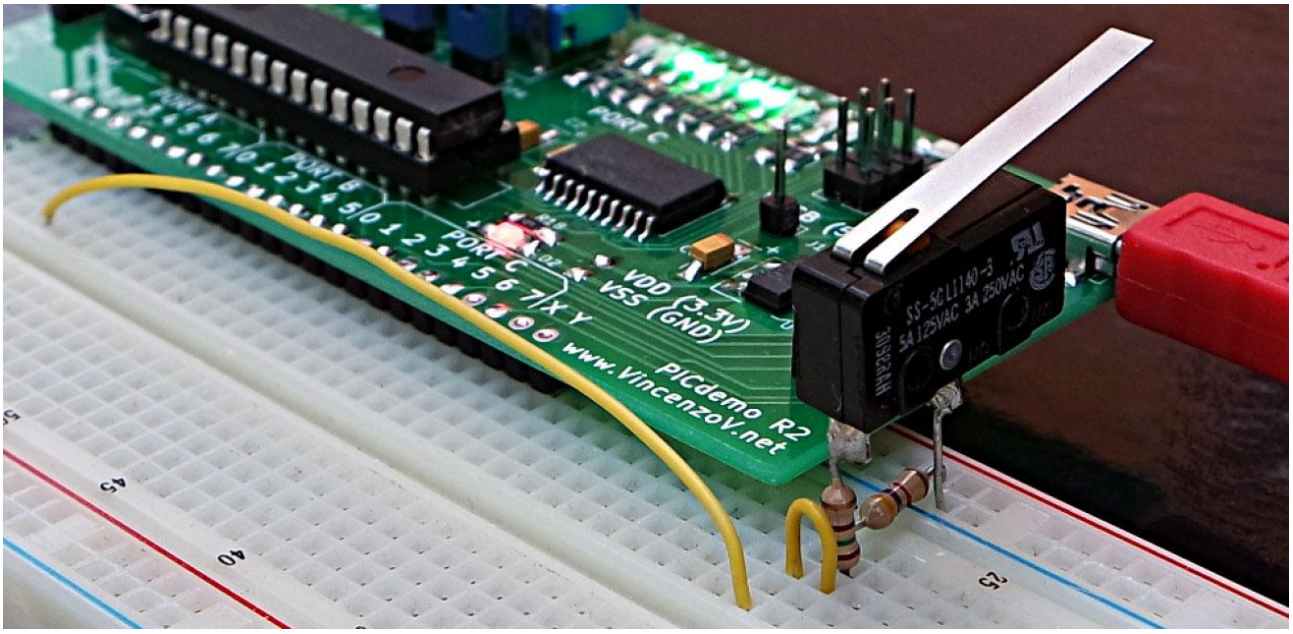
PIC18: interruzioni in assembly



In questa pagina verrà mostrato come gestire in assembly le interruzioni del PIC18 generate da una sorgente esterna, quale un semplice interruttore.

L'interruzione hardware (hardware interrupt) è il modo utilizzato da un generico processore per gestire eventi asincroni, cioè che avvengono indipendentemente dall'esecuzione del programma principale.

Un'analogia con il mondo reale:

1. Sto facendo un lavoro
2. Arriva una telefonata (evento esterno e asincrono rispetto a quello che sto facendo)
3. Sospendo il lavoro per rispondere, memorizzando a che punto del lavoro sono arrivato
4. Quando la telefonata termina, riprendo il lavoro che stavo facendo, da dove lo avevo lasciato

Ovviamente posso decidere di non essere disturbato da telefonate, staccando il telefono; posso anche decidere di ignorare le telefonate non urgenti. O anche decidere se una telefonata urgente può interrompere una telefonata meno urgente. E devo sperare/imporre che la telefonata non sia troppo lunga, altrimenti il lavoro non procede...

Qualche dettaglio più tecnico:

- Il lavoro principale è per esempio svolto dal programma principale
- Il codice eseguito all'arrivo di una interruzione è quello contenuto in una o più funzioni, indicate come ***Interrupt Service Routine*** **- ISR**
- I segnali di interruzione devono poter essere attivati o disattivati in funzione di quanto il codice principale sia più o meno critico e urgente. In particolare devo decidere se è possibile l'interruzione di una isr
- Devo stabilire quali periferiche possono generare interruzioni e quali no, anche in modo dinamico
- Devo riconoscere quale periferica ha generato l'interruzione
- Devo soprattutto essere in grado di riprendere il lavoro principale al termine della isr, ritrovando tutti le variabili come erano all'istante di interruzione. In genere si usa il meccanismo dello *stack* oppure l'uso di due banchi di registri

Questa tecnica permette ad un singolo processore di gestire (quasi) contemporaneamente più attività:

- Una principale, codificata nel programma principale
- Una o più eseguite direttamente su richiesta di una periferica. Il codice corrispondente è quello presente nella funzione (o nelle funzioni) ISR

Al solito, è vivamente consigliata la lettura in parallelo della spiegazione semplificata qui riportata e della la spiegazione "ufficiale" presente nei data sheets.

