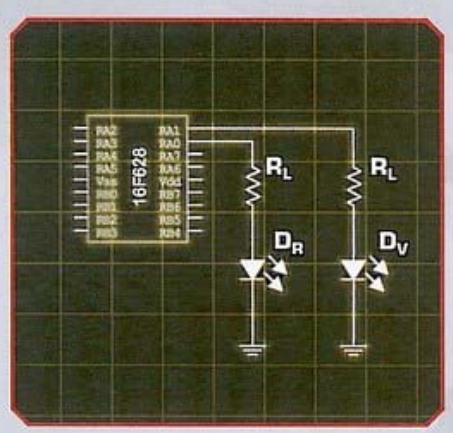
ACCENDIAMO IL NOSTRO TIMER

Mettiamo in pratica quanto visto in modo teorico nel fascicolo precedente, scrivendo e testando il codice sorgente che ci consentirà di effettuare il primo esperimento sull'uso dei timer.

Nel fascicolo precedente abbiamo introdotto la teoria alla base del funzionamento dei **timer**; non ci resta che applicarla concretamente per giungere alla scrittura del sorgente del firmware di esempio. Prima di tutto, però, definiamo quello che sarà il circuito su cui vogliamo operare. Per iniziare, ovviamente, affronteremo un problema estremamente semplice, che **sfrutterà il TIMERØ e la generazione degli interrupt per produrre un fenomeno di intermittenza luminosa su un LED collegato in uscita a uno dei pin del microcontrollore.** Per mostrare il funzionamento 'autonomo' della funzione di risposta agli interrupt, **inseriremo un secondo LED**, con l'obiettivo di **rendere anch'esso intermittente**, ma controllandolo direttamente attraverso la **funzione 'main'** del programma.

IL CIRCUITO ›››

Il problema descritto coinvolge, di conseguenza, la **gestione di due diodi a emissione luminosa** (ne useremo uno rosso e uno verde), che devono essere trattati in maniera completamente indipendente. Il **diodo rosso** sarà collegato al **pin 0 della porta A** e verrà **pilotato utilizzando la funzione di risposta agli interrupt prodotti dal TIMERØ.** Il **LED verde,** al contrario, dovrà essere collegato al **pin 1 della porta A** e sarà **gestito direttamente da un loop contenuto nella funzione 'main'.** Per entrambi i LED imporremo una **frequenza di funzionamento di 2 Hz** (tale da generare, cioè, due lampeggi al secondo). Nello schema sotto puoi vedere come il PIC 16F628 dovrà essere collegato ai LED (con i soliti resistori di limitazione).



In questo Workshop utilizzeremo il PIC 16F628 per controllare due LED collegati ai pin RA0 e RA1.

LA STRUTTURA DEL FIRMWARE ›››

Prima di vedere e commentare il sorgente del firmware di questo Workshop soffermiamoci a pensare a come dovrà essere strutturato. Partiamo dalla **funzione 'main',** che dovrà essere strutturata in **due parti.** Nella prima parte si occuperà di **applicare le configurazioni di base al TIMERØ e di predisporre il PIC all'utilizzo degli interrupt,** agendo opportunamente sugli appositi registri (**INTCON** e **OPTION**). In un secondo tempo, terminata l'inizializzazione dell'hardware, potrà essere **avviato il loop infinito con cui temporizzeremo l'accensione del LED verde.** Dovrà, inoltre, essere dichiarata la **funzione 'interrupt'** che verrà chiamata ogniqualvolta il TIMERØ genererà un overflow. Tale funzione gestirà **l'accensione e lo spegnimento del LED** e la **reinizializzazione del timer e degli interrupt.** Per i parametri di temporizzazione utilizzati in questo fascicolo, fai riferimento all'ultimo paragrafo del Workshop 67. Nelle pagine seguenti, invece, ti mostriamo il sorgente del firmware di test che useremo nello StepbyStep.

