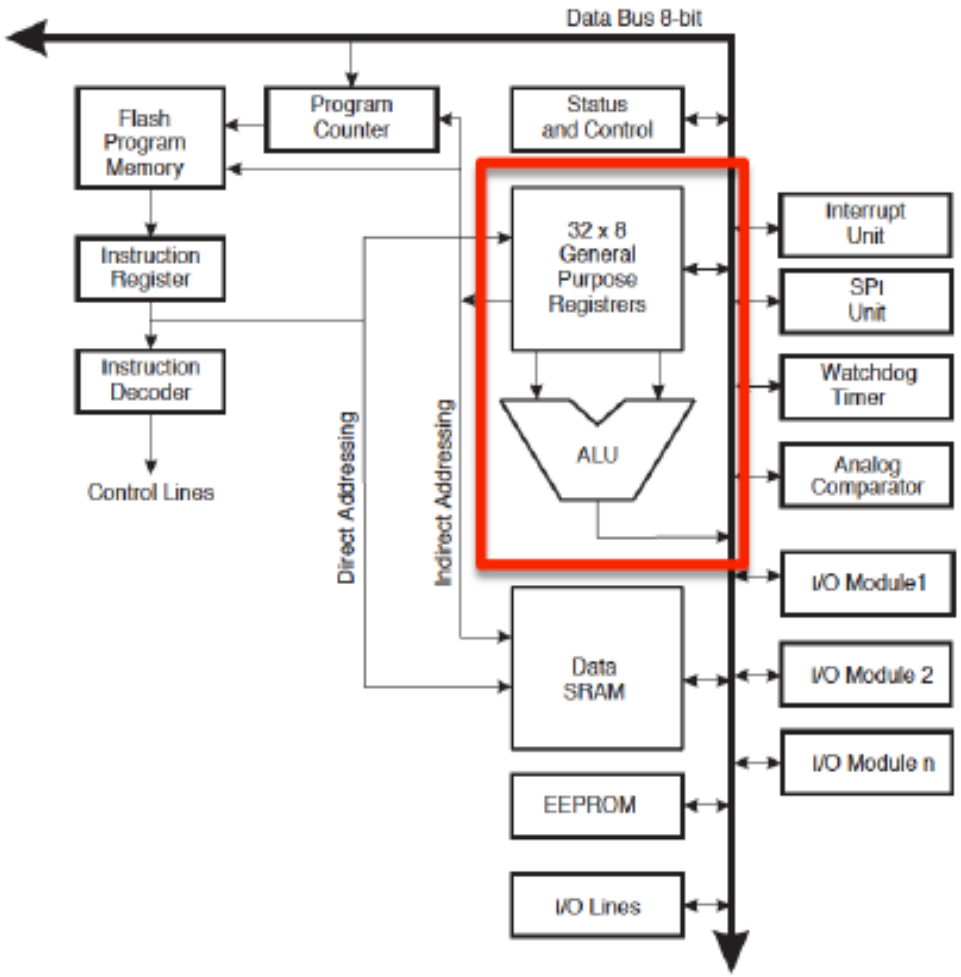
- **Caricamento istruzione dalla memoria programma (Fetch)**
- **Caricamento dati dalla memoria dalla memoria dai registri.**
- **Decodifica dell'istruzione (Decode)**



AVR CPU

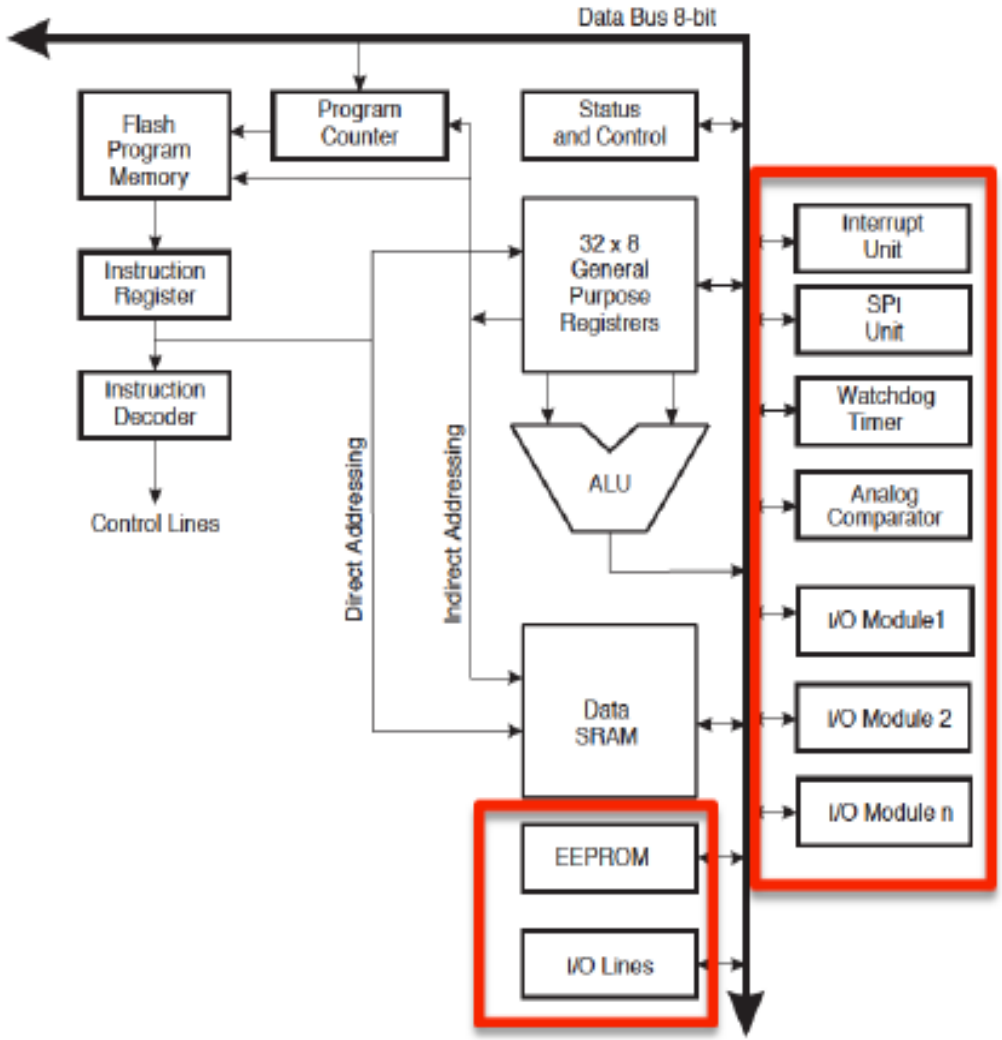
- ALU Instructions

- **Esecuzione istruzione (Execute)**



AVR CPU

- I/O and special functions



ATMega328: Timing

Figure 6-4. The Parallel Instruction Fetches and Instruction Executions

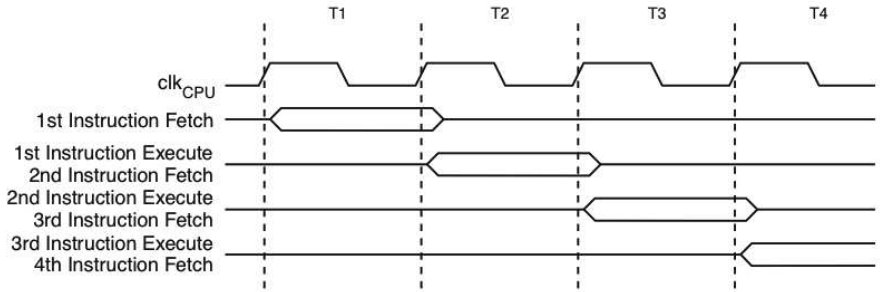
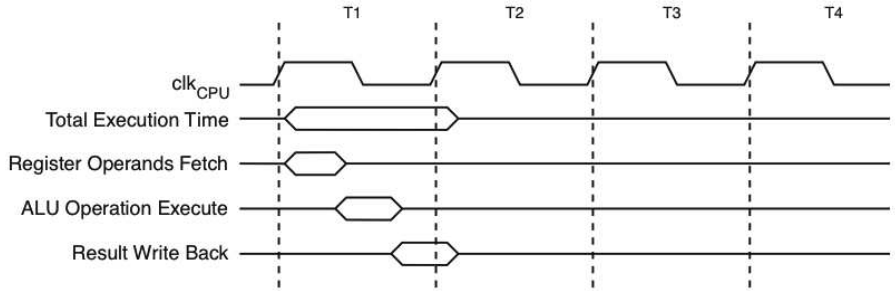


Figure 6-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 6-5. Single Cycle ALU Operation