Il PIC18 prevede due distinte modalità di uso delle interruzioni:
- La modalità attiva all'accensione è indicata come **Mid-Range Compatibility mode** che, è compatibile con quella presente nei PIC16.
- La modalità tipica del PIC18, **Interrupt Priority mode**, utilizza due livelli di priorità per le interruzioni e deve essere attivata esplicitamente dal software.
Di seguito verranno esaminate entrambe le modalità anche se la prima ha valenza soprattutto per chi vuole un approccio più semplice.

Compatibility mode

Questa modalità è attiva all'accensione e quindi per attivarla non occorre fare nulla; è associata al valore zero del flag Interrupt Priority ENable bit (IPEN=0).

Il codice di esempio fa scorrere di una posizione un LED su PORTC quando viene premuto un pulsante, in corrispondenza del fronte di salita; per fare ciò vengono utilizzate le interruzioni generate da un segnale esterno applicato direttamente al pin RB0/INT0.

Per la sua gestione vengono utilizzati tre flag del registro **INTCON** (INTERRUPT CONTROL REGISTER):

Global Interrupt Enable bit (bit 7): quando settato, abilita il processore alla ricezione di tutte le interruzioni.

INT0 External Interrupt Enable bit (bit 4): quando settato, permette a PORTB di generare un interrupt sul fronte di salita del pin fisico INT0/RB0

INT0IF: INT0 External Interrupt Flag bit (bit 1): quando INT0/RB0 rileva un fronte di salita, questo flag viene settato dall'hardware. Possiamo quindi dire che, quando il suo valore è 1, è stata generata una interruzione da INT0. Un'osservazione importante: tale bit deve essere riportato esplicitamente a 0 dal software dopo che sono state eseguite le istruzioni relative alla gestione della interruzione.

Programma:

Inizializzazione

Occorre evidentemente come prima cosa configurare PORTC e PORTB rispettivamente come uscita e come ingresso con pull-up interno per collegare un interruttore

```
clrf    TRISC           ; PORTC come uscita, tutti gli 8 pin
movlw   B'00000001'     ; LED0 acceso; LED1-7 spenti
movwf   LATC
movlw   0xFF            ; PORTB come ingresso, tutti gli 8 pin
movwf   TRISB
bsf     WPUB, INT0      ; Abilita il pull-up interno su INT0
bcf     INTCON2, RBPU   ; Pull-up attivo su tutti i pin abilitati
```

Occorre quindi procedere alla abilitazione delle interruzioni, impostando il flag relativo a INT0 (INT0 Interrupt Enable) e quello globale (Global Interrupt Enable), relativa all'intero processore:

```
bsf     INTCON, INT0IE  ; Abilito INT0 a generare interruzioni
bsf     INTCON, GIE     ; Abilito la CPU a ricevere interruzioni
```

Superloop

Dopo le configurazioni iniziali, il processore inizia un ciclo infinito dove non vene fatto assolutamente nulla di significativo. In particolare il codice NON legge il valore dell'interruttore e NON imposta l'accensione dei LED.

```
repeat
        nop                                     ; non fa nulla (No OPeration
        bra             repeat
```

ISR

La Interrupt Service Routine è il codice che viene eseguito quando l'hardware rileva la generazione di un interrupt. Alcune operazioni vengono fatte automaticamente quando l'interrupt è generata e quindi non corrispondono ad alcuna istruzione:

- Il *Program Counter* corrente viene salvato nello *stack* hardware del PIC18. Questa operazione è necessaria per permettere al PIC di riprendere l'esecuzione del superloop esattamente nel punto in cui era arrivato nel momento di generazione della interruzione.

- I registri WREG, STATUS e BSR vengono copiati nel *Fast Register Stack*, a volte indicato some *shadow register*. Questo rende possibile ripristinare tali registri al termine della ISR

- Le *interruzioni vengono automaticamente disabilitate* per impedire che un nuovo interrupt possa interrompere una ISR già in esecuzione.

