**La gestione degli interrupt nel PIC16F877A**

Il PIC16F877A ha 3 possibili sorgenti di interrupt, comuni a tutti i microcontrollori della famiglia PIC16, più tutti quelli che possono essere generati dai periferici specifici.

Quelle comuni a tutta la famiglia sono:

Interrupt esterni dal pin RB0/INT.

Interrupt su cambiamento di livello su almeno uno degli ingressi RB7:RB4.

Interrupt su overflow del counter/timer TMR0.

Possono poi generare interrupt il convertitore A/D, la porta seriale, i timer 1 e 2, i moduli Compare/Capture ecc., per un totale complessivo di 14 sorgenti.

Non esistono interrupt non mascherabili: tutti gli interrupt sono “filtrati” da un flag di abilitazione generale degli interrupt GIE (bit 7 del registro INTCON), che di default è basso (interrupt disabilitati).

La logica degli eventi legati a un’interrupt è la seguente: ogni periferica che può chiedere interruzione ha un “flag d’interruzione” che si alza quando si verificano le condizioni previste per l’interrupt; **la CPU testa i flag d’interruzione dei periferici ogni ciclo Q1 del clock, cioè all’inizio del ciclo macchina**



Se ne trova uno alto (e se è *quel* periferico è abilitato a generare interrupt), l’indirizzo di rientro viene salvato sullo stack hardware, e il PC (Program Pounter) caricato con 0004h, locazione dove di conseguenza la CPU salta, con un ritardo (tempo di latenza) di 2 – 3 cicli macchina. In 0004h deve iniziare la routine di gestione, o un salto alla routine di gestione.

Il **vettore d’interruzione**è pertanto unico: è il software che può (di solito *deve*) individuare quale periferica tra quelle abilitate ha chiesto interrupt interrogando (polling) i loro flag d’interruzione.

Quando il PIC risponde a un’interruzione, *ulteriori interrupt vengono disabilitati* dall’abbassamento automatico di GIE.

E’ possibile, anche se non molto raccomandabile, permettere la nidificazione degli interrupt, rialzando GIE (che è scrivibile come tutti i flag di INTCON) all’interno della routine di gestione.

Al rientro dall’interruzione il flag GIE si rialza automaticamente, riabilitando le interruzioni. (consiglio di controllare)

***E’ assolutamente importante ricordare che il flag d’interruzione della periferica non si riabbassa automaticamente*: poiché il flag alzato che causa interrupt, ci si deve ricordare di abbassarlo all’interno della routine di gestione – di solito alla fine della routine – per impedire che al rientro l’interrupt riparta immediatamente, innescando un loop infinito.**

Il meccanismo degli interrupt non è semplicissimo da gestire; per capirlo meglio si può vedere la Figura 10, che lo traduce in uno schema logico:



**La richiesta di interrupt alla CPU è un livello alto all’uscita dell’ultimo AND a destra, da cui si vede che è inibita da GIE = 0**

Ogni periferico ha un flag d’interruzione, che si alza comunque sempre quando si verificano le condizioni che potrebbero chiedere interrupt, e un flag di abilitazione specifico. A titolo d’esempio, consideriamo gli interrupt dalla linea RB0/INT: il flag d’interruzione è INTF e quello d’abilitazione è INTE. Dallo schema si vede che perché la CPU “senta” l’innalzamento di INTF, non basta che sia alto GIE, ma deve essere alto anche INTE: i flag di abilitazione specifici dei periferici permettono quindi di selezionare quali di essi possono chiedere interrupt.

La richiesta d’interrupt da parte dei periferici specifici dell’877A è generata dalla combinazione ANDs – OR a sinistra, e si vede che perché tale richiesta “passi” è necessaria un’ulteriore abilitazione rappresentata dal flag PEIE (PEripheral Interrupt Enable). Nel registro principale collegato agli interrupt – INTCON – ci sono i flag di interrupt e abilitazione interrupt delle tre sorgenti di base, oltre a GIE e PEIE



Di INTF e INTE abbiamo già detto. TMR0IF/TMR0IE è la coppia flag d’interrupt/flag d’abilitazione interrupt relativa al Timer 0, RBIF/RBIE quella relativa all’interrupt da cambiamento su RB7:RB4. Le coppie relative agli altri periferici sono sparse in altri registri specializzati. Vedere la documentazione relativa ai singoli periferici. Notare che anche INTCON è mappato in tutti i banchi di memoria.