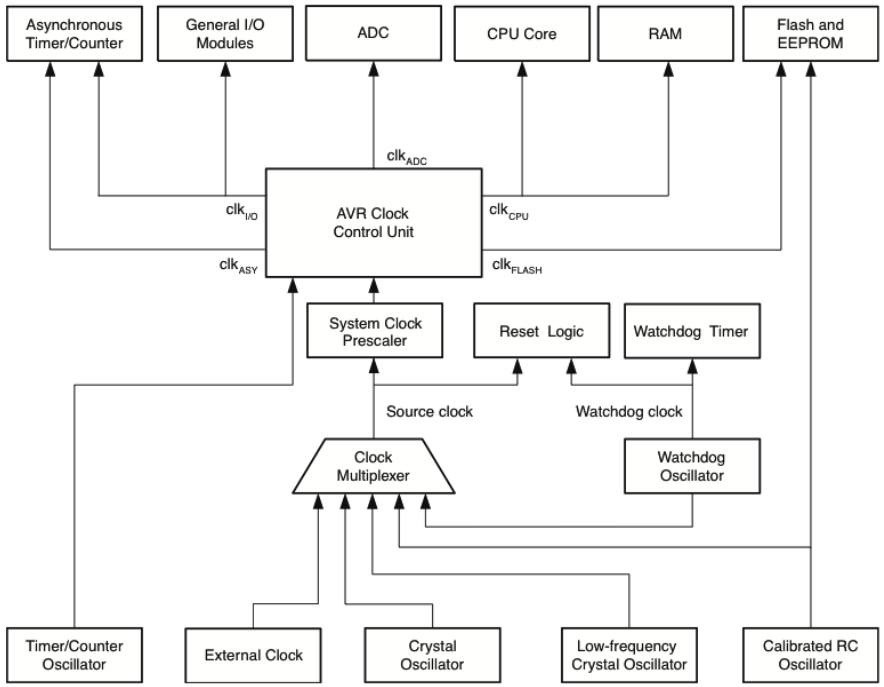
ATMega328: Interrupts

Ci sono varie possibili sorgenti di interrupts, interne o esterne e per ciascuna di esse un interrupt vector. Gli interrupt vectors sono contenuti nella memoria di programma e ordinati in base alla priorita'. Il Reset e' l'interrupt con priorita' piu' alta. Per ogni interrupt e' previsto 1 bit individuale di abilitazione, ma nello Status Register vi e' un bit (Global interrupt enable) di abilitazione globale. Quando arriva un interrupt il Global interrupt enable viene disabilitato e si avvia l'esecuzione della routine di servizio corrispondente. All'uscita di questa il Global bit viene nuovamente abilitato. Ci sono sostanzialmente due tipi di interrupt: nel primo tipo, l'evento viene memorizzato in un bit (interrupt flag) e viene servito appena possibile. Ovvero, se una o piu' condizioni di interrupts avvengono quando il Global enable e' disabilitato, non vengono perse e vengono servite non appena il Global enable e' abilitato di nuovo. Nel secondo tipo invece l'interrupt verra' servito solo se la condizione che lo ha causato persiste quando il Global enable e' abilitato di nuovo. Quando l'AVR esce da un interrupt ritorna sempre al programma principale ed esegue una istruzione prima di occuparsi di eventuali altri interrupt in attesa.

ATMega328: Clock

Sistema di gestione e distribuzione del clock:

Figure 8-1. Clock Distribution



Sono possibili diverse sorgenti di clock:

- interno (8 *MHz*, ma modificabile);
- esterno ($0 \div 20$ *MHz*), di vario tipo.

ATMega328: Alimentazione

- vari *sleep mode* selezionabili da programma per ridurre i consumi;
- *Brown out detector*: invia un segnale di reset quando la tensione di alimentazione scende sotto un livello prefissato;
- *Power on reset*: invia il segnale di reset finche' la tensione di alimentazione e' al di sotto di una certa soglia.

ATMega328: Watchdog

Il *watchdog* e' un mecccanismo di sicurezza che puo' essere utilizzato per evitare che il microcontrollore resti bloccato a causa di errori inaspettati nel programma o comunque altre cause. il sistema e' composto da un oscillatore a 128 *kHz* e un contatore con varie uscite. Sulla base di questo si puo' costruire una logica di *time-out* che puo' essere utilizzata per inviare un segnale di reset al microcontrollore.

ATMega328: Reset

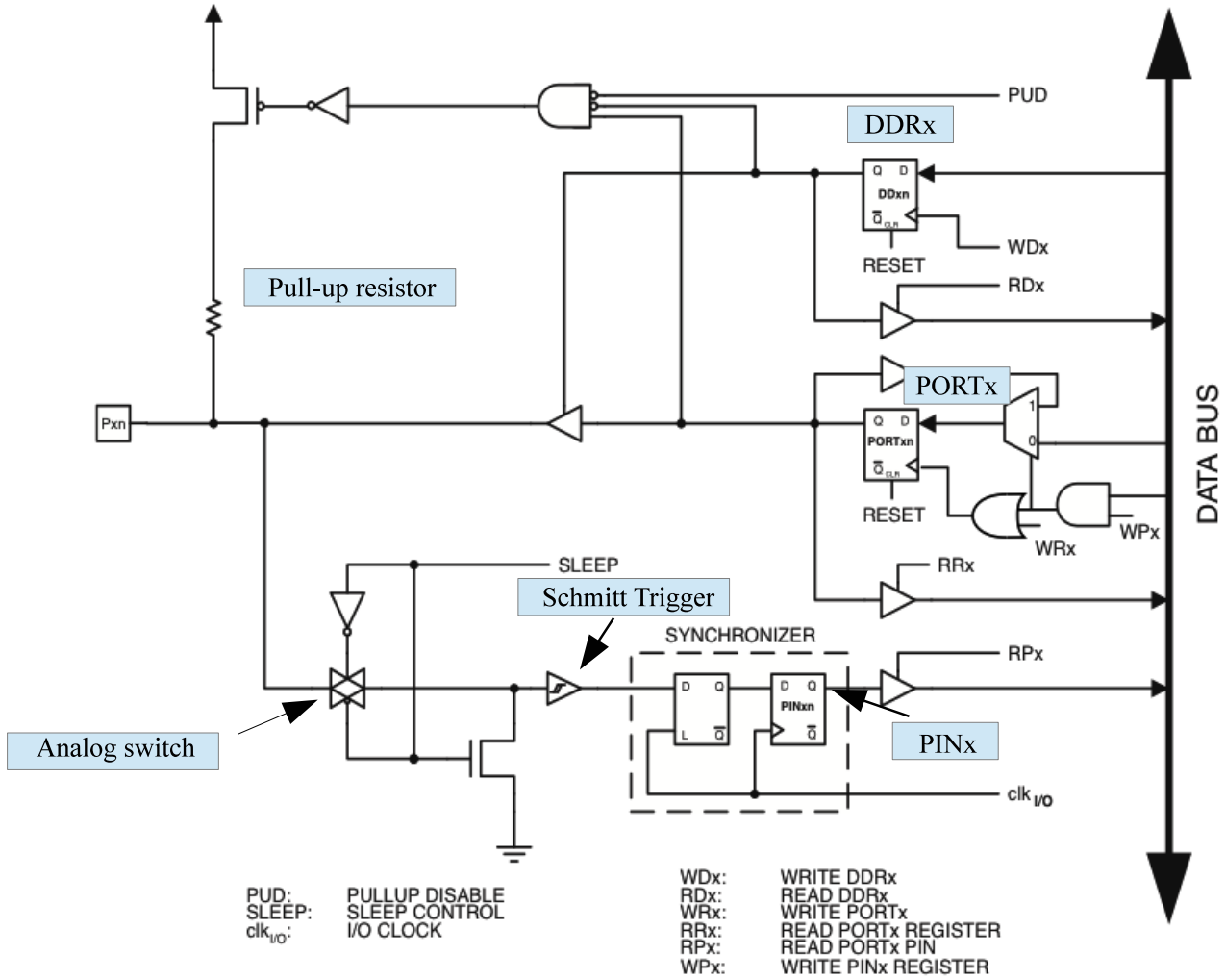
Ci sono quindi varie sorgenti che provocano il reset del microcontrollore:

- Un livello basso sul pin di reset;
- Power-on Reset;
- Brown-out-Reset;
- Watchdog Reset

ATMega328: Timers

Il microcontrollore comprende 2 timers a 8 bit e 1 timer a 16 bit che possono essere utilizzati in vari modi dal programma. Utilizzano un clock interno ma possono anche essere collegati ad un clock esterno.

ATMega328: Input-output digitale



ATMega328: Input-output digitale

Tutte le porte hanno le seguenti caratteristiche:

- resistore di pull-up selezionabile;
- logica 3-state;
- trigger di Schmitt in ingresso;
- sincronismo con il clock di sistema;
- in uscita possono erogare o assorbire corrente.

Tutte le porte degli AVR hanno una funzione di *true Read-Modify-Write* quando vengono usate come I/O digitali. Questo significa che la direzione di un singolo pin puo' essere cambiata al volo senza interferire con gli altri pin.

