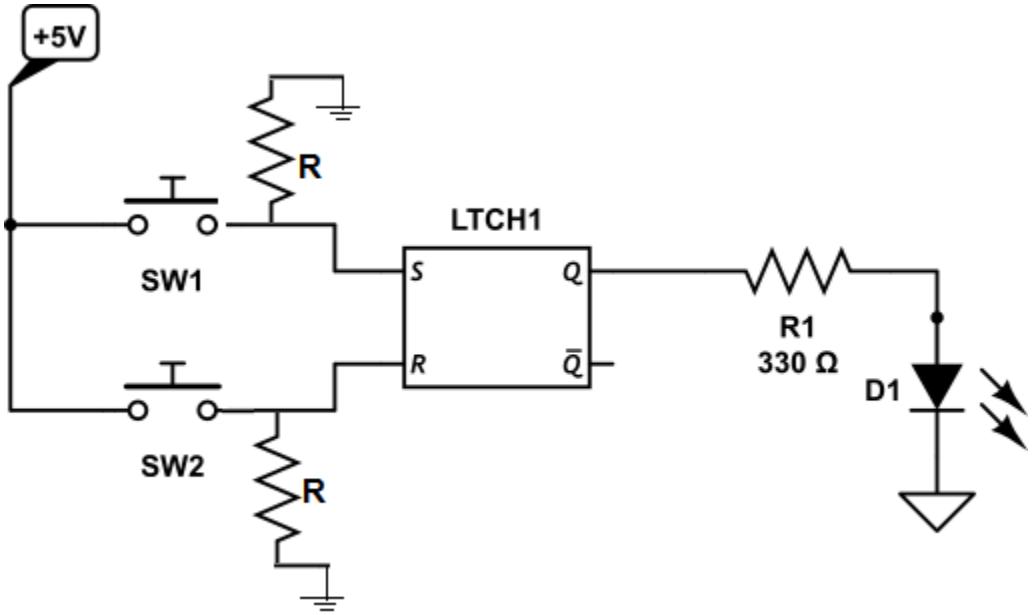
Circuiti antirimbalzo

La figura seguente mostra una semplice applicazione di un latch SR per l'accensione o lo spegnimento di un LED mediante due pulsanti "con ritorno":



Le due resistenze R sono resistenze di pull-down necessarie per essere sicuri che, quando i pulsanti non sono premuti, sugli ingressi sia presente un livello basso (zero logico).

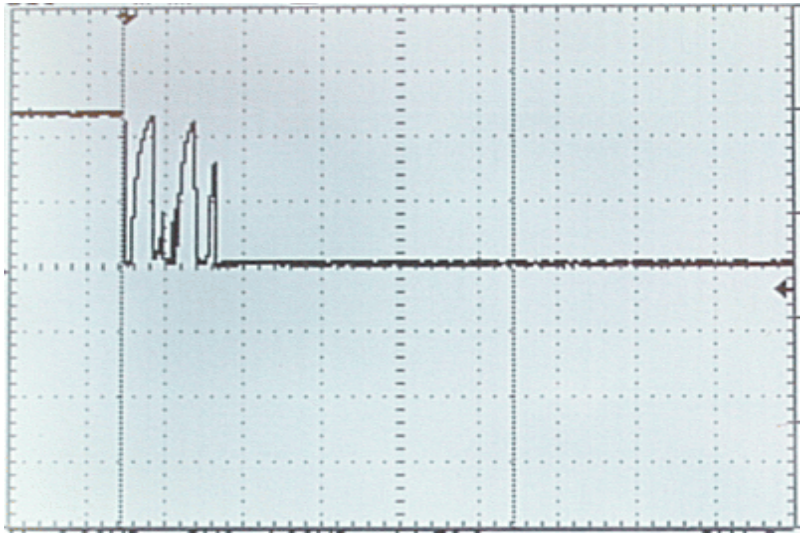
Si noti che il LED si accende quando viene premuto SW1 e rimane acceso (grazie al latch) anche quando il pulsante SW1 viene rilasciato. Analogamente per spegnere il LED bisogna premere SW2.

