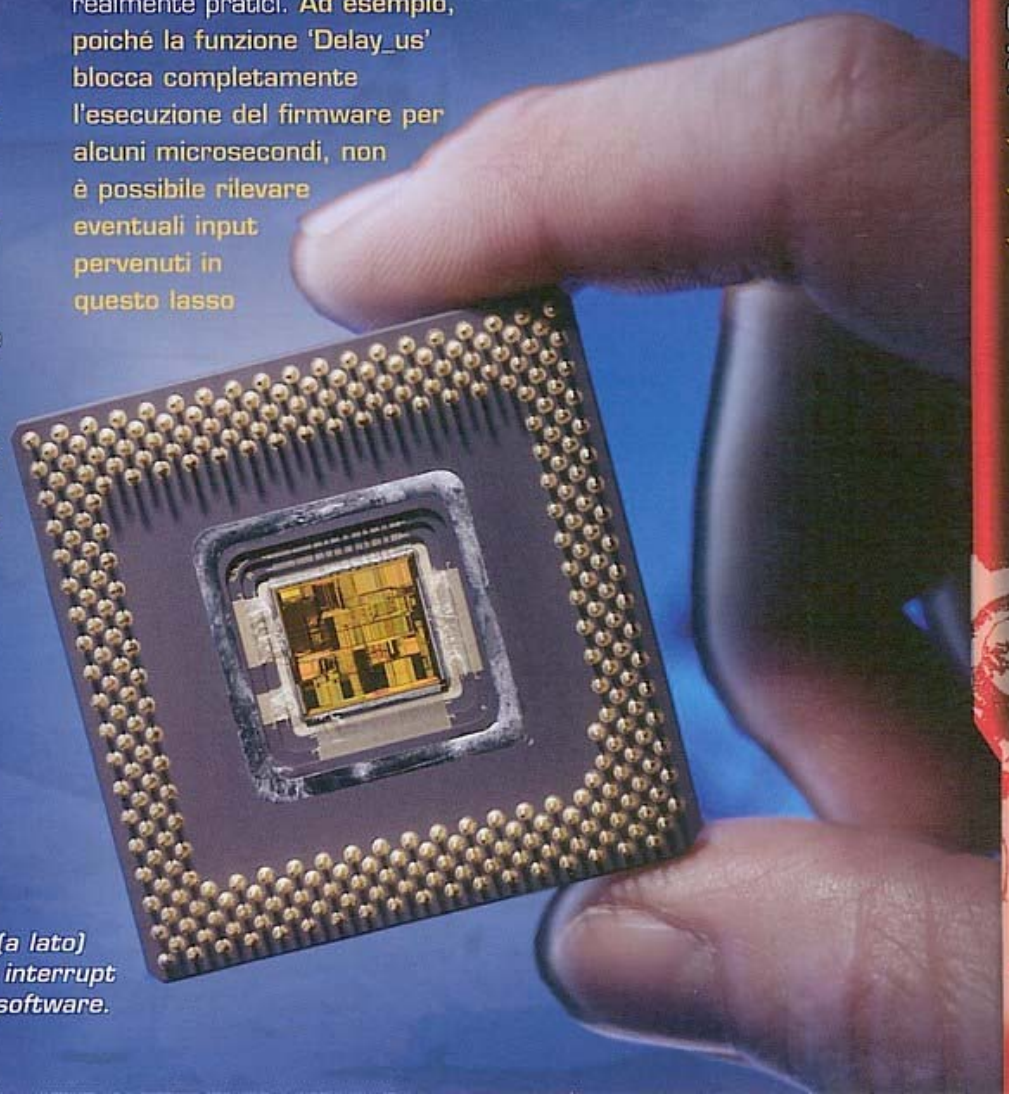
INTERRUPT ED EFFICIENZA

In questo Workshop introduciamo un nuovo argomento inerente alla programmazione dei microcontrollori: l'utilizzo degli interrupt.

Negli ultimi esperimenti che abbiamo condotto utilizzando il PIC 16F628 abbiamo approcciato la **programmazione in maniera sequenziale** con l'idea, cioè, che ogni istruzione C potesse essere eseguita esclusivamente al termine della funzione che la precede. Ma se da un lato questo paradigma di progettazione è di facile comprensione, dall'altro ci può portare a situazioni di scarsa efficienza. Prendiamo come esempio la funzione **'Delay_us'** che abbiamo impiegato per temporizzare il firmware dei fascicoli 64 e 65: tale funzione si comporta, in pratica, come un ciclo 'for' appositamente tarato che 'brucia' l'esecuzione del programma per un certo periodo di tempo (ed è proprio questo il concetto di 'delay', ossia di 'ritardo'). Tuttavia, può essere più utile per fini applicativi 'scorporare' un processo periodico (come la generazione degli impulsi) o più in generale la gestione degli 'eventi' dall'esecuzione del programma principale.