ATMega328: Input-output digitale

Il controllo di ogni porta e' effettuato con 3 registri:

- DDRx: Data Direction Register;
- PORTx: Output Register;
- PINx: Input Register

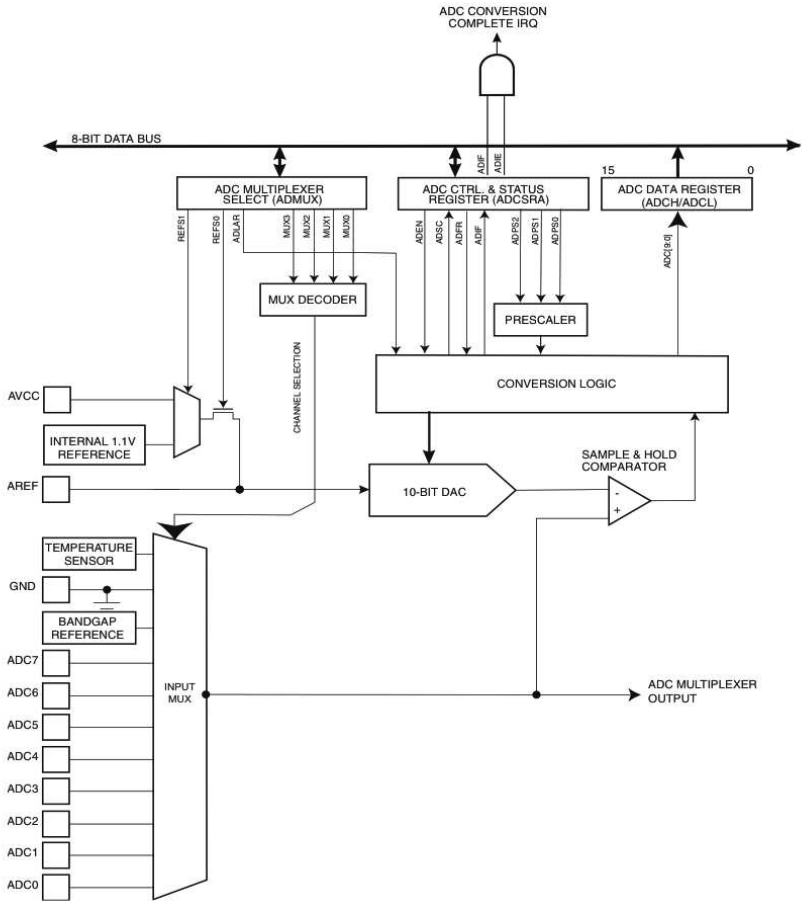
Ad esempio, PORTB3 contiene il bit in uscita per il pin 3 del Port B.

DDx	PORTx	PUD	I/O	Pull-up	Note
0	0	x	Input	No	Alta impedenza
0	1	0	Input	SI	
0	1	1	Input	No	Alta Impedenza
1	0	x	Output	No	
1	1	x	Output	No	

ATMega328: Uscite analogiche

Questa famiglia di microcontrollori non dispone di uscite analogiche. Tuttavia, l'uscita di alcuni pin digitali puo' essere modulata, ovvero utilizzata per emettere un'onda rettangolare (fra 0 e 5 V) con duty cycle variabile. Con un semplice integratore si puo' quindi avere una tensione *quasi* analogica.

ATMega328: Ingressi analogici



ATMega328: Ingressi analogici

Un ADC a 10 bit (sample and hold, successive approximation). Il tempo di conversione e'compreso tra 13 e 260 μs.
E' collegato tramite un multiplexer a 8 ingressi (solo 6 effettivamente utilizzabili, collegati ai pin $PC0 \div PC5$). L'ADC ha un'alimentazione separata (pin AV_{CC}) per migliore immunita' ai disturbi. L'uscita digitale e':

$$\text{Output} = \frac{2^{10} - 1}{V_{ref}} V_{in}$$

Ci sono 3 possibili opzioni per definire V_{ref}:

- AV_{CC};
- 1.1 V (Riferimento di tensione interno a bandgap);
- A_{ref} (Tensione sul pin A_{ref}).

La gestione dell'ADC e' delegata a 4 registri a 8 bit:

- ADMUX: Contiene le informazioni sulla scelta del canale d'ingresso e della V_{ref};
- ADCSRA: bits di status e controlli;
- ADCL e ADCH: contengono i 10 bit del risultato.

ATMega328: Comparatore analogico

Il comparatore analogico compara i valori in ingresso sull'ingresso positivo $AIN0$ e su quello negativo $AIN1$ (ovvero i pin $PD6$ e $PD7$); quando $AIN0 > AIN1$ l'uscita del comparatore ACO va ad 1. Questo segnale puo' essere usato per generare un Interrupt (sul fronte di salita o di discesa), oppure come trigger per il Timer/Counter 1. E' possibile anche utilizzare, all'ingresso non invertente, la tensione di riferimento interna (1.1 V).
E' anche possibile selezionare uno qualunque dei pin analogici di ingresso ($ADC0 \ldots ADC7$) come ingresso negativo del comparatore (poiche' utilizza lo stesso multiplexer dell'ADC, quest'ultimo deve essere off)

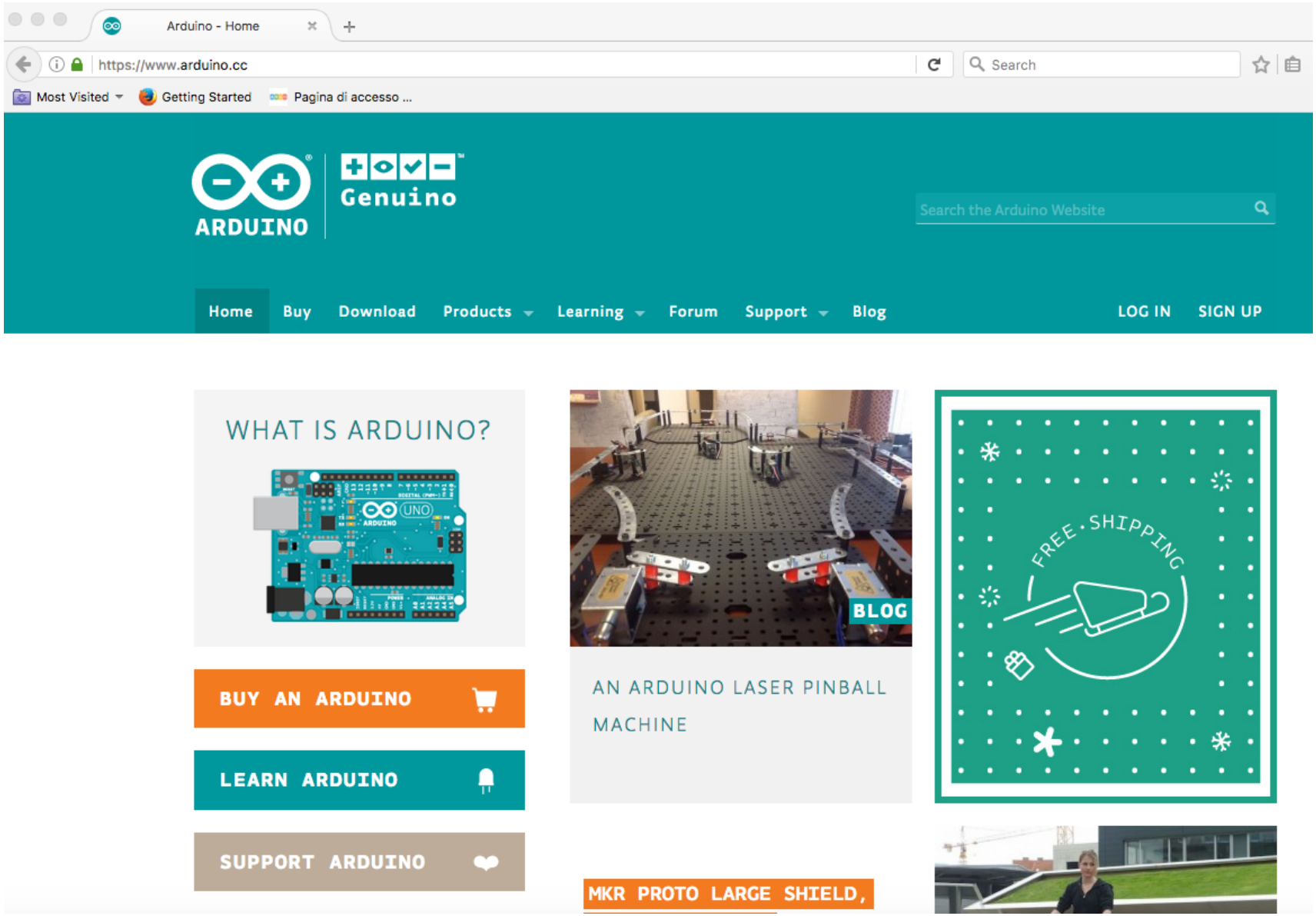
ATMega328: Comunicazione seriale

Il microcontrollore supporta vari meccanismi di comunicazione con dispositivi esterni:

- USART (Universal Synchronous Asynchronous Receive Transmit): Pin *PD*0 e *PD*1;
- SPI (Serial Peripheral Interface): Pin *PB*2, *PB*3, *PB*4, *PB*5;
- I2C/TWI (Inter Integrated Circuit / Two Wire Interface: Pin *PC*4 e *PC*5.

Sono anzitutto necessari per consentire all'utilizzatore di caricare e gestire il programma che deve essere eseguito sul microcontrollore, ma consentono anche di gestire dispositivi esterni (integrati) ovvero di trasferire dati verso l'esterno.
Ne discuteremo in dettaglio piu' avanti.