- Viene *eseguito il codice ISR* che, obbligatoriamente, si trova all'indirizzo 0x0008

In genere il codice della ISR inizia con la ricerca del dispositivo hardware ha effettivamente generato l'interruzione. Nell'esempio solo INT0 è stata abilitata e quindi una interruzione è sicuramente generata da essa. In una applicazione reale invece sono molte le sorgenti di interrupt e quindi l'unica ISR deve esaminate tutti i vari flag alla ricerca di quello attivo.

Inoltre, obbligatoriamente, la ISR deve azzerare in modo esplicito il flag dalla periferica corrispondente all'interruzione generata. Saltando tale passaggio infatti, subito dopo la conclusione della ISR, verrebbe generata una nuova interruzione, evidentemente non esistente, in una ripetizione infinita.

Nell'esempio:

```
        bcf         INTCON, INT0IF      ; Azzera il flag che segnala INT0
```

La ISR termina infine sempre con l'istruzione retfie, a volte seguita da FAST. L'istruzione ha i seguenti effetti:

- Toglie dallo stack l'indirizzo di ritorno e quindi permette di riprendere l'esecuzione del superloop nel punto in cui era stato interrotto

- Abilita le interruzioni (disabilitate temporaneamente in modo automatico)

- Ripristina i shadow register (solo se seguita dall'opzione FAST)

Di seguito l'intera struttura di una ISR tipica dove vengono esaminate a titolo di esempio tre possibili sorgenti di interrupt (INT0, Timer0, cambiamento di RB), osservando i corrispondenti flag ed eseguendo di conseguenza parti specifiche di codice (non tutte implementate):

```
ISR CODE 0x0008
        btfsc       INTCON, INT0IF      ; Interruzione causata da INT0 ?
        bra         GestioneINT0        ; Esegue il codice corrispondente
        btfsc       INTCON, TMR0IF      ; Interruzione da Timer0 ?
        bra         GestioneTimer0
        btfsc       INTCON, RBIF        ; Interruzione da un bit di RB ?
        bra         GestioneRB0
        retfie      FAST                ; Nessuna interrupt riconosciuta...
        Indizio di un errore?
GestioneINT0
        rlncf       LATC                ; "Ruota" i LED
        bcf         INTCON, INT0IF      ; Azzera il flag che segnala INT0
        retfie      FAST                ; Torna al codice principale
GestioneTimer0                          ; Non implementato
        bcf         INTCON, TMR0IF
        retfie      FAST
GestioneRB0                             ; Non implementato
        bcf         INTCON, RBIF
        retfie      FAST
```

Altri esempi di interruzione, generati da specifiche periferiche come: Timer0, EUSART

Latenza

La latenza, in questo contesto, misura il tempo necessario al PIC18 per "rispondere" ad una richiesta di interruzione. Dipende da diversi fattori:

- Il clock del processore. Ovviamente aumentandolo, diminuisce in modo rigorosamente proporzionale il tempo di esecuzione

- Il tipo di interruzione. Le interruzioni asincrone (come INT0) hanno una variabilità casuale di ± il tempo di esecuzione di una istruzione

- La priorità della ISR (quando attivate). Infatti solo quelle ad alta priorità possono sfruttare i shadow register e quindi risparmiano il tempo necessario all'eventuale salvataggio temporaneo dei registri

Priority mode

Quando le interruzioni del PIC18 sono configurate in ***Priority mode*** è possibile definire due *classi di priorità*:

- Interruzioni a bassa priorità
- Interruzioni ad alta priorità

La differenza essenziale tra le due è che le seconde possono interrompere l'esecuzione delle prime, ma non viceversa.

La priorità delle interruzioni generate da una specifica periferica è in genere programmabile, a seconda delle esigenze.

Il codice dell'esempio che utilizza un interrupt ad alta priorità ed uno a bassa priorità ha le seguenti particolarità:

- INT0, configurato sempre e automaticamente ad alta priorità.
- INT1, configurato in questo esempio a bassa priorità

Per la gestione vengono utilizzati diversi flag, distribuiti su vari registri:

- Un flag per attivare la modalità di funzionamento con priorità (non attiva all'accensione)
- Un flag per attivare globalmente le interruzioni a bassa priorità
- Un secondo flag per attivare globalmente le interruzioni ad alta priorità (se sono disattivate le interrupt ad alta priorità, sono automaticamente disattivate anche quelle a bassa priorità)
- Un flag per ciascuna periferica per attivare la generazione di interrupt
- Un flag per ciascuna periferica per specificare se l'interrupt generata sarà ad alta oppure a bassa priorità (se questa possibilità è prevista)
- Un flag per ciascuna periferica per segnalare l'avvenuta generazione dell'interrupt

