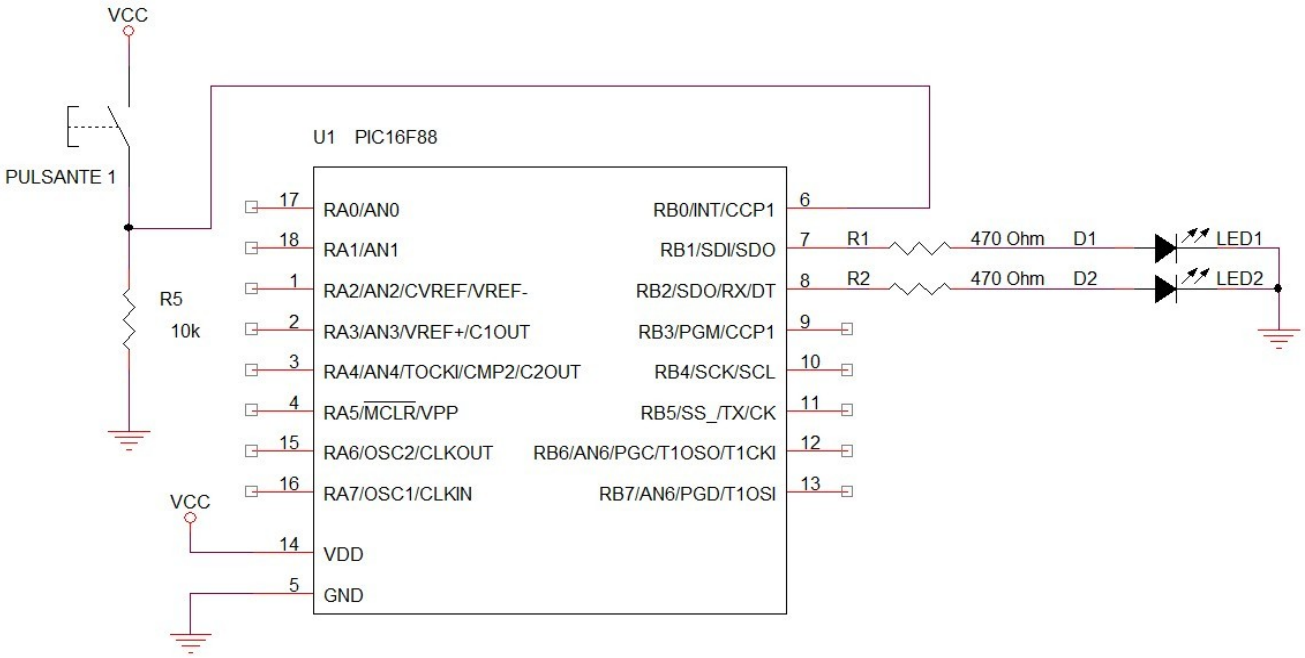
INTERRUPT NEI MICROCONTROLLORI

Supponiamo di stare comodamente seduti sul divano a leggere un libro, e nello stesso tempo siamo in attesa di un pacco di un corriere.
La lettura del libro richiede la nostra attenzione, e sequenzialmente sfogliamo ogni pagina del libro.
Anche l'arrivo del corriere richiederà la nostra attenzione, ed al suo arrivo non potremo continuare a leggere il libro ma dovremo aprire la porta firmare la ricevuta del pacco e tornare alla nostra attività.

Abbiamo essenzialmente I due seguenti modi di procedere:

- Ogni intervallo da me definito, potrebbe essere ad esempio ogni pagina del libro, mi alzo e vado alla porta a controllare l'arrivo del corriere, ed in caso del suo arrivo firmo la ricevuta e torno alla mia attività di lettura. Questa è una modalità "sincrona" richiede cioè un controllo periodico della porta che avviene in tempi ben definiti.
- Continuo a leggere il libro fino a quando non sento suonare il campanello, a quel punto metto un segnalibro sulla pagina che sto leggendo, mi alzo per firmare la ricevuta e torno come prima alla lettura del libro. In questo caso la modalità non è sincrona, ma asincrona in quanto non c'è nessun intervento periodico da fare, ma un'operazione da effettuare quando viene segnalata la presenza di un corriere alla porta tramite il campanello, che richiede la nostra attenzione.