GLI INTERRUPT ›››

Negli esempi di codice dei Workshop precedenti abbiamo visto come le soluzioni proposte per la lettura degli input digitali e la temporizzazione fossero caratterizzate da una serie di problematiche che le rendevano, di fatto, poco adatte a fini realmente pratici. **Ad esempio, poiché la funzione 'Delay_us' blocca completamente l'esecuzione del firmware per alcuni microsecondi, non è possibile rilevare eventuali input pervenuti in questo lasso di tempo** (come input si possono intendere non solo i segnali generati esternamente al microcontrollore, ma anche eventi prodotti dai moduli interni). Il concetto di **'interrupt'** (o 'interruzione') risponde proprio a questa necessità tecnologica. Vediamo in che

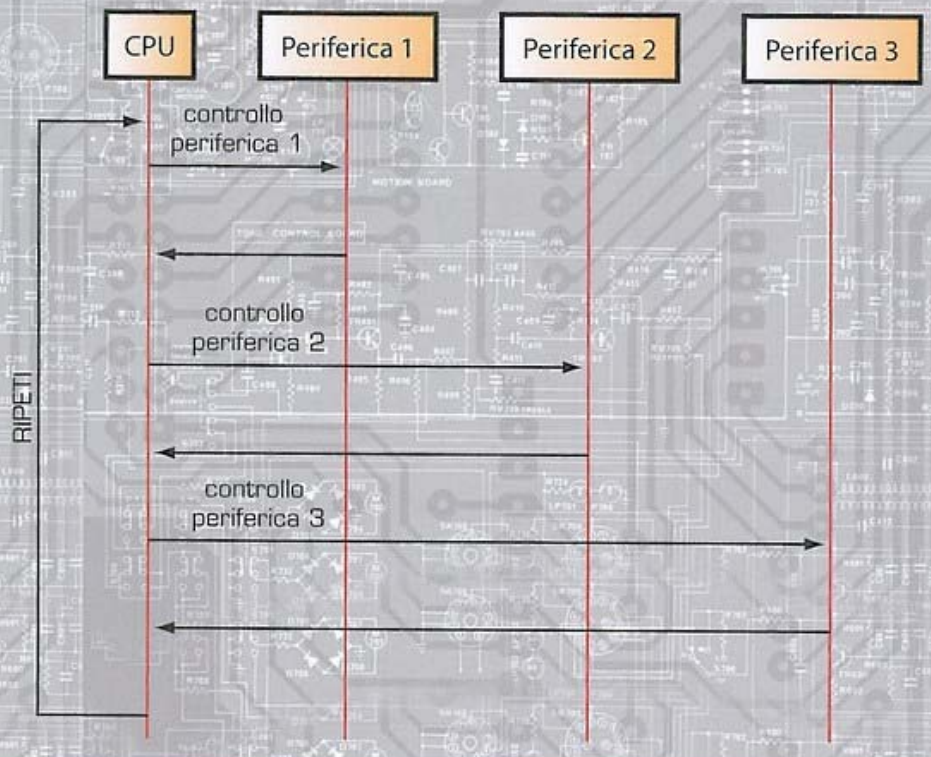


I moderni microprocessori (a lato) permettono di sfruttare gli interrupt per rendere più efficienti i software.

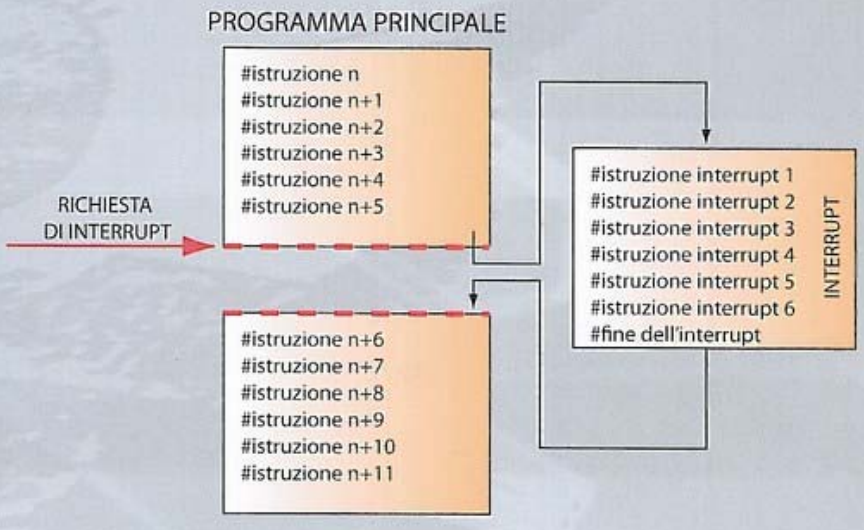
modo attraverso un semplice esempio, legato all'acquisizione di uno o più input digitali. Immaginiamo di avere, come nel Workshop 64, un microcontrollore con tre ingressi digitali a pulsante controllabili da un ipotetico utente. Finora abbiamo effettuato la lettura degli input attraverso un'analisi ciclica degli stati logici dei pin di input con una metodologia di lavoro che in informatica è più nota con il nome di 'polling' (vedi il box sottostante). In quegli esempi era, quindi, il nostro programma che si occupava di effettuare un controllo periodico degli stati di ingresso, lasciando i circuiti esterni completamente passivi rispetto al funzionamento dell'intero sistema. Il meccanismo degli interrupt è, invece, molto differente e consente di rendere realmente attive le periferiche collegate alla CPU (o al 'core' del microcontrollore). Nei sistemi basati sugli interrupt, infatti, ogni periferica può notificare attivamente la necessità di