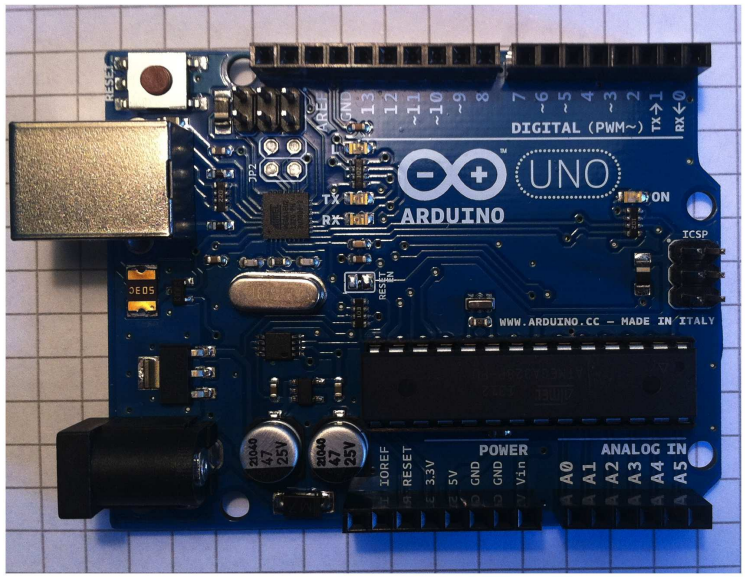
Tutte le info qui –> https://www.arduino.cc



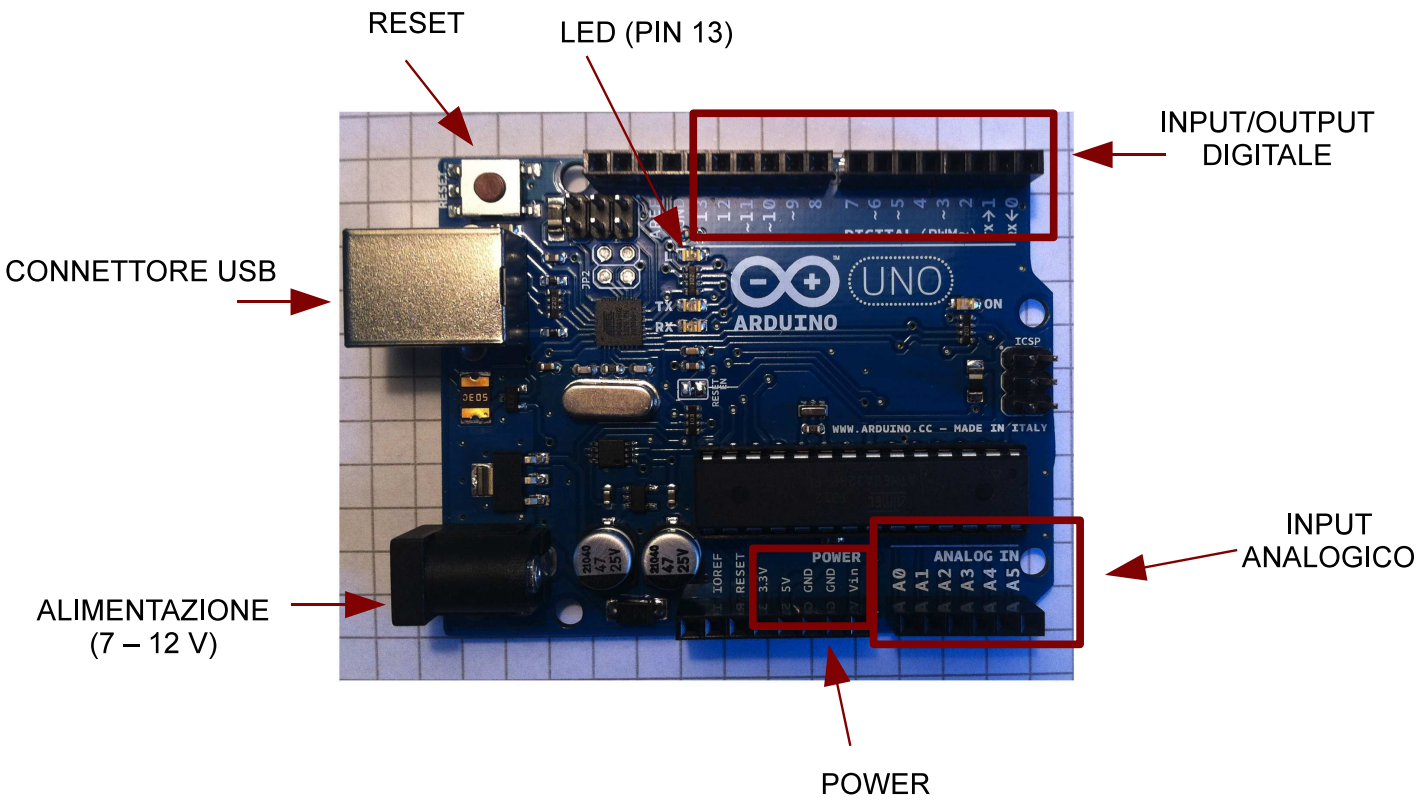
Arduino

E’ un sistema di sviluppo di piccole dimensioni basato su Microcontrollori ATMEL, adatto per sviluppo di prototipi e per scopi didattici.
E’ integrato con un ambiente di sviluppo software (open source) che permette un facile e rapido utilizzo del microcontrollore.
Esistono varie versioni, noi utilizzeremo la scheda Arduino Uno, basata sul microcontrollore ATMEL ATMEGA328.