Ora spostiamo la situazione in un circuito con un microcontollore che esegue una funzione come ad esempio il lampeggio di un led (LED1) ma allo stesso tempo deve verificare se qualcuno preme un pulsante per accendere o spegnere un secondo led (LED2).



Anche in questo caso il microcontrollore, può eseguire due tipologie di programmi che effettuano lo stesso procedimento dell'esempio precedente.

- Accendo e spengo il LED1 ed ad ogni spegnimento, controllo il PULSANTE e modifico lo stato del LED2.
  Tradotto in linguaggio C avremo la situazione di lato. Questo metodo viene definito con il termine **POLLING**.

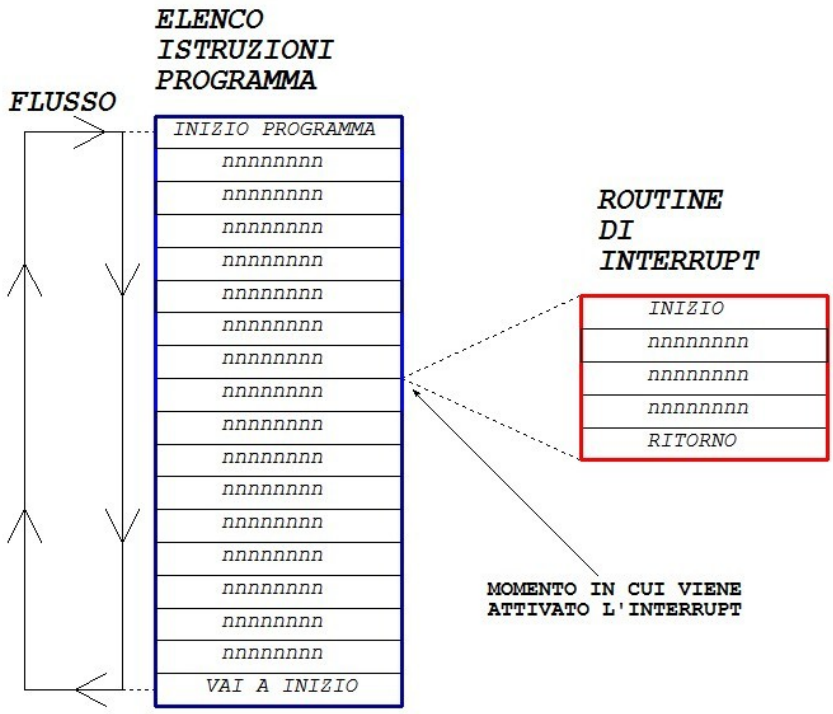
```
sbit LED1       at RB1_bit;
sbit LED2       at RB2_bit;
sbit PULSANTE   at RB0_bit;

void main(){
  while(1){
    LED1=1;           //accendo LED1
    delay_ms(500);    //attesa 0,5sec
    LED1=0;           //spengo LED1
    delay_ms(500);    //attesa 0,5sec
    if(PULSANTE==1) //controllo il pulsante
     {  //spengo o accendo il LED2
          if(LED2==0) LED2=1;
              else            LED2=0;
     }//if
   }//while
}
```