Il primo esperimento con il timer numero 0

```
/* Definiamo i valori simbolici ACCESO e SPENTO che ti permetteranno di portare
'alti' e 'bassi' i valori dei pin di output*/
#define ACCESO 1
#define SPENTO 0

/* definiamo il valore simbolico 'START' che utilizzeremo come valore di inizializzazione per
il TIMER0. Utilizziamo il valore '4' in quanto tra la richiesta di interrupt e l'avvio della
funzione di risposta trascorrono, appunto, 4 cicli di clock.*/
#define START 4

/* definiamo i simboli LED_ROSSO e LED_VERDE associati ai pin del PIC a cui verranno collega-
ti i due diodi. L'utilizzo di questi due valori simbolici permette di rendere più 'leggibile'
il codice, non rendendo più necessario ricordarsi quale dei due led è collegato al pin RA0 e
quale al pin RA1. */
#define LED_ROSSO PORTA.F0
#define LED_VERDE PORTA.F1


/* la variabile 'numero_conta' verrà utilizzata per memorizzare il numero di chiamate di
interrupt che si sono verificate dall'ultimo cambiamento di stato del LED ROSSO. Quando nume-
ro_conta ha raggiunto il valore 15 significa che sono trascorsi circa 250 msec e il LED rosso
deve cambiare stato */
int numero_conta;


/* definizione della funzione interrupt che verrà chiamata in risposta alle richieste di
interrupt */
void interrupt (void)
{
  /* ad ogni chiamata di interrupt incremento la variabile 'numero_conta' per contare le
occorrenze*/
  numero_conta++;

  /* se è stata chiamata una sequenza di 15 interrupt significa che sono trascorsi circa 250
msec ed è necessario invertire lo stato del LED rosso */
  if (numero_conta == 15)
  {
   /* se sono trascorsi 250 msec inverto lo stato del LED rosso. */
   if(LED_ROSSO == ACCESO)
   {
      LED_ROSSO = SPENTO;
   }
   else
   {
      LED_ROSSO = ACCESO;
   }

   /* nel caso in cui sia stato invertito lo stato del LED resetto 'numero_conta'*/
   numero_conta = 0;
  }

  /* terminata la chiamata di interrupt, reimpostiamo i registri del PIC per abilitarlo alla
generazione del prossimo interrupt */

  /* reimposto il TIMER0 al valore di avvio */
  TMR0 = START;
```