Circuiti antirimbalzo

La presenza del latch SR svolge anche la funzione di eliminare l'effetto dei "rimbalzi" presenti in ogni interruttore o pulsante meccanico.

Infatti quando un interruttore viene chiuso, tale chiusura non avviene in genere istantaneamente. A causa della flessibilità elastica della lamina metallica interna dell'interruttore, la chiusura produce infatti una serie di micro-*rimbalzi* (in inglese *bounce*), cioè in pratica una rapida sequenza di stati aperto/chiuso in successione.

La figura seguente mostra i tipici rimbalzi visualizzati sullo schermo di un oscilloscopio:



I rimbalzi non creano problemi se il circuito comandato dall'interruttore è un circuito "lento" (come ad esempio una lampadina), mentre possono essere molto dannosi nel caso di circuiti logici, in quanto i rimbalzi dell'interruttore potrebbero essere erroneamente interpretati dal circuito come stati logici alti e bassi validi.

Circuiti antirimbalzo

Il malfunzionamento dipende, dunque, dal fatto che i pulsanti meccanici rimbalzano per alcuni millisecondi nel momento in cui sono premuti e rilasciati, questo è il grafico del segnale elettrico:

Come si evince dalla figura, appena il pulsante viene premuto o rilasciato si genera un segnale “sporco” e per risolvere il problema via software con Arduino , come vedremo più avanti, è necessario aggiungere una funzione ”antirimbalzo”, altrimenti si può agire via hardware con il latch.

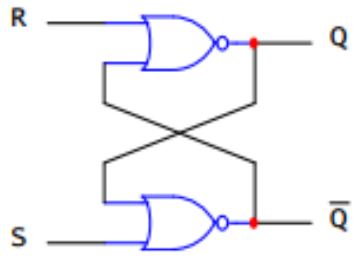
Circuiti antirimbalzo

Il latch SR elimina i rimbalzi in quanto la commutazione dell'uscita avviene sulla prima chiusura (in pratica sul primo rimbalzo) dell'interruttore. A quel punto, grazie alle funzioni di memoria del latch, lo stato dell'uscita rimane memorizzato (stabile) e non cambia anche in presenza di rimbalzi dell'ingresso.

Funzionamento

Un latch SR con porte NOR ha la seguente tabella della verità:

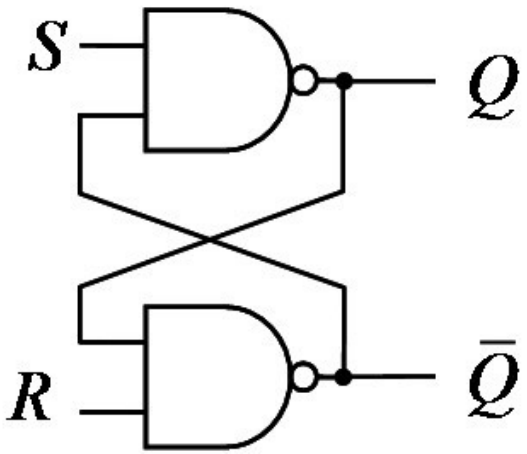
S	R	Q_{n+1}	funzione
0	0	Q_n	memoria
0	1	0	reset
1	0	1	set
1	1	da evitare	------



Per cui supponendo all'atto dell'accensione del dispositivo l'uscita Q allo stato basso e i pulsanti non premuti , il led è spento.

Premendo il pulsante di Set l'uscita Q si porta al livello logico alto e il led si accende. Rilasciando il pulsante il led rimane acceso perché ci si porta nella condizione S=R=0 nella quale lo stato delle uscite del latch o del flip-flop rimane invariato.