  Con il metodo del polling però ci sono diversi problemi, uno di questi ad esempio, è dovuto proprio al fatto che il controllo viene fatto in maniera sincrona ogni intervallo di tempo ben definito, ciò significa che il microcontrollore potrebbe non accorgersi della pressione del pulsante se questo dovesse avvenire durante l'intervallo di tempo in cui non viene testato.
  Potremmo controllare il pulsante più spesso, magari spezzettando il delay in durate più piccole (500 da 1msec) ma il problema pur se ridotto rimarrebbe. Inoltre il programma potrebbe diventare molto più complesso. Un altro problema è legato al fatto che potrei avere un programma con diversi cicli e situazioni dove non potremmo permetterci il lusso di "perdere tempo" per testare un ingresso.
- La seconda soluzione, cioè quella asincrona, risolverebbe tutti questi problemi, in quanto il microcontrollore svolgerebbe le funzioni previste nel programma principale, ed in ogni momento se qualcuno "suona il campanello", il programma verrebbe momentaneamente interrotto per svolgere la funzione prevista dal pulsante. Per attivare questa seconda modalità dobbiamo ricorrere all'utilizzo dell'INTERRUPT.

L'interrupt hardware è un segnale asincrono, proveniente da una o più periferiche del microcontrollore, per richiedere lo svolgimento di una parte di codice di programma.

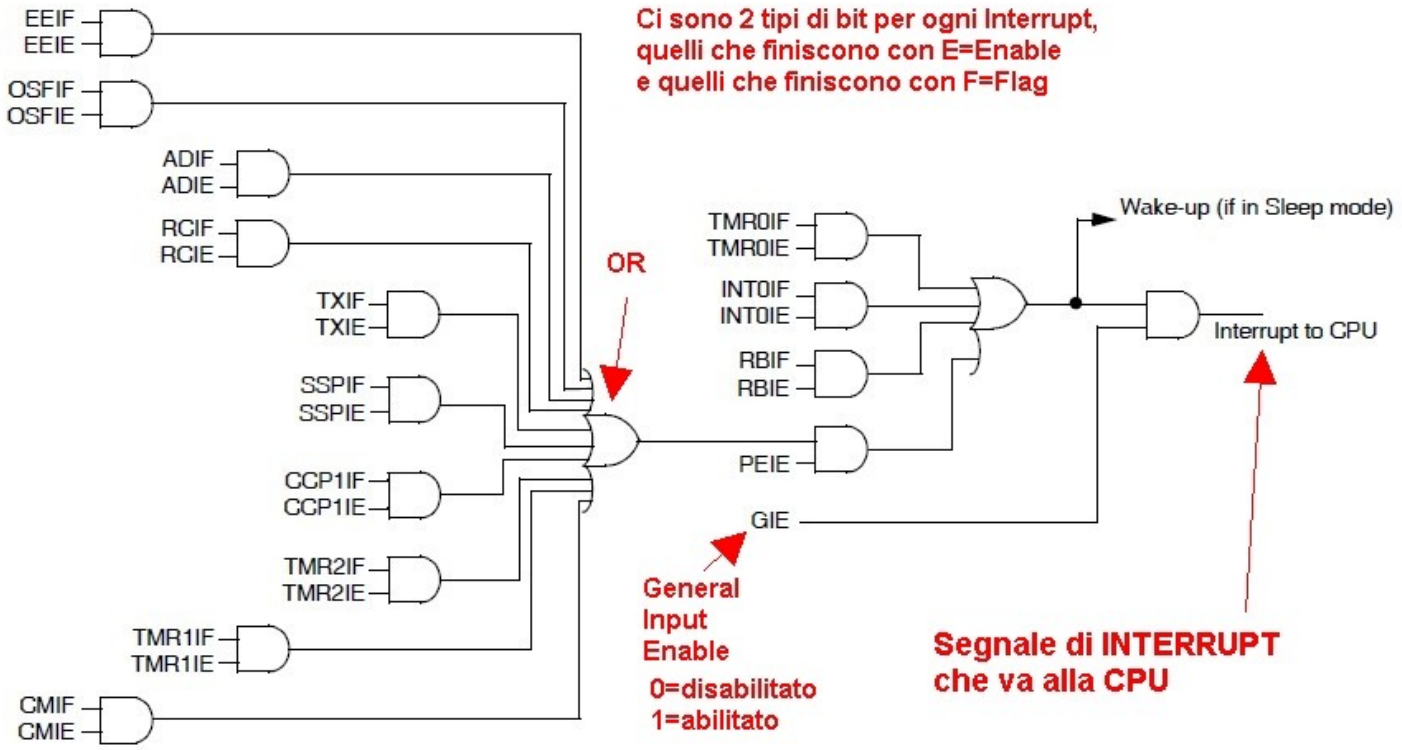
Solitamente quando il microcontrollore rivece la richiesta di interrupt verifica quale periferica lo ha attivato e svolge la funzione relativa a quella periferica per poi riprendere il flusso normale del programma.