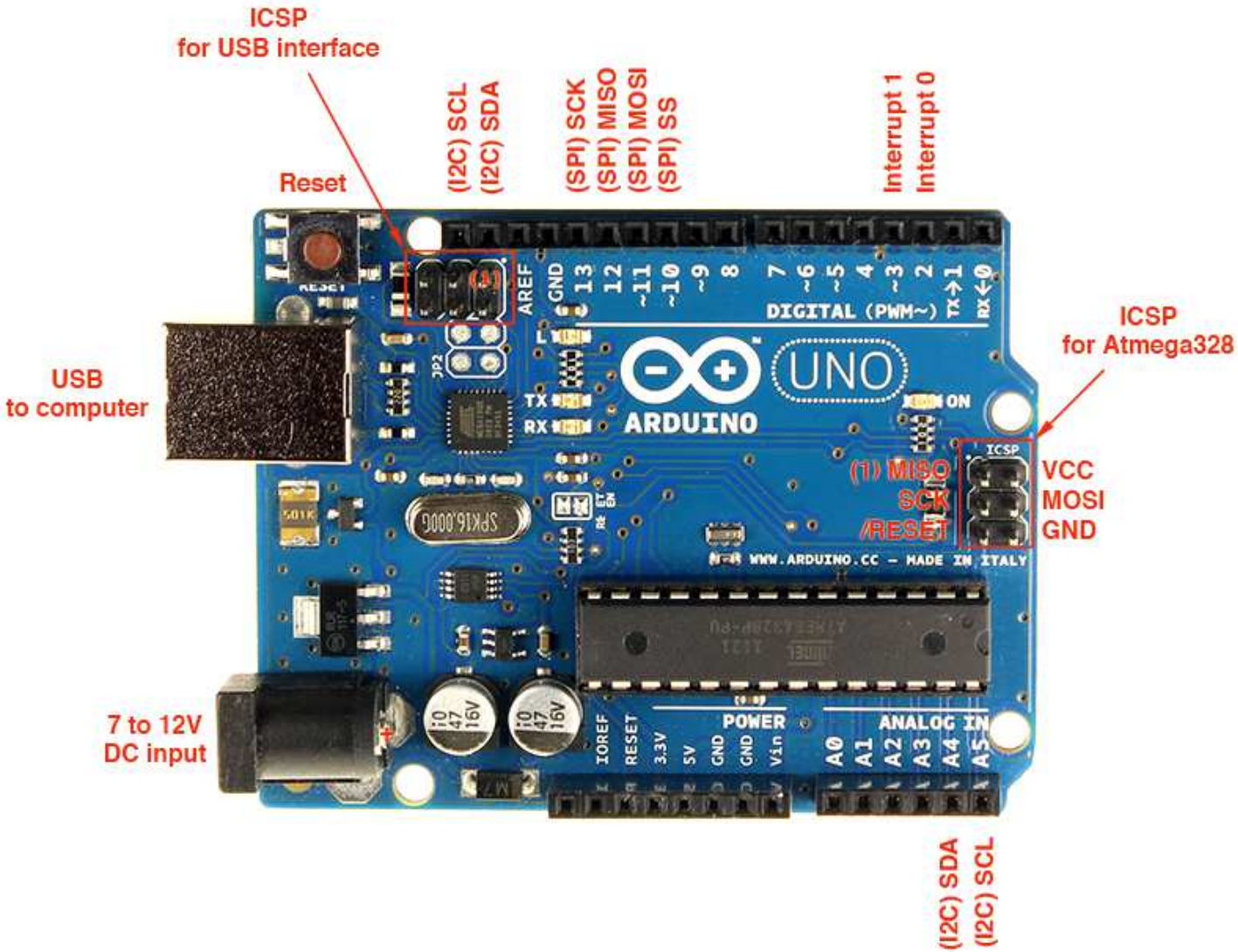
Arduino Uno: la scheda



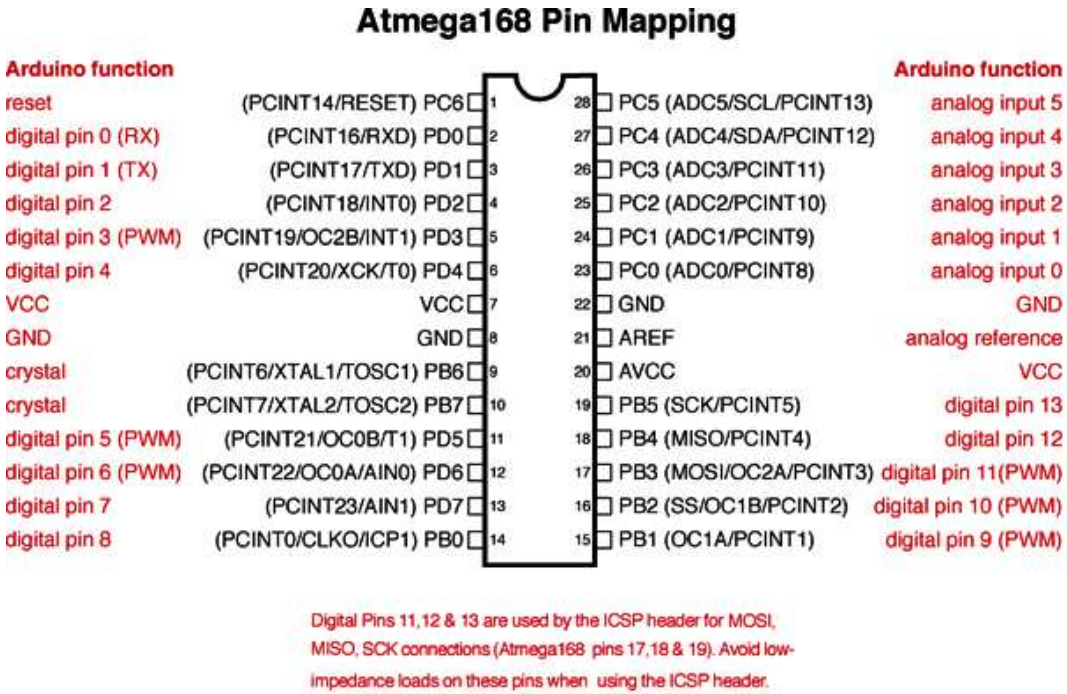
Arduino Uno: la scheda



Arduino Uno: la scheda



Arduino: pin mapping del ATMega328

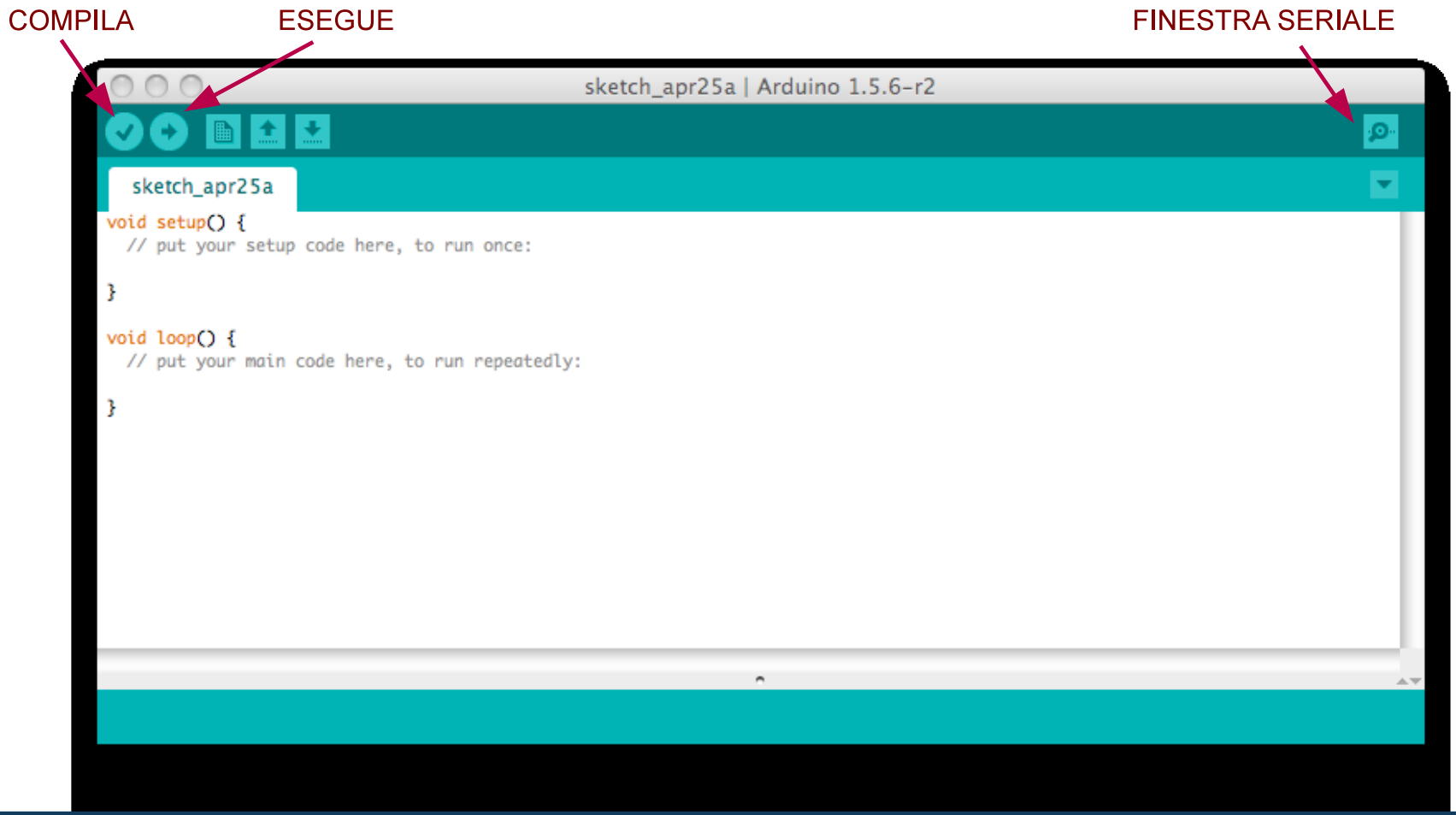


Ambiente di sviluppo

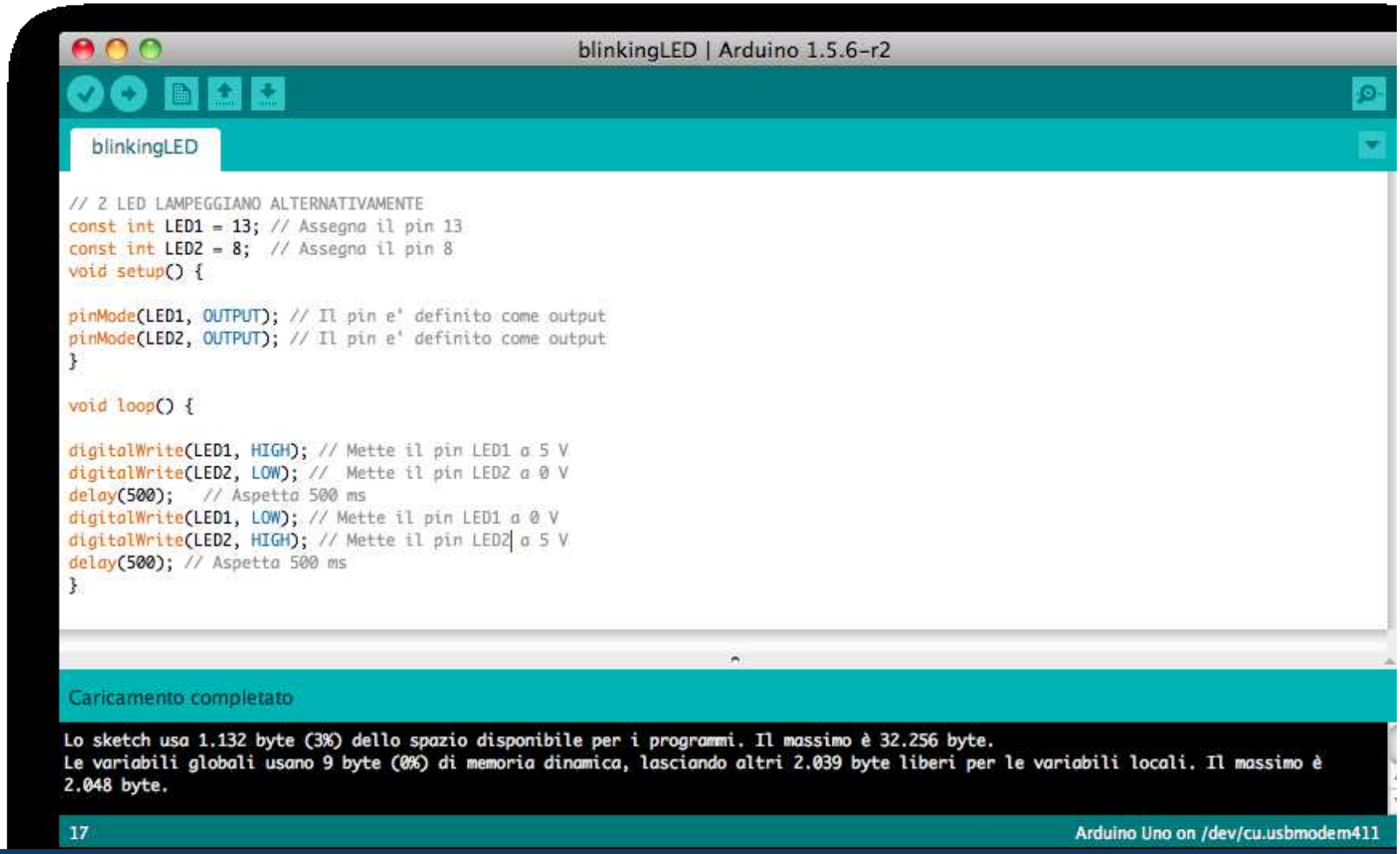
- Arduino dialoga con un computer host (PC Windows, Mac, Linux) tramite la porta USB;
- E' necessario installare sull'host il programma Arduino (open source). E' scritto in Java e quindi in grado di funzionare su molte piattaforme.
- La programmazione del microcontrollore e' in linguaggio C.