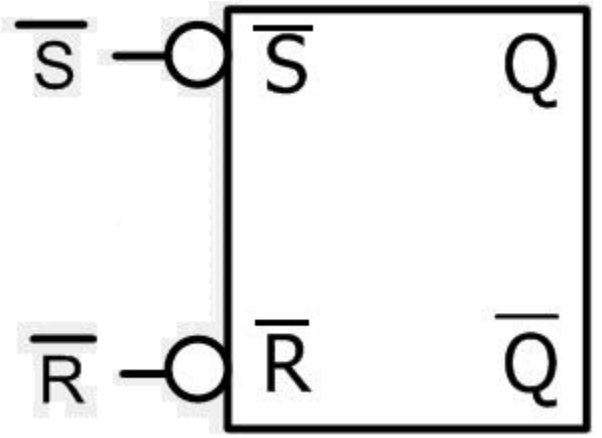
Il latch set-reset a porte NAND



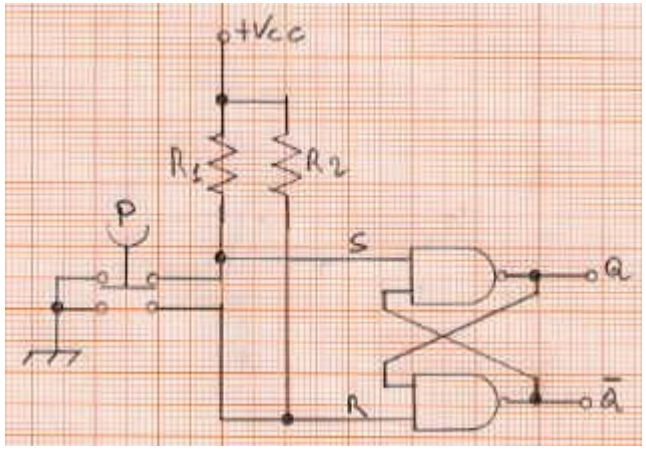
La tabella di verità è la seguente:

S	R	$\overline{Q}$	Q
0	0	not used	not used
0	1	0	1
1	0	1	0
1	1	$\overline{Q}_0$	Q_0

Questo tipo di circuito funziona in logica invertita ovvero in **logica negativa**: per attivare un ingresso (per esempio S) bisogna fornire un valore basso (cioè o nella nostra tabella); per disattivarlo bisogna invece fornire un valore alto.
Questo fatto viene indicato nel seguente modo nel simbolo circuitale del latch:
Si osservi il simbolo di negazione e il "pallino" sugli ingressi che funzionano in logica negativa.



Ciruito antirimbalzo ad un pulsante



S	R	$\bar{Q}$	Q
0	0	not used	not used
0	1	0	1
1	0	1	0
1	1	$\bar{Q}_0$	Q_0

Funzionamento

Quando il pulsante **P** si trova verso l'alto si ha

S = 0; R = 1; Q = 1.

Mentre il pulsante scende verso il basso il falso contatto nella parte superiore del pulsante non riesce a portare R a 0, anche se S può oscillare tra 0 e 1;

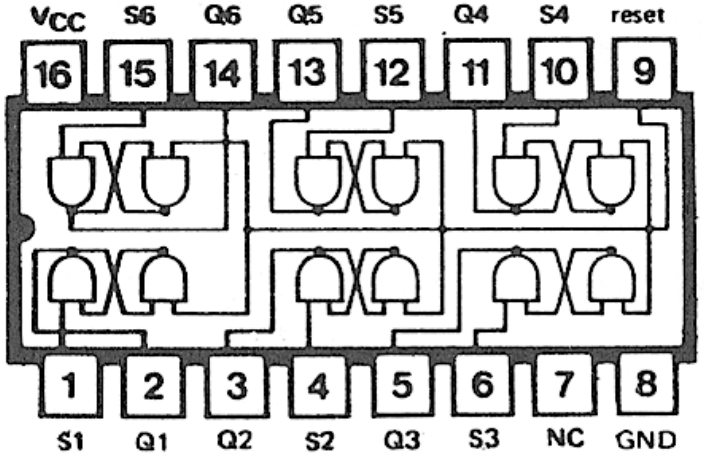
una volta che il pulsante ha toccato la parte inferiore **R = 0; S = 1; Q = 0**;

successivi falsi contatti nella parte inferiore non riusciranno più a far commutare l'uscita, infatti avremo S=1 ed R=1 per cui resterà Q=0;

perché commuti l'uscita occorre che il pulsante ritorni verso l'alto portando S=0 ed R=1 ->Q=0.