Riassumendo, la programmazione degli interrupt richiede i seguenti passi:

In fase di inizializzazione:

Alzare il flag di abilitazione interruzione della periferica e, se la periferica è esterna al core, anche PEIE

Se non si è sicuri dello stato del flag d’interruzione della periferica, abbassarlo per sicurezza

3. Alzare il flag di abilitazione generale GIE

Nella routine d’interruzione, che deve iniziare in 0x0004:

1. Salvare STATUS e W in una coppia di registri opportunamente definiti
2. Se le sorgenti d’interruzione possono essere più d’una, controllare i flag d’interruzione per stabilire quale periferico ha chiamato, ed eseguire di conseguenza le operazioni richieste dalla gestione del periferico stesso (se il periferico che può chiamare interruzione è uno solo, il controllo sui flag necessario)
3. Abbassare il flag d’interruzione del perififerico chiamante
4. Recuperare STATUS e W
5. Rientrare dall’interrupt (istruzione **RETFIE**)

Ecco un esempio abbastanza generale e completo di gestore di interrupt, che può essere facilmente adattato ad esigenze specifiche. Si suppone che possano chiamare interrupt il Timer 0 (flag d’interruzione TMR0IF) e l’ingresso RB0/INT (flag d’interruzione INTF). Le routine di servizio dei due interrupt sono rispettivamente **sbr\_1** e **sbr\_2** (che devono terminare con l’istruzione di rientro 3 Ciò non è in contraddizione con il fatto che GIE è basso e inibisce uleriori interrupt: la CPU *è già in interrupt*, quando legge il secondo flag d’interruzione

da subroutine **RETURN**). Ricordare che il tutto deve essere allocato in 0x004 (mediante una direttiva **org 0x004**):

interr MOVWF W\_TEMP ; Salva w

SWAPF STATUS,W ; e lo stato

MOVWF STATUS\_TEMP ; (il perché di questa complicazione lo si

; capisce al rientro)

BTFSC INTCON,TMR0IF ; Se è interrupt da Timer 0

CALL sbr\_1 ; chiama sbr\_1 e rientra

BTFSC INTCON,INTF ; Se è (anche) interrupt da RB0/INT

CALL sbr\_2 ; chiama sbr\_2 e rientra

BCF INTCON,TMR0IF ; Abbassa i

BCF INTCON,INTF ; flag d'interruzione (in realtà solo

; uno sarà di solito alto)

SWAPF STATUS\_TEMP,W ; Recupera

MOVWF STATUS ; lo stato

SWAPF W\_TEMP,F ; e poi W (con una finezza per evitare di

; alterare il flag Z,

SWAPF W\_TEMP,W ; che è influenzato da movf, ma non da swapf)

RETFIE ; Ritorna dall'interrupt

Al solito fare riferimento a **Descrizione delle istruzioni** per i particolari. Ricordare in ogni caso che **BTFSC** *scavalca* l’istruzione successiva se il bit specificato del registro specificato è basso, e che **CALL** è la chiamata a subroutine. Così il segmento di programma

BTFSC INTCON,TMR0IF

CALL sbr\_1

scavalca l’istruzione CALL sbr\_1 se il flag TMR0IF del registro INTCON è basso, *cioè Timer 0 non ha chiamato interrupt*; se invece il flag è alto, cioè è Timer 0 ad aver chiamato, l’istruzione di chiamata viene eseguita e la CPU salta ad eseguire sbr\_1.