La finestra di scrittura del programma

Offre un *framework* per la scrittura dello *sketch* (come e' chiamato nel gergo di Arduino)



Semplice esempio



Semplice esempio (2)

Prima di compilare Arduino trasforma lo sketch creando il programma in *c* completo:

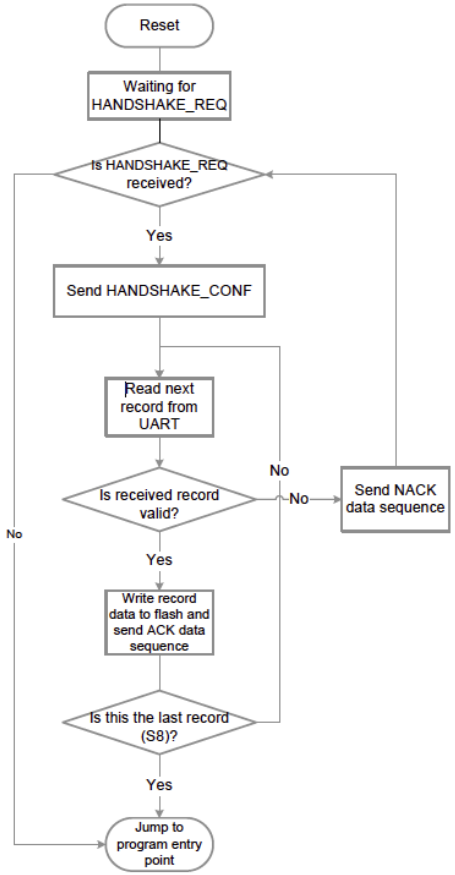
```
#include "Wprogram.h";
void setup();
void loop();
void setup() {
pinMode(LED1, OUTPUT); // Il pin e' definito come output
pinMode(LED2, OUTPUT); // Il pin e' definito come output
}
void loop() {
digitalWrite(LED1, HIGH); // Mette il pin LED1 a 5 V
digitalWrite(LED2, LOW); //  Mette il pin LED2 a 0 V
delay(500);    // Aspetta 500 ms
digitalWrite(LED1, LOW); // Mette il pin LED1 a 0 V
digitalWrite(LED2, HIGH); // Mette il pin LED2 a 5 V
delay(500); // Aspetta 500 ms
}
int main() {
setup();
for(;;) { loop();}
return 0;
}
```

Semplice esempio (3)

Il compilatore (residente sul PC) crea il codice eseguibile. Il codice viene poi trasferito sulla memoria flash del μC (attraverso la connessione USB) ed eseguito.
Dal lato del μC il trasferimento e la scrittura in memoria sono eseguiti da un programma residente (BootLoader) che occupa gli ultimi 512 bytes della memoria flash.
Il programma e' memorizzato permanentemente e viene eseguito ogni volta che si accende la scheda (anche senza aprire Arduino sul PC), finche' non e' sostituito da un altro programma.

Il bootloader

Al momento dell'accensione (o al reset) il bootloader si avvia e verifica se c'e' una richiesta di comunicazione proveniente dalla porta seriale.
Se non c'e' passa il controllo all'applicazione esistente.
Se invece c'e' avvia il dialogo tramite la porta seriale, scarica l'applicazione nuova e poi la avvia.
La presenza del bootloader non è indispensabile. Il programma da eseguire puo' essere caricato connettendo il μC ad una apposita scheda (programmatore).
Questa soluzione e' adatta per situazioni in cui non si prevede che l'applicazione debba subire modifiche o aggiornamenti.



Linguaggio

Un sommario del linguaggio di Arduino puo' essere reperito su:

http://arduino.cc/en/Reference/HomePage

E' disponibile un'ampia libreria software per molteplici applicazioni.
Qui vedremo solo alcune cose fondamentali per iniziare.

I/O Digitale

- pinMode(pin,mode); (mode: *INPUT*, *OUTPUT*, *INPUT_PULLUP*)
- digitalWrite(pin, value); (value: *HIGH*, *LOW*)
- digitalRead(pin);

I/O Digitale: esempio

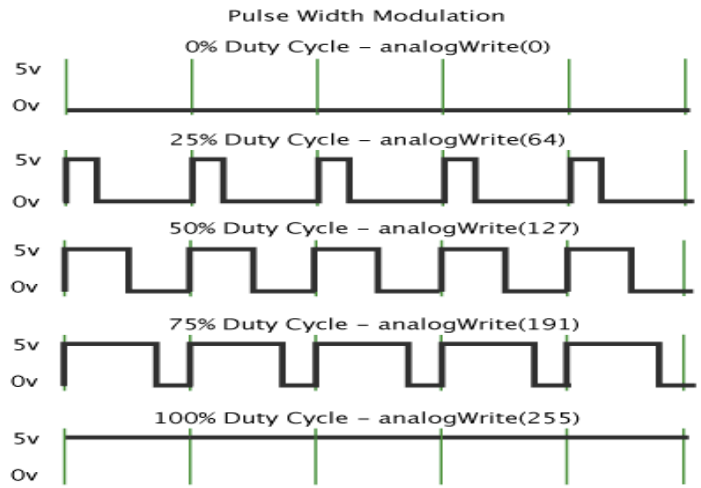
```
int ledPin = 13;
int inPin = 7;
int val = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(inPin, INPUT);
}
void loop()
{
  val = digitalRead(inPin);
  digitalWrite(ledPin, val);
}
```

Output Analogico

I pin 3,5,6,9,10,11 possono, in uscita, essere utilizzati in modo “analogico” (PWM). (Pin 5,6: 980 Hz; Pin 3,9,10,11: 490 Hz)

Esempio:

```
int ledPin = 9;       // LED connected to digital pin 9
int val = 128;          // val: 0-255 (duty cycle)
void setup()
{
  pinMode(ledPin, OUTPUT);    // sets the pin as output
}
void loop()
{
  analogWrite(ledPin, val);   //
}
```



ADC

Il microcontroller contiene un ADC a 10 bit (sample and hold, successive approximation).
E' collegato tramite un multiplexer a 8 ingressi (6 effettivamente utilizzabili su Arduino).

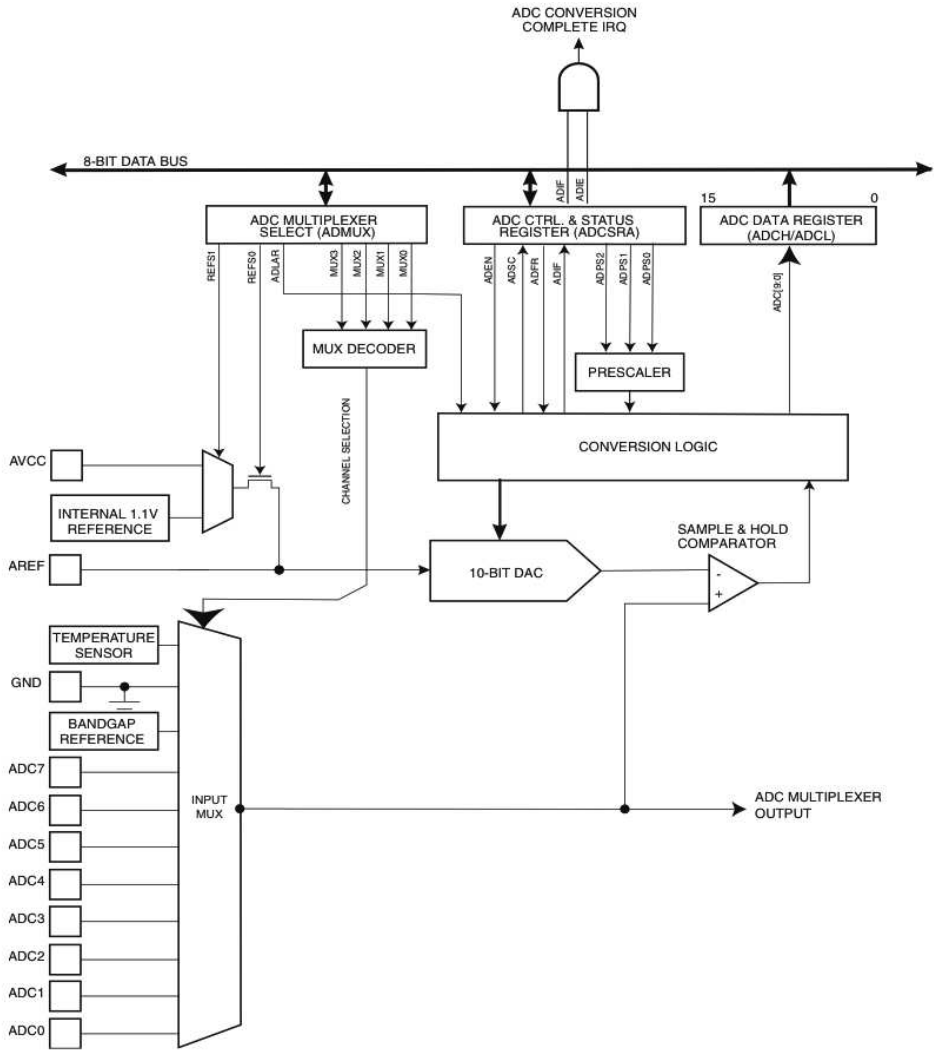
$$Output = \frac{2^{10}}{V_{ref}} V_{in}$$

3 possibili V_{ref}:

- 5 V (default)
- 1.1 V
- A_{ref} (esterna)