IL POLLING

In informatica il termine 'polling' viene utilizzato per indicare una particolare metodologia di dialogo tra la CPU e le periferiche che si basa su un controllo sistematico e ciclico degli stati dei singoli dispositivi. La CPU, in pratica, interroga a intervalli più o meno regolari tutti i componenti hardware connessi alla ricerca di periferiche che necessitano dell'invio di dati o che hanno bisogno di trasferire informazioni o richiedere servizi (vedi schema sotto). Pur offrendo alcuni vantaggi sul piano tecnico, questo tipo di approccio alla gestione delle periferiche è, però, obsoleto e poco efficiente. Innanzitutto perché prevede che la CPU effettui sempre e comunque il controllo dello stato dei singoli dispositivi (sia che questi abbiano necessità di dialogare, sia che non la abbiano), sprecando cicli di lavoro e potenza di calcolo. In secondo luogo non vi è la certezza che la risposta a un determinato evento sia immediatamente successiva alla richiesta: se, infatti, il microprocessore sta servendo una determinata periferica e un secondo dispositivo si trova in uno stato di emergenza che richiede un intervento diretto da parte della CPU, questa condizione di emergenza non potrà essere rilevata fino a quando la CPU non sarà giunta a effettuare il controllo sullo stato di quel preciso sistema. Nello schema sottostante è mostrato un esempio di diagramma sequenziale che descrive il ciclo di polling di una ipotetica CPU posta in relazione con tre periferiche (enumerate da 1 a 3).



comunicare con la CPU attraverso un particolare 'segnale', chiamato 'richiesta di interrupt'. Come conseguenza della richiesta di interrupt, la CPU ferma temporaneamente l'esecuzione del programma per richiamare una specifica funzione di risposta, al termine della quale il programma principale viene ripristinato. Come esempio possiamo far riferimento allo schema mostrato a lato, nel quale viene presentato un semplice scenario pratico. In tale schema viene presa in considerazione l'esecuzione di un programma generico a partire dall'ennesima istruzione. Ipotizziamo che durante l'esecuzione numero n+5 un dispositivo produca una richiesta di interrupt per chiedere l'intervento della CPU. A questo punto, il sistema 'congela' temporaneamente il programma principale salvando lo stato del sistema in un'apposita porzione di memoria, per poi avviare una funzione di risposta (blocco INTERRUPT), come se fosse una chiamata a una procedura. Al termine di essa, la CPU riprenderà l'esecuzione del programma principale dall'istruzione n+6. Come è facile intuire, la richiesta di interrupt sarebbe potuta pervenire in un punto qualsiasi del programma principale, ma in ogni caso la risposta alla richiesta sarebbe stata pressoché istantanea, sollevando il ciclo principale dal compito di valutare lo stato delle singole periferiche.