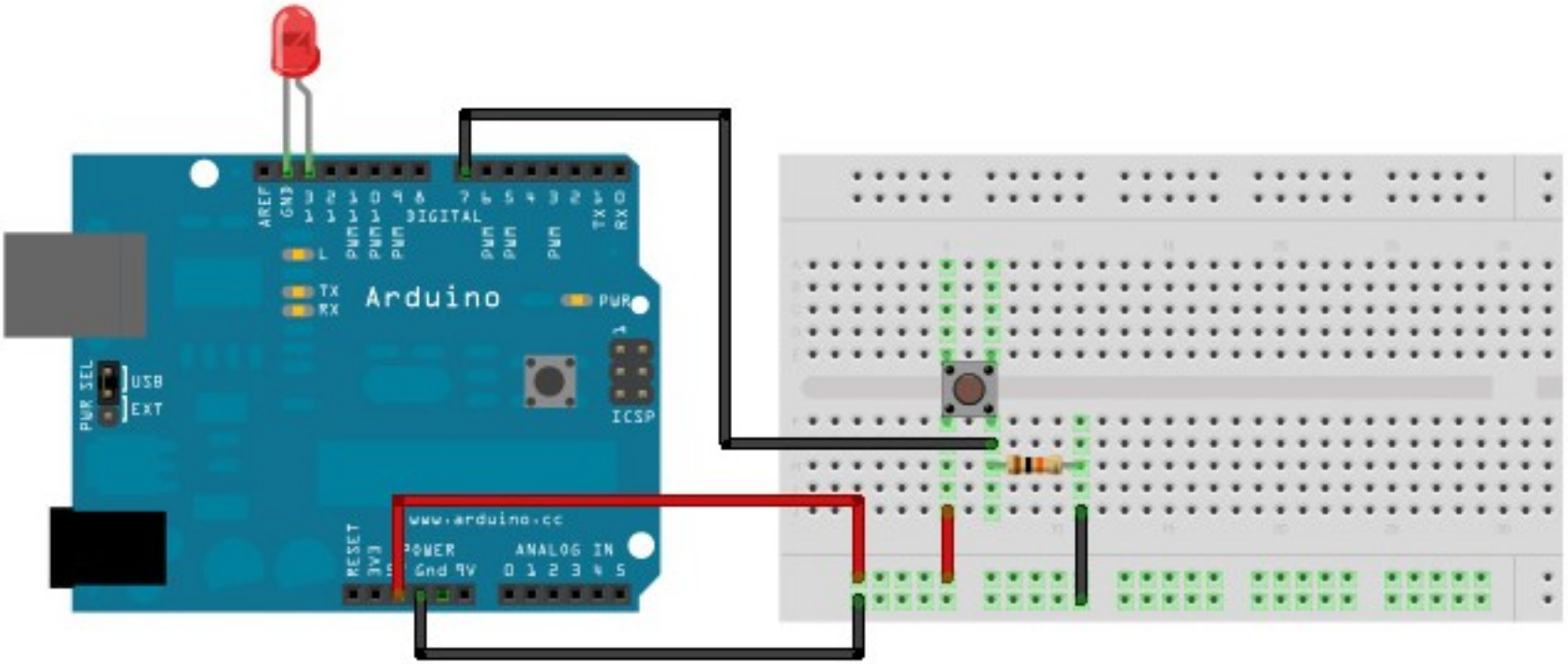
Il latch set-reset integrato

Il latch SR è disponibile anche come integrato della serie CMOS e TTL. A titolo di esempio la figura seguente mostra un 74118 contenente 6 latch SR a porte NAND. Si noti che, per risparmiare sul numero di piedini, l'ingresso di reset è unico per tutti i latch sull'integrato:



Antirimbalzo sw per Arduino

Realizziamo il seguente circuito



Controlliamo un led con un pulsante

Questo il codice usato per controllare il LED:

view plain copy to clipboard print ?

```
01. // Esempio 01: accendi il led appena è premuto il pulsante
02.
03. #define LED 13                  // LED collegato al pin digitale 13
04. #define BUTTON 7                // pin di input dove è collegato il pulsante
05. int  val = 0;                   // si userà val per conservare lo stato del pin di input
06.
07. void setup() {
08.   pinMode(LED, OUTPUT);         // imposta il pin digitale come output
09.   pinMode(BUTTON, INPUT);       // imposta il pin digitale come input
10. }
11.
12. void loop() {
13.   val = digitalRead(BUTTON);  // legge il valore dell'input e lo conserva
14.
15.   // controlla che l'input sia HIGH (pulsante premuto)
16.   if (val == HIGH) {
17.     digitalWrite(LED, HIGH);  //accende il led
18.   }
19.   else {
20.     digitalWrite(LED, LOW);   //spegne il led
21.   }
22. }
```

Controlliamo un led con un pulsante

Se il valore di ***val*** è HIGH (valore logico 1) vuol dire che il pulsante è premuto e allora, tramite l'istruzione **"digitalWrite(LED, HIGH);"** viene acceso il LED, se il primo confronto risulta falso, ciò vuol dire che il pulsante non è premuto come conseguenza viene eseguita la parte di codice **else** che spegne il LED, ciò viene eseguito con l'istruzione: **"digitalWrite(LED, LOW);"**

Però lo scopo di questa lezione è quello di realizzare un circuito simile a quello di un impianto di illuminazione: "premo il pulsante accendo la luce, premo una seconda volta e spengo la luce", l'esempio precedente, ci costringe a mantenere il dito sul pulsante per mantenere acceso il led.

Controlliamo un led con un pulsante

Questa è una prima versione che risolve il problema:

view plain copy to clipboard print ?

```
01. // Esempio 02: accendi il led appena è premuto il pulsante mantenendolo acceso quando s
02. // premendo una seconda volta il pulsante spegne il led
03.
04. #define LED 13                  // LED collegato al pin digitale 13
05. #define BUTTON 7                // pin di input dove è collegato il pulsante
06. int val = 0;                    // si userà val per conservare lo stato del pin di input
07. int stato = 0;                  // ricorda lo stato in cui si trova il led, stato = 0 led
08.
09. void setup() {
10.   pinMode(LED, OUTPUT);         // imposta il pin digitale come output
11.   pinMode(BUTTON, INPUT);       // imposta il pin digitale come input
12. }
13.
14. void loop() {
15.   val = digitalRead(BUTTON);  // legge il valore dell'input e lo conserva
16.
17.   // controlla che l'input sia HIGH (pulsante premuto)
18.   // e cambia lo stato del led
19.   if (val == HIGH) {
20.     stato = 1 - stato;
21.   }
22.
23.   if (stato == 1) {
24.     digitalWrite(LED, HIGH);   // accende il led
25.   }
26.   else {
27.     digitalWrite(LED, LOW);    //spegne il led
28.   }
29. }
```

Controlliamo un led con un pulsante

Spiegazione:

Passo 1: pulsante non premuto – diodo spento

Il valore assunto da ***val*** è **0** in quanto il pulsante non è premuto (l'istruzione "**digitalRead(BUTTON);**" restituisce **0**), la condizione logica **val == HIGH** del prim **if** restituisce **falso** e la variabie **stato** resta al valore iniziale **0**.
Poiché il confronto **stato == 1** del secondo **if** restituisce falso, viene eseguito l'else in cui è presente l'istruzione "**digitalWrite(LED, LOW);**" che **spegne il diodo led**.