Nota bene:

$$0 \leq A_{ref} \leq +5\ V$$



Esempio

```
int analogPin = 3;      // analog pin 3
int val = 0;            // variable to store the value read
void setup()
{
  analogReference(EXTERNAL); // La Vref e' fornita esternamente
  Serial.begin(9600);        //  setup serial
}
void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);            // debug value
}
```

Comunicazione

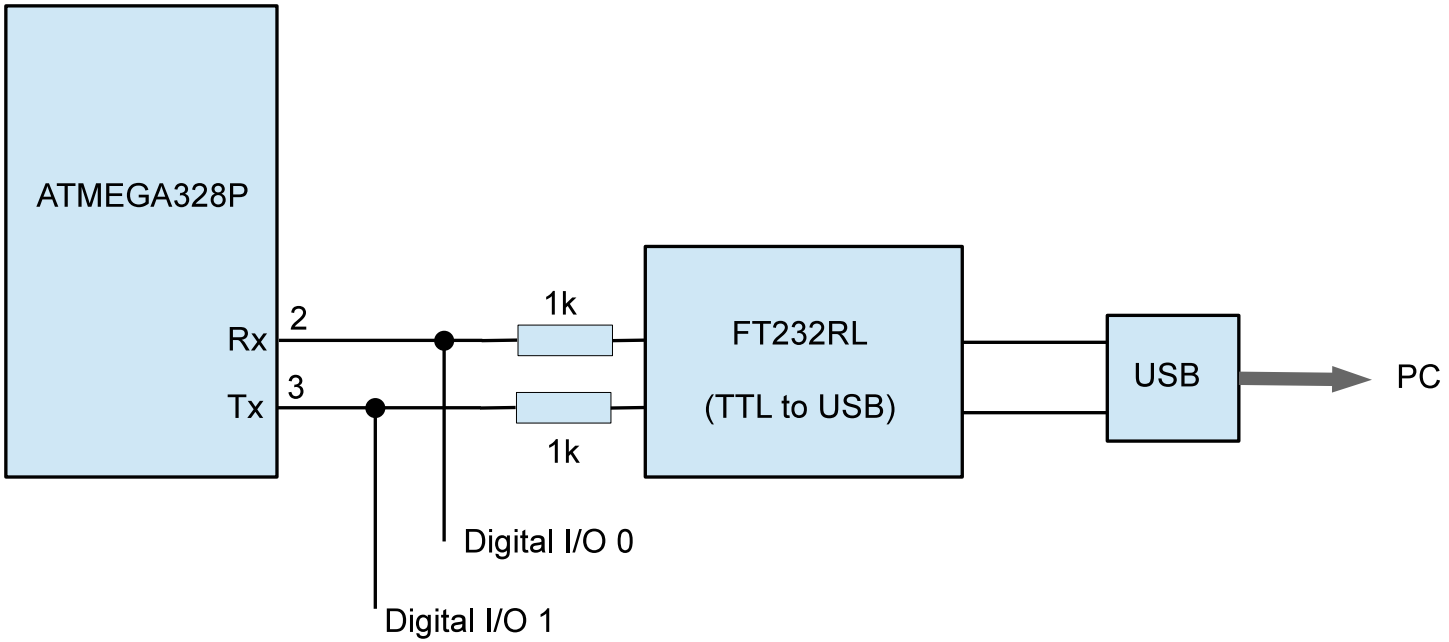
Il microcontrollore supporta vari meccanismi di comunicazione con dispositivi esterni:

- USART (Universal Synchronous Asynchronous Receive Transmit)
  Digital I/O 0 e 1
- SPI (Serial Peripheral Interface)
  Digital I/O 10, 11, 12, 13
- I2C/TWI (Inter Integrated Circuit / Two Wire Interface
  Analog pin A4, A5

Arduino fornisce librerie che facilitano l'uso di questi protocolli.

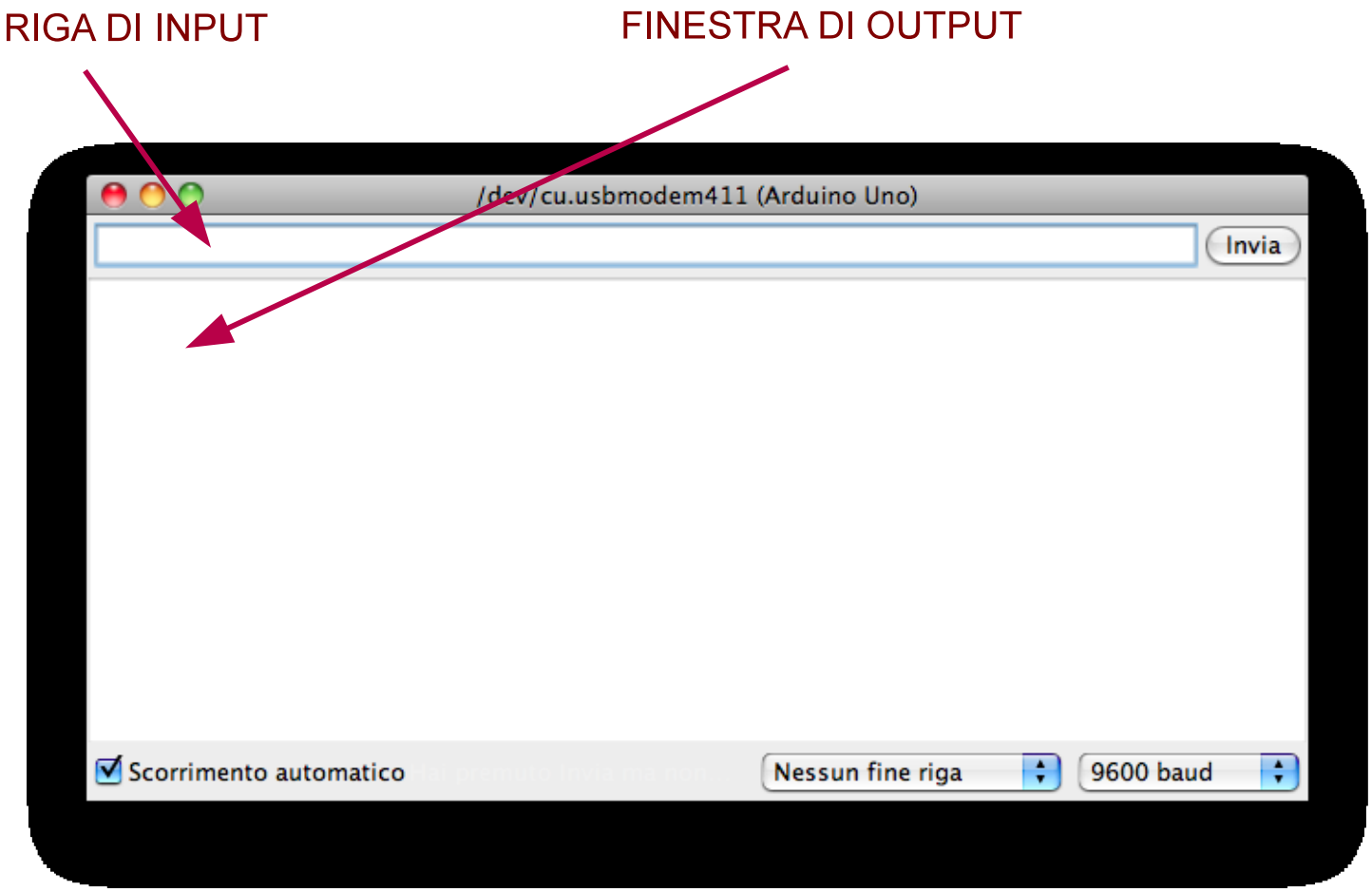
Comunicazione seriale USART

E' utilizzata da Arduino per connettere il μC al PC, tramite la porta USB: da qui viene fatto il download del programma.



Puo' liberamente essere poi utilizzata dall'utente.

Comunicazione seriale; finestra di monitor



Comunicazione seriale: esempio

```
int data = 0;
void setup() {
Serial.begin(9600); //Inizializzazione baud rate
}

void loop() {
  if(Serial.available() > 0) // verifica se esiste un INPUT
  {
data = Serial.parseInt();   // legge come numero intero
Serial.println(data);       // scrive sulla finestra di OUTPUT
delay(200);
  }
}
```

NOTA BENE:

Se si usa la comunicazione seriale i pin I/O 0 *e* 1 *non possono essere usati per altri scopi!*

Interrupts

La scheda Arduino Uno puo' gestire 2 fonti di interrupts

- Interrupt 0: Digital I/O 2;
- Interrupt 1: Digital I/O 3;

Quando arriva un interrupt viene eseguita una routine specificata. Alla fine il controllo torna al programma precedentemente in esecuzione.
Esempio:

```
void setup()
{
  Serial.begin(9600);
  attachInterrupt(0, myprog, CHANGE); //interrupt se I/O 2 cambia
}
void loop()
{
    ....
    ....
}
void myprog()
{
// Interrupt Service Routine
 Serial.println("E' arrivato un interrupt");
}
```

Interrupts:funzioni

`attachInterrupt(interrupt, ISR, mode);`

- *interrupt:* 0 *o* 1*;*
- *ISR: nome della routine di servizio;*
- *mode: LOW,CHANGE, RISING, FALLING*

`detachInterrupt(interrupt);`

Elimina l'interrupt precedentemente abilitato.

`noInterrupts();`

Disabilita tutti gli interrupts.

`interrupts();`

Abilita gli interrupts (precedentemente disabilitati).