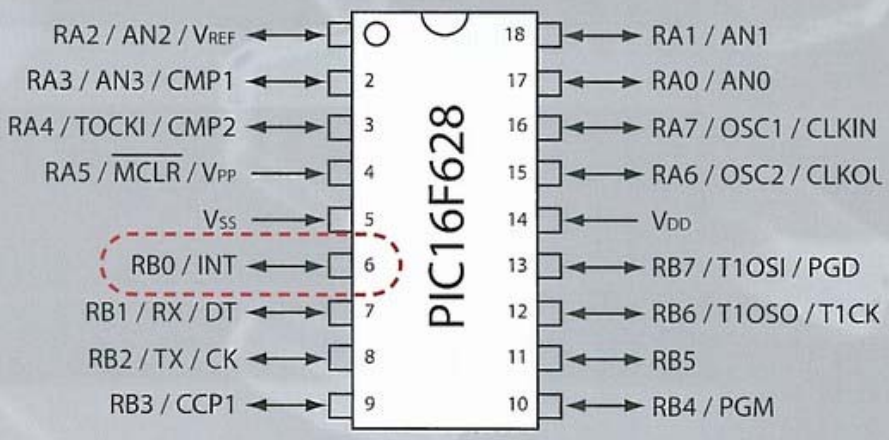
Uno schema che illustra, a livello teorico, il funzionamento degli interrupt. All'arrivo di una richiesta di interrupt, il sistema sospende temporaneamente il programma principale per eseguire una specifica funzione di risposta.

GLI INTERRUPT E IL PIC 16F628 IN MIKROC

Anche il microcontrollore 16F628 è in grado di essere programmato in modo da sfruttare la tecnologia degli interrupt. La sua architettura, infatti, mette a disposizione ben dieci diverse fonti di generazione delle interruzioni, alcune associate a periferiche interne al circuito integrato (come i timer o il modulo di comunicazione seriale), altre pensate, invece, per utilizzare segnali esterni al chip (il pin 6, chiamato RB0/INT, ad esempio, può essere configurato proprio come pin di attivazione della chiamata interrupt). Data la complessità globale dell'argomento e la necessità di conoscere approfonditamente l'architettura interna del 16F628 per poter utilizzare correttamente tutte le tipologie di interruzioni disponibili, nelle pagine del Workshop affronteremo solo alcuni esempi, che avranno lo scopo di far comprendere come operare a livello pratico con il linguaggio mikroC per rispondere ad alcune richieste di interrupt. Per una panoramica più completa delle potenzialità del microcontrollore, come sempre, rimandiamo alla documentazione ufficiale rilasciata da Microchip e facilmente reperibile in Internet. Ma torniamo agli aspetti più pratici della programmazione vedendo come è possibile trattare gli interrupt nell'ambiente di sviluppo mikroC. L'operazione è di per sé molto semplice ed è legata alla dichiarazione di una speciale funzione (chiamata 'interrupt'), che deve essere definita come mostrato qui di seguito:

```
void interrupt(void)
{
  .....
}
```

Tale funzione deve (come tutte le funzioni C) essere dichiarata prima della funzione principale 'main' e deve rispettare la tipizzazione presentata. Abbiamo detto, però, che il 16F628 è in grado di gestire fino a 10 sorgenti diverse di interruzioni, eppure è stata presentata una sola funzione di risposta. Come mai? Questo dipende dal fatto che il PIC in questione non ha più livelli di interrupt, come invece hanno le CPU più complesse, ma è dotato di una sola linea di attivazione, che viene messa a disposizione per tutte le periferiche integrate e non. La funzione di risposta è, quindi, unica e per identificare la sorgente reale della richiesta è necessario controllare lo stato di bit specifici di alcuni registri interni del PIC (non è indispensabile controllarli tutti ogni volta: è sufficiente valutare solo i registri di interesse, poiché le singole sorgenti di input devono essere abilitate esplicitamente con precisi comandi). Ad esempio, se avessimo abilitato l'interrupt associato all'overflow del TIMER0 (vedremo più avanti nel dettaglio di cosa stiamo parlando), implementando la funzione 'interrupt' potremmo identificare TIMER0 come sorgente semplicemente controllando lo stato del bit numero 2 del registro INTCON (bit che prende il nome di T0IF, ossia '*Timer 0 Interrupt Flag*'). Se tale bit ha valore alto, allora il TIMER0 ha generato la richiesta di interrupt; al contrario, se basso significa che non si è ancora verificato l'evento di overflow del timer. Il codice associato alla funzione 'interrupt' diverrebbe, di conseguenza, simile a quello presentato nel box in basso. Nei prossimi Workshop inizieremo a sperimentare il funzionamento degli interrupt proprio con un esempio legato al timer interno numero 0, che impareremo a configurare e utilizzare per generare eventi periodici.



Il pin numero 6 del 16F628 può essere configurato per fare in modo che il microcontrollore avvii una chiamata interrupt in risposta a un segnale digitale esterno.

La funzione speciale 'interrupt'

```
void interrupt(void)
{
   if(INTCON.T0IF == 1)
   {
    /* blocco istruzioni di risposta all'interrupt generato dall'overflow
        del TIMER0*/
    ..........
    ..........
    /* prima di concludere è necessario resettare lo stato del bit T0IF
        portandolo a 0*/
     INTCON.T0IF = 0;
   }

   /* eventuali controlli su altre periferiche */
   ..............
```