Esistono anche gli interrupt software e cioè delle chiamate a delle subroutine che avvengono con la stessa modalità dell'interrupt hardware.

Tornando però agli interrupt hardware guardiamo ad esempio come funzionano gli interrupt nel microcontrollore della Microchip PIC16F88.

La seguente immagine è tratta dal DataSheet del microcontrollore PIC16F88, nella sezione Interrupt.



In questa immagine è racchiuso tutta la parte hardware del segnale di Interrupt.

Il segnale che va alla CPU può infatti scaturire da uno dei tanti segnali che finiscono con la lettera F (Flag) e che sono relativi alle periferiche interne o esterne del microcontrollore.

Più esattamente:

- **EEIF** EEPROM Write Operation Interrupt Flag bit =1 al termine di una scrittura nella EEPROM.
- **ADIF** A/D Converter Interrupt Flag bit =1 al termine di una conversione Analogico Digitale.
- **RCIF** AUSART Receive Interrupt Flag bit =1 quando il buffer di ricezione seriale RS232 è pieno.
- **SSPIF** Synchronous Serial Port (SSP) Interrupt Flag bit =1 al termine della trasmissione su I2C o SPI.
- **CCP1IF** Capture Compare PWM Interrupt Flag bit =1 al termine dell'operazione "compare" o "capture".
- **TMR2IF** TMR2 to PR2 Interrupt Flag bit =1 quando il registro TMR2 è uguale al registro PR2 (TIMER2)
- **TMR1IF** TMR1 Overflow Interrupt Flag bit =1 quando il registro del Timer1 va in overflow
- **CMIF** Comparator Interrupt Flag bit =1 quando c'è una variazione dell'uscita dei comparatori.

I bit sopra menzionati sono accessibili nei registri PIR1 e PIR2 (**P**eripheral **I**nterrupt **R**egister) questi bit dicono se si è verificato uno degli eventi ad essi associati. Come si può vedere dall'immagine tutti I bit vanno in ingresso di una porta AND che ha nell'altro ingresso il corrispondente bit Enable, come: EEIE, ADIE, RCIE, SSPIE, CCP1IE, TMR2IE, TMR1IE, CMIE. Questi ultimi bit sono accessibili nei registri PIE1 e PIE2.

In pratica se il bit Enable vale 0, allora l'eventuale attivazione del Flag di Interrupt non passa all'uscita della porta AND, se invece il bit Enable vale 1 allora il Flag passa in uscita della porta fino ad arrivare alla porta OR

evidenziata nel disegno, essendo una porta OR, il segnale passerà in uscita per essere ancora filtrato dal bit PEIE cioè il Peripheral Interrupt Enable bit che abillita perciò il passaggio di un eventuale flag di interrupt delle periferiche.