Esperienza 8: Familiarizzazione con Arduino

Esperienza 8. Semplici programmi con Arduino

- L'obiettivo di questa esperienza e' di familiarizzare con la scheda Arduino Uno, con il microcontrollore ATMEL ATMega328 e con il relativo software (vedi Appendice per maggiori dettagli).
- E' bene avere sempre disponibile per consultazione il sito internet di Arduino (http://www.arduino.cc) ed in particolare la pagina di reference della programmazione

  `https://www.arduino.cc/en/Reference/HomePage`
- I circuiti da utilizzare possono essere montati sulla consueta scheda sperimentale, connettendola ad Arduino Uno mediante opportuni ponticelli.

Il primo programma...

- "Hello, world!" (–> http://helloworldcollection.de/)
- Ricordarsi di far partire la *Finestra di Monitor* (in genere sotto tools/)

Comunicazione seriale

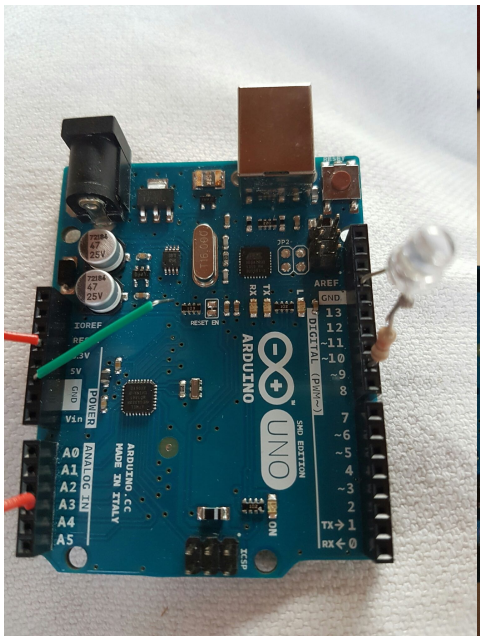
- Programmi di I/O verso la finestra di monitor.
- Misurare la velocita' di esecuzione del μC per varie istruzioni: operazioni aritmetiche, funzioni, operazioni di Input/Output.
- Funzioni di timer: `millis()` e `micros()`
- `delay()` in millisecondi

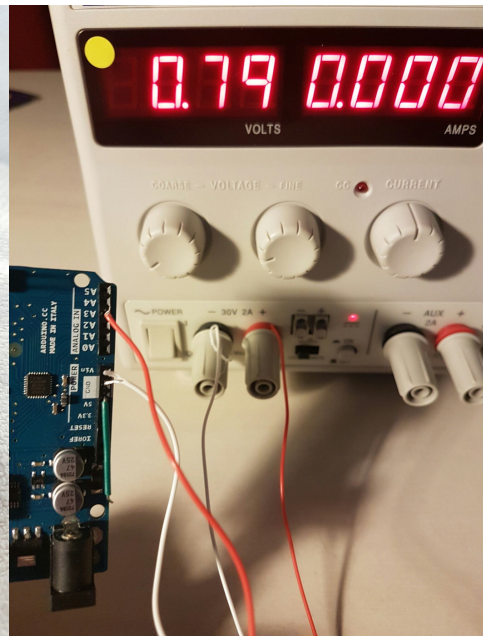
serial_io §

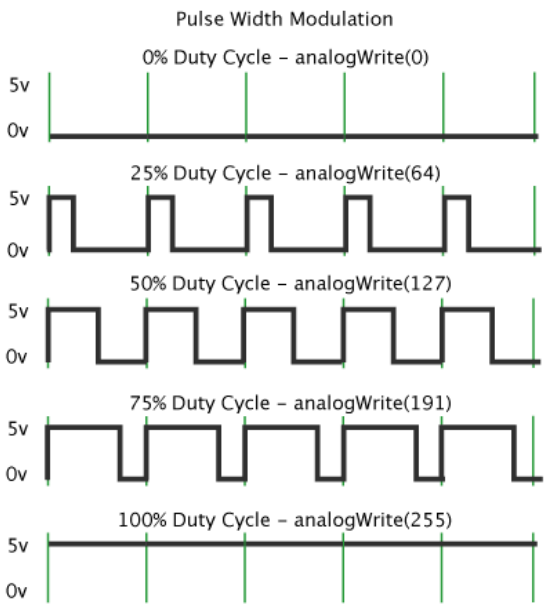
```
long int data=0;
long unsigned t0, t1; //time...

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  Serial.println("Here we go!");
  Serial.println(" ");
}

void loop() { // run over and over
  if (Serial.available() > 0) {
    data = Serial.parseInt();
    t0=micros();
//  | Serial.print("Hai scritto il numero ");
    Serial.println(data);
//    Serial.print("E questo e' moltiplicato per 10: ");
    Serial.println(data*10);
    t1=micros();
    Serial.print("durata: ");
    Serial.print(t0);
    Serial.print(" ");
    Serial.print(t1);
    Serial.print(" ");
    Serial.println(t1-t0);
    delay(100);
  }
}
```







- Led e oscilloscopio per visualizzazione degli outputs
- Input forniti da generatore triplo o dai 5v (3.3V) di Arduino
- `analogRead(analogPin)`, `analogWrite(pin, value)`
- `analogWrite` emula generazione di segnali analogici attraverso modulazione PWM (*Pulse Width Modulation*).
  - PWM permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e di quello negativo (*duty cycle*)

AnalogWrite()

analog_write

```
int ledPin = 9;
int val = 128;

void setup() {
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
Serial.println("Light your fire...");
Serial.println(" ");
}

void loop() {
  if (Serial.available() > 0) {
    val = Serial.parseInt();
 // check value in
    if (val >= 0 && val <= 255) {
     analogWrite(ledPin, val);
     Serial.print("Scritto valore :");
     Serial.println(val);
     }
    else {
     Serial.println("Valore fuori range (0:255)!!!");
     }
  }
}
```

- I pin 3, 5, 6, 9, 10 e 11 possono essere usati in uscita in modo "analogico" (PWM): 0 = 0% ; 255 = 100%
- Pin 5, 6: 980 Hz;
- Pin 3, 9, 10, 11: 490 Hz;
- verificate il comportamento dei pins PWM (uno per tipo...), visualizzando i segnali sull'oscilloscopio, al variare dei valori del duty cycle

AnalogWrite()

Led_Rampa

```
int ledPin = 9;        // LED connected to digital pin 9
int val = 0;          // val: 0-255 (duty cycle)
int i = 0;
int millisecondi = 10;
void setup()
{
  Serial.begin(9600);
  Serial.println("This is an Arduino-based dimmer...");
  pinMode(ledPin, OUTPUT);
}
void loop()
{
for (i=0;i<255;i++)
  {
  val = i;
  analogWrite(ledPin, val);
  delay(millisecondi);
  }
for (i=0;i<255;i++)
  {
  val = 255-i;
  analogWrite(ledPin, val);
  delay(millisecondi);
  }
}
```

- led dovrebbe variare la luminosita' in maniera periodica (provate a cambiarne il periodo)
- Nell'esempio e' riportato un dente di sega, ma potete scrivere anche altri tipi di variazione, ad esempio un'onda triangolare
- Visualizzate con l'oscilloscopio l'uscita del pin 9, dovreste vedere un andamento a "fisarmonica".

`AnalogRead()` e calibrazione ADC Arduino

analog_input

```
int analogPin = 3; // analog pin 3
int RdVal = 0;
float Vx = 0.0;
void setup()
{
analogReference(DEFAULT); // Arduino 5 V
Serial.begin(9600);
Serial.println("Arduino Voltage meter");
Serial.println(" ");

void loop ()
{
RdVal = analogRead(analogPin);
Vx = float(5.0)*(float(RdVal)/float(1023));
delay(1000);
//Serial.print(RdVal);
Serial.print(" Tensione letta: ");
Serial.println(Vx);
}
```

- Utilizzare come Analog Reference il default (5 V) e come tensione da convertire (da porre in ingresso al pin 3) l'uscita dell'alimentatore triplo (0 - 5V);
- confronto tra misura di input (multimetro) e valore prodotto da Arduino (sulla porta seriale)
- costruire retta di taratura utilizzando un numero "congruo" di valori V e ricordando che $Output = \frac{2^{10}-1}{V_{ref}} * V_{in}$

Esperienza 8: Semplici programmi con Arduino: ADC completo

ADC_1

```
int data;
int analogPin = 3;
int i = 0;
long unsigned t0, t1; //time...

const long interval = 2; // 2ms --> sample at 500Hz

void setup() {
  Serial.begin(9600);
  Serial.println("ADC");
}

void loop() {
  t0 = millis();
  for (i = 0; i < 100; i++) { // 100 samples...
    my_delay(interval);
    data = analogRead(analogPin);
    Serial.println(data);
  }
  t1 = millis();
  Serial.print("Durata: ");
  Serial.println(t1 - t0);
  delay(1000);
}

void my_delay(unsigned long interval)
{
  unsigned long MillisStart = millis();
  unsigned long currentMillis = MillisStart;
  //timer
  while (currentMillis - MillisStart <= interval) {
    currentMillis = millis();
  }
}
```

- Campionare **onda sinusoidale e triangolare** e riprodurle in un grafico
- input (pin 3): segnale 0V-5V
- Attenzione alla frequenza di campionamento (in questo caso 500 Hz). Quale e' la max frequenza del segnale di ingresso per evitare *aliasing*?
- per i dati fate cut&paste dalla finestra di serial monitor
- Confrontate il grafico ottenuto con open Office e verificatene la fedelta' del risultato ottenuto rispetto al segnale in ingresso (screenshot dell'oscilloscopio...)