Notare che il sistema permette di gestire interrupt che si sovrappongano, stabilendo fra essi una scala di priorità: se, mentre la CPU sta rispondendo all’interrupt ad Timer 0, si verifica la condizione per un interrupt sull’ingresso RB0/INT, il flag INTF si alza (indipendentemente da ogni abilitazione), di modo che, al rientro da sbr\_1, il test

BTFSC INTCON, INTF

lo trova alto e chiama sbr\_23. Notare che l’inverso non avviene: se, durante l’esecuzione di sbr\_2 in risposta all’interrupt da RB0/INT, si verifica una richiesta da Timer 0, questa viene ignorata, perché TMR0IF non viene più interrogato. Se si vuole escludere la possibilità di sovrapposizione, si può modificare il gestore così:

interr MOVWF W\_TEMP ; Salva w

SWAPF STATUS,W ; e lo stato

MOVWF STATUS\_TEMP ; (il perché di questa complicazione lo si

; capisce al rientro)

BTFSC INTCON,TMR0IF ; Se è interrupt da Timer 0

GOTO sbr\_1 ; chiama sbr\_1

BTFSC INTCON,INTF ; Se è interrupt da RB0/INT

GOTO sbr\_2 ; chiama sbr\_2

ret BCF INTCON,TMR0IF ; Qui al rientro da sbr\_1 o sbr\_2: abbassa i

BCF INTCON,INTF ; flag d'interruzione (in realtà solo

; uno sarà di solito alto)

SWAPF STATUS\_TEMP,W ; Recupera

MOVWF STATUS ; lo stato

SWAPF W\_TEMP,F ; e poi W (con una finezza per evitare di

; alterare il flag Z,

SWAPF W\_TEMP,W ; che è influenzato da movf, ma non da swapf)

RETFIE ; Ritorna dall'interrupt

16

Qui sbr\_1 e sbr\_2 vengono chiamate da un salto incondizionato **GOTO**, e devono terminare con **GOTO ret**, in modo da rientrare entrambe nello stesso punto, senza la possibilità che vengano eseguite entrambe.

Una spiegazione a parte richiede lo strano modo di salvare W e STATUS, e di recuperarli al rientro. Anzitutto, i registri W\_TEMP e STATUS\_TEMP è bene che vadano definiti nella fascia di indirizzi 0x70 – 0x7F, che si “specchia” in tutti i quattro blocchi RAM, in modo da non doverci preoccupare di che banco è indirizzato quando si verifica l’interrupt. In secondo luogo, la soluzione che parrebbe ovvia:

MOVWF W\_TEMP ; Salvataggio

MOVF STATUS,W

MOVWF STATUS\_TEMP

;

;

;

;

MOVF STATUS\_TEMP,W ; Recupero

MOVWF STATUS

MOVF W\_TEMP,W

non va bene, perché le istruzioni **MOVF STATUS\_TEMP** e **W MOVF W\_TEMP,W** toccano il flag di zero (vedi Descrizione delle istruzioni) e quindi possono alterare lo stato ripristinato (il flag di zero fa parte di STATUS). L’unica istruzione che può copiare un registro RAM in W senza alterare i flag è **SWAP** (con destinazione W), che però lo fa invertendo i nibble (blocchi di 4 bit) alto e basso: basta allora usarla complessivamente due volte per ogni salvataggio/recupero. La sequenza corretta è allora4:

MOVWF W\_TEMP ; Salvataggio

SWAPF STATUS,W

MOVWF STATUS\_TEMP

;

;

;

;

SWAPF STATUS\_TEMP,W ; Recupero

MOVWF STATUS

SWAPF W\_TEMP,F

SWAPF W\_TEMP,W

Un’ultima precauzione riguarda la disabilitazione generale degli interrupt da programma: la soluzione ovvia **BCF INTCON,GIE** è rischiosa; se interviene un interrupt durante l’esecuzione di questa istruzione, questa viene terminata prima del salto a 0x0004. A questo punto GIE è basso, ma la CPU *ha già accettato l’interrupt* (nella prima fase di TCY), per cui salta alla routine di gestione, *al rientro della quale GIE viene alzato di nuovo*. Si va sul sicuro controllando che effettivamente GIE sia andato basso ed eventualmente riprovandoci:

riprova BCF INTCON,GIE

BTFSC INTCON,GIE

GOTO riprova