Allo stesso modo dopo la porta OR troviamo altri bit di Flag e di Enable e cioè:

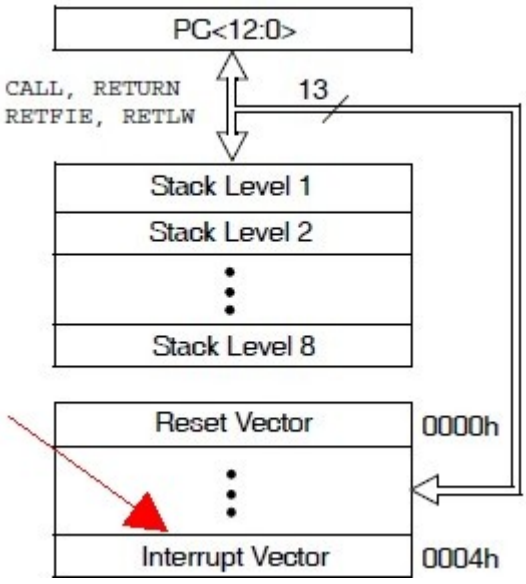
- **TMR10F** TMR0 Overflow Interrupt Flag bit =1 quando il registro del Timer0 va in overflow.
- **INTOIF** RB0/INT External Interrupt Flag bit =1 se il bit RB0 va a livello alto.
- **RBIF** RB Port Change Interrupt Flag bit =1 se uno dei bit che vanno da RB4 ad RB7 cambia di stato.

Anche questi bit vanno su una porta OR per poi raggiungere l'ultima porta OR filtrati dal bit GIE.
General Input Enable. Questi bit ed I corrispontenti bit di enable sono nel registro INTCON.
Sul registro OPTION_REG troviamo inoltre il bit INTEDG per selezionare il tipo di fronte (salita o discesa) che scatena l'interrupt dal pin RB0.

Ma cosa accade se viene attivato uno di questi interrupt tramite il suo bit di enable, e successivamente si verifica la richiesta di interrupt con il bit di flag relativo?

In primo luogo viene automaticamente resettato il bit GEIE, per evitare che vengano chiamati altri INTERRUPT, e parallelamente nel Program Counter viene messo l'indirizzo 0x04, definito INTERRUPT VECTOR.
Questa zona della memoria programma deve essere perciò lasciata libera per contenere la routine di interrupt che verrà eseguita. Al termine della routine di interrupt con l'istruzione RETFIE e dopo aver riattivato gli interrupt (solo il bit GIE viene riattivato automaticamente dall'istruzione RETFIE) si ritorna nel punto in cui era stato interrotto il programma.

Nell'immagine a fianco vediamo la struttura della memoria di programma, il Reset Vector e cioè la locazione dove inizia il flusso del programma (il valore del Program Counter all'avvio è 0).
Ed all'indirizzo 0004h, l'Interrupt Vector, cioè il valore assegnato al PC, quando viene ricevut l'interrupt.
Alla chiamata dell'interrupt, il valore del PC viene salvato nello stack, per essere recuperato al termine con l'istruzione RETFIE.



Ma quali funzioni vanno fatte nella routine di interrupt?
Innanzitutti devo testare tutti I Flag di interrupt per capire da chi è stato chiamato, successivamente occorre spegnere il bit di Flag che è stato trovato ad uno ed eseguire la funzione richiesta.

E' buona norma non sviluppare funzioni complesse o lunghe all'interno della routine di interrupt, meglio segnalare con apposite variabili la richiesta effettuata in modo da poterla effettuare al momento opportuno.
La gestione dell'interrupt nel linguaggio mikroC, viene effettuata semplicemente dichiarando una routine di tipo void chiamata "interrupt()".

La stessa funzione realizzata precedentemente con il polling viene in questo esempio realizzata utilizzando l'interrupt.

Nel main viene attivato il bit GIE ed il bit INT0IE, successivamente il programma effettuerà il [1]suo compito di far lampeggiare il LED1, con il ciclo while.

Se durante questo ciclo arriva un segnale sul bit RB0, allora viene chiamato l'interrupt ed il programma esegue la relativa routine, dove viene gestita la richiesta di spegnere o accendere il LED2.
Al termine viene effettuata l'istruzione RETFIE che noi non vediamo nel codice C, ma possiamo vederla nel file assembler da esso generato semplicemente compilando questo