Inizializzazione

Il codice di inizializzazione delle porte è sostanzialmente identico a quello già visto per il *Compatibility mode* (senza priorità), con la sola aggiunta della configurazione di INT1 come ingresso:

```
 clrf TRISC ; PORTC come uscita, tutti gli 8 pin
 movlw 0x01 ; LED0 acceso; LED1-7 spenti
 movwf LATC

 movlw 0xFF ; PORTB come ingresso, tutti gli 8 pin
 movwf TRISB
```

Occorre quindi procedere alla configurazioni delle interruzioni, impostando:

- la modalità delle interruzioni con priorità (Interrupt Priority ENable bit)
- i flag di abilitazione relativi a INT0 (INT0 Interrupt Enable) e INT1 (INT1 Interrupt Enable)
- la priorità dell'interruzione generata da INT1
- la ricezione da parte della CPU delle interruzioni ad alta priorità (Global Interrupt Enable High)
- la ricezione da parte della CPU delle interruzioni a bassa priorità (Global Interrupt Enable Low)

Il codice corrispondente:

```
 bsf RCON, IPEN ; Abilito le interruzioni a livelli di
priorità
 bsf INTCON, INT0IE ; Abilito INT0 a generare interruzioni
(ad alta priorità)
 bcf INTCON3, INT1IP ; Imposto INT1 a bassa priorità
 bsf INTCON3, INT1IE ; Abilito INT1 a generare interruzioni
 bsf INTCON, GIEH ; Abilito la CPU a ricevere interruzioni
ad alta priorità
 bsf INTCON, GIEL ; Abilito la CPU a ricevere interruzioni
a bassa priorità
```

ISR ad alta priorità

Questa ISR è il codice che viene eseguito quando l'hardware rileva la generazione di un interrupt ad alta priorità. La descrizione è identica a quanto già scritto a proposito delle interruzioni senza priorità. In particolare non cambia l'indirizzo in cui il codice deve essere memorizzato (`0x0008`) e la modalità di salvataggio e recupero dei registri nei *registri shadow*.

Il codice:

```
ISR_Alta CODE 0x0008 ; Vettore per ISR ad alta priorità
 goto AltaPriorita ; Salta all'ISR ad alta priorità


AltaPriorita
 btfsc INTCON, INT0IF ; Interruzione causata da INT0 ?
 bra GestioneINT0 ; Esegue il codice corrispondente

 retfie FAST ; Nessuna fleg settato... Indizio di un
errore?

GestioneINT0
 ; qui il codice
 bcf INTCON, INT0IF ; Azzera il flag che segnala INT0
 retfie FAST ; Torna ad eseguire il codice principale
```

L'unica cosa che è bene mettere in evidenza è che questo codice non può essere mai interrotto.

ISR a bassa priorità

Due sono le differenze fondamentali rispetto alla ISR ad alta priorità:

- Il codice della ISR deve essere memorizzato a partire dall'indirizzo `0x0018`
- Non è possibile l'uso dei *registri shadow* per salvare temporaneamente i registri `WREG`, `STATUS` e `BSR`. Infatti il codice di questa ISR potrebbe essere a sua volta interrotto da un interrupt ad alta priorità, con l'effetto di sovrascrivere i *registri shadow*, perdendone il contenuto. Il modo più semplice per salvaguardare questi registri è l'uso di tre variabili temporanee. Una alternativa, più lenta e complessa, è l'uso di uno **stack software** che però avrebbe il vantaggio di poter scrivere interrupt pre-emptive.

Il codice:

```
ISR_Bassa CODE 0x0018 ; Vettore per ISR a bassa priorità
 goto BassaPriorita ; Salta all'ISR bassa priorità

BassaPriorita
 movff WREG, WREG_TEMP ; salva il registro W nella
variabile "temporanea"
 movff STATUS, STATUS_TEMP ; salva i flag nella variabile
"temporanea"
 movff BSR, BSR_TEMP ; salva il registro BSR nella
variabile "temporanea"

 btfsc INTCON3, INT1IF ; Interruzione causata da INT1 ?
 bra GestioneINT1 ; Esegue il codice corrispondente

 bra Ritorna ; Nessuna fleg settato... Indizio di un
errore?

GestioneINT1
 ; qui il codice
 bra fine_int_Bassa

fine_int_Bassa
 movff STATUS_TEMP, STATUS ; ripristina i flags
 movff WREG_TEMP, WREG ; ripristina il registro W
 movff BSR_TEMP, BSR ; ripristina il registro BSR
 bcf  INTCON3, INT1IF ; Azzera il flag che segnala INT1
retfie
```