Passo 2: pulsante premuto – diodo acceso

Il valore assunto da ***val*** è **1** in quanto il pulsante è premuto (l'istruzione "**digitalRead(BUTTON);**" restituisce **1**), la condizione logica **val == HIGH** del primo **if** restituisce **vero** e la variabie **stato** viene impostata a **stato = 1 – 0**, cioè a **1**.
Poiché il confronto **stato == 1** del secondo **if** restituisce vero, viene eseguita l'istruzione che segue, cioè: "**digitalWrite(LED, HIGH);**" che **accende il diodo led.**

Passo 3: pulsante premuto – diodo spento

Il valore assunto da ***val*** è **1** in quanto il pulsante è premuto (l'istruzione "**digitalRead(BUTTON);**" restituisce **1**), la condizione logica **val == HIGH** del primo **if**restituisce **vero** e la variabie stato viene impostata a **stato = 1 – 1** (stato è stato impostato ad 1 nel passo precedente), cioè a **0**.
Poiché il confronto **stato == 1** del secondo **if** restituisce **falso**, viene eseguito l'else in cui è presente l'istruzione "**digitalWrite(LED, LOW);**" che **spegne il diodo led.**

Controlliamo un led con un pulsante

Se provate il programma noterete un funzionamento molto strano, potrebbe capitare che premendo e rilasciando il pulsante il led non si accende e si spegne correttamente, ad esempio premo e non si accende oppure se acceso premendo il pulsante non si spegne il led.

Ciò è dovuto al fatto che Arduino legge le istruzioni del vostro programma ad una velocità di milioni di istruzioni al secondo ciò vuol dire che la lettura dello stato del pulsante viene letta moltissime volte al secondo e l'accensione e lo spegnimento del diodo led potrebbe essere imprevedibile.

Per comprendere questa situazione immaginate che al Passo 3 precedente si continui a mantenere premuto il pulsante, noterete che la variabile **stato** ad ogni ciclo, assume valori diversi che possono portare ad una situazione di accensione o spegnimento del diodo.

Per ovviare a questo problema bisogna riuscire a identificare il momento esatto in cui il pulsante viene premuto è questo può essere fatto conservando in una variabile lo stato del pulsante al passo precedente con una nuova versione del programma:

Controlliamo un led con un pulsante

view plain copy to clipboard print ?

```
01. // Esempio 03: antirimbalzo
02. // accendi il led appena è premuto il pulsante mantenendolo acceso quando si rilascia
03. // premendo una seconda volta il pulsante spegne il led
04.
05. #define LED 13                // LED collegato al pin digitale 13
06. #define BUTTON 7              // pin di input dove è collegato il pulsante
07. int val = 0;                  // si userà val per conservare lo stato del pin di input
08. int vecchio_val = 0;          // si userà vecchio_val per conservare lo stato del pin d
09. int stato = 0;                // ricorda lo stato in cui si trova il led, stato = 0 led
10.
11. void setup() {
12.   pinMode(LED, OUTPUT);       // imposta il pin digitale come output
13.   pinMode(BUTTON, INPUT);     // imposta il pin digitale come input
14. }
15.
16. void loop() {
17.   val = digitalRead(BUTTON);  // legge il valore dell'input e lo conserva
18.
19.   // controlla se è accaduto qualcosa
20.   if ((val == HIGH) && (vecchio_val == LOW)){
21.     stato = 1 - stato;
22.   }
23.
24.   vecchio_val = val;            // ricordiamo il valore precedente di val
25.
26.   if (stato == 1) {
27.     digitalWrite(LED, HIGH);   // accende il led
28.   }
29.   else {
30.     digitalWrite(LED, LOW);    //spegne il led
31.   }
32. }
```

Controlliamo un led con un pulsante

In questo caso usiamo la variabile **vecchio_val** per conservare lo stato al ciclo precedente, in questo modo verifichiamo lo stato effettivo del pulsante verificando se lo stato attuale è "**pulsante premuto**" E "**pulsante non premuto al passo precedente**", se la condizione è vera viene modificato lo stato.
L'operatore boleano utilizzato è AND che nel linguaggio di programmazione si indica con &&.