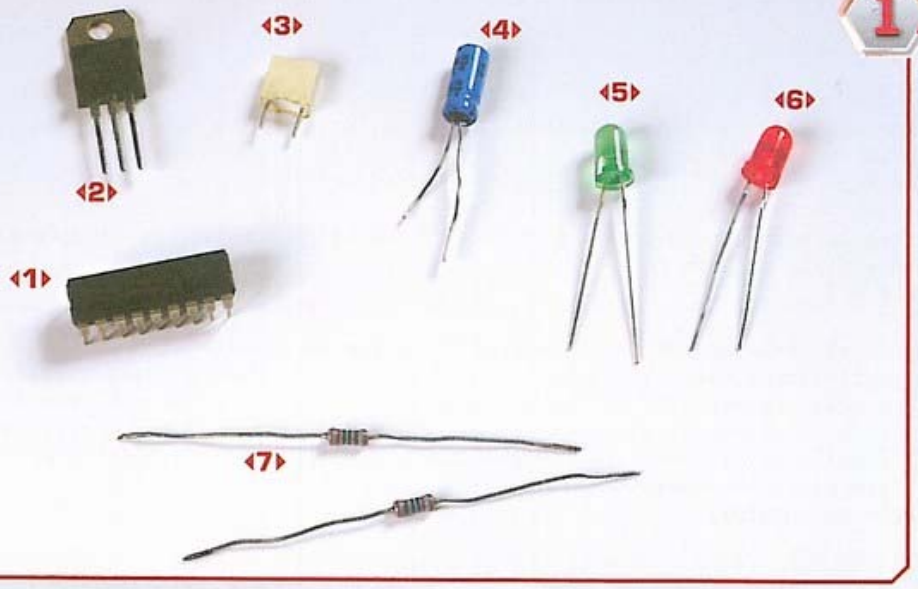
```
/* reimposto il bit T0IF del registro INTCON  in modo resettare l'interrupt flag associato
all'overflow del TIMER0*/
  INTCON.T0IF = 0;
}


/* la funzione main alla base del firmware */
void main(void)
{
 /* inizializzazione dell'hardware */
 /* impostiamo tutti i pin della porta A come pin di output */
 TRISA = 0b00000000;

 /* disabilitiamo i comparatori in modo da poter utilizzare i pin della porta A come I/O
digitale */
 CMCON = 0b00000111;

 /* attivazione degli interrupt */
 /* abilitiamo il bit 'GLOBAL INTERRUPT ENABLE' per attivare la generazione degli
    interrupt del PIC */
 INTCON.GIE = 1;

/* abilitiamo il bit 'TIMER0 OVERFLOW INTERRUPT ENABLE' per abilitare la generazione degli
interrupt associati all'overflow del timer 0 */
 INTCON.T0IE = 1;

/* impostiamo il registro OPTION (in mikroC vi si accede tramite il nome 'OPTION_REG') por-
tando, in particolare, bassi i pin 5, 4 e 3 in modo da decidere di temporizzare il TIMER0
tramite l'oscillatore del PIC, di far incrementare il suo contatore nel passaggio da '0' a
'1' del clock e di assegnare il prescaler al TIMER0. Gli ultimi 3 bit configurati a '101',
infine, ci permettono di impostare il prescaler a un rapporto di riduzione pari a 1:64, come
deciso nel precedente Workshop */
 OPTION_REG = 0b10000101;

 /* inizializzazione timer con il valore di avvio */
 TMR0 = START;

 /* attivazione dei due LED */
 LED_ROSSO = ACCESO;
 LED_VERDE = ACCESO;

/* inizializzazione della variabile 'numero_conta' a 0 */
 numero_conta = 0;


/*  loop infinito di funzionamento del firmware */
 while (1)
 {
   /* inversione dello stato del LED verde */
   if(LED_VERDE == ACCESO)
   {
      LED_VERDE = SPENTO;
   }
   else
   {
      LED_VERDE = ACCESO;
   }

   /* generazione di un ritardo di 250 msec attraverso la funzione Delay_msec */
   Delay_msec(250);

}
```

STEPbySTEP

L'INTERRUPT DI OVERFLOW DEL TIMER NUMERO 0

Passiamo ora alla realizzazione fisica del circuito di test e al suo collaudo.

1



‹1› un microcontrollore 16F628 programmato con il firmware di test che abbiamo presentato nelle pagine precedenti (**PIC**)

‹2› un regolatore di tensione 2940 da 5 V (**REG**)

‹3› un condensatore da 0,47 µF (C_{IN})

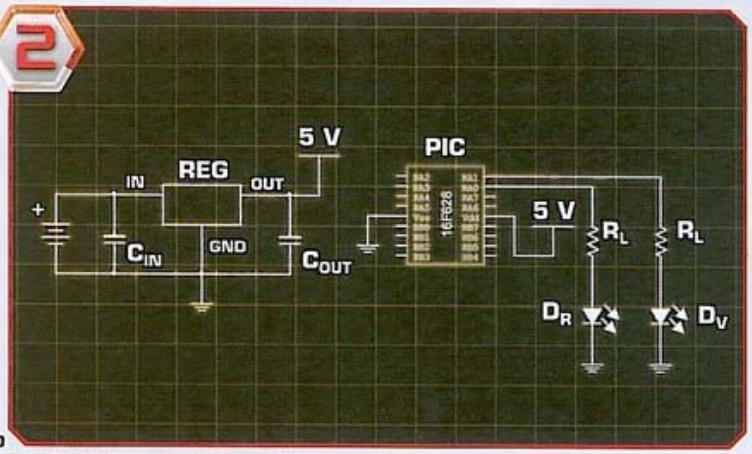
‹4› un condensatore da 10 µF (C_{OUT})

‹5› un LED verde (D_V)