La cosa più bella nell'applicare l'informatica al mondo reale e che è indispensabile far i conti con le caratteristiche fisiche dei dispositivi su cui operiamo. Non appena aumentiamo la frequenza con cui premiamo il pulsante, si ricade in una situazione di incongruenza per cui il led non risponde più ai comandi. Questo problema avviene perché il pulsante è un apparato meccanico costituito da contatti elettrici ed una molla, quando premiamo e rilasciamo, si manifestano situazioni di rimbalzo del contatto che creano dei segnali non corretti, detti spuri, che modificano lo stato del diodo led.
Per risolvere il problema è sufficiente attendere che questi rimbalzi spuri si concludano e quindi bisogna attendere una certa quantità di tempo dopo che è stato rilevato un cambiamento di stato, provate con valori non superiori a 50 millisecondi, nell'esempio ho introdotto un valore di **15 millisecondi di ritardo**.

Controlliamo un led con un pulsante

view plain copy to clipboard print ?

```
01. // Esempio 04: antirimbalzo2 - accendi il led appena è premuto il pulsante mantenendolo
02. // premendo una seconda volta il pulsante spegne il led
03.
04. #define LED 13                  // LED collegato al pin digitale 13
05. #define BUTTON 7                // pin di input dove è collegato il pulsante
06. int val = 0;                    // si userà val per conservare lo stato del pin di input
07. int vecchio_val = 0;            // si userà vecchio_val per conservare lo stato del pin d
08. int stato = 0;                  // ricorda lo stato in cui si trova il led, stato = 0 led
09.
10. void setup() {
11.   pinMode(LED, OUTPUT);        // imposta il pin digitale come output
12.   pinMode(BUTTON, INPUT);      // imposta il pin digitale come input
13. }
14.
15. void loop() {
16.   val = digitalRead(BUTTON);  // legge il valore dell'input e lo conserva
17.
18.   // controlla se è accaduto qualcosa
19.   if ((val == HIGH) && (vecchio_val == LOW)){
20.     stato = 1 - stato;
21.     delay(15);                 // attesa di 15 millisecondi
22.   }
23.
24.   vecchio_val = val;             // ricordiamo il valore precedente di val
25.
26.     if (stato == 1) {
27.     digitalWrite(LED, HIGH);   // accende il led
28.   }
29.   else {
30.     digitalWrite(LED, LOW);    //spegne il led
31.   }
32. }
```

Esercizio:
realizzare un programma che esegue questa funzione: quando si preme un pulsante il diodo led lampeggia, quando premo una seconda volta il pulsante il led termina di lampeggiare:

view plain copy to clipboard print ?

```
01. // Esempio 05: led lampeggia se premo il pulsante
02. // premendo una seconda volta il pulsante si spegne il led
03.
04. #define LED 13                // LED collegato al pin digitale 13
05. #define BUTTON 7              // pin di input dove è collegato il pulsante
06. int val = 0;                  // si userà val per conservare lo stato del pin di input
07. int vecchio_val = 0;          // si userà vecchio_val per conservare lo stato del pin d
08. int stato = 0;                // ricorda lo stato in cui si trova il led, stato = 0 led
09.
10. void setup() {
11.   pinMode(LED, OUTPUT);       // imposta il pin digitale come output
12.   pinMode(BUTTON, INPUT);     // imposta il pin digitale come input
13. }
14.
15. void loop() {
16.   val = digitalRead(BUTTON);  // legge il valore dell'input e lo conserva
17.
18.   // controlla se è accaduto qualcosa
19.   if ((val == HIGH) && (vecchio_val == LOW)){
20.     stato = 1 - stato;
21.     delay(15);                // attesa di 15 millisecondi
22.   }
23.
24.   vecchio_val = val;          // ricordiamo il valore precedente di val
25.
26.     if (stato == 1) {
27.     digitalWrite(LED, HIGH);  // accende il LED
28.     delay(1000);              // aspetta un secondo
29.     digitalWrite(LED, LOW);   // spegne il LED
30.     delay(1000);              // aspetta un secondo
31.   }
32.   else {
33.     digitalWrite(LED, LOW);    //spegne il led
34.   }
35. }
```