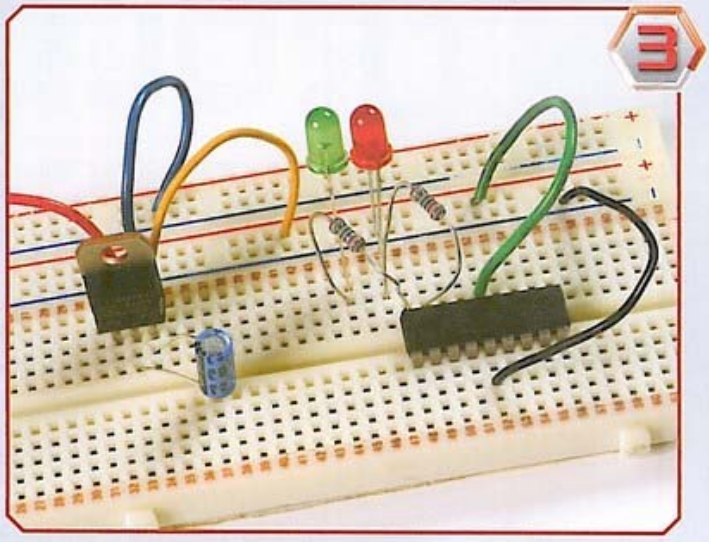
‹6› un LED rosso (D_R)

‹7› due resistori da 220 ohm (R_L)

2



Nell'immagine a lato è mostrato lo schema del circuito che dovrai realizzare. Come nei Workshop precedenti, anche in questo caso utilizziamo il regolatore **2940** per ottenere i **5 V** di alimentazione del **microcontrollore**. I **LED**, invece, dovranno essere collegati ai pin **RA0** (**LED rosso**) e **RA1** (**LED verde**) inserendo **un resistore da 220 ohm su ognuna delle due linee**, in modo da limitare la corrente emessa dal **PIC**.



Ecco come apparirà il circuito una volta che lo avrai assemblato. Se lo avrai montato correttamente, non appena collegherai l'alimentazione il PIC accenderà i LED in modo intermittente. Noterai, tuttavia, che i due diodi non lampeggeranno esattamente alla stessa frequenza. Questo per due semplici motivi: innanzitutto gli interrupt non sono generati ogni 250 msec esatti, a causa dell'approssimazione che abbiamo applicato con la scelta dei parametri di funzionamento del prescaler e dal fatto che, per come è stata scritta la funzione 'interrupt', tra l'inizio della risposta all'interrupt e l'avvio del processo di temporizzazione successivo (reset del timer e del flag T0IF) trascorre un certo lasso di tempo nel quale il PIC esegue la risposta all'interruzione. Inoltre, se la richiesta di interrupt viene generata mentre il programma principale è impegnato nell'esecuzione della funzione 'Delay_msec' (utilizzata per generare il ritardo di 250 msec che temporizza il LED verde), quest'ultima viene 'congelata' per tutto il periodo di tempo necessario all'esecuzione della risposta all'interruzione comportando, quindi, un allungamento effettivo di tale periodo di tempo (il ritardo sarà, di conseguenza, leggermente superiore ai 250 millisecondi teorici). A questo punto, puoi sperimentare come varia l'intermittenza utilizzando valori diversi per il prescaler o per l'inizializzazione del TIMERØ.

ALCUNE NOTE SUL SORGENTE ›››

Il sorgente che abbiamo mostrato come esempio in questo Workshop sembra contraddire quanto abbiamo detto nei numeri precedenti in merito alla necessità di identificare la sorgente della richiesta di interrupt. Di fatto, se confronti la struttura della funzione 'interrupt' presentata in questo fascicolo con l'esempio contenuto nel box a pagina 8 del Workshop 66, noterai sicuramente la mancanza del test di controllo:

```
if (INTCON.T0IF==1) {...}
```

che serve appunto a riconoscere l'interrupt generato dall'overflow del timer numero 0. Nel caso odierno (ma ciò vale anche per tutti i casi analoghi) tale test non è necessario, proprio perché il PIC 16F628 è stato configurato in maniera tale da utilizzare un solo interrupt (non si potevano generare, di conseguenza, ambiguità). Resta, comunque, invariata la necessità di resettare l'interrupt flag al termine dell'implementazione della funzione di risposta all'interruzione, inserendo la semplice istruzione di assegnamento:

```
INTCON.T0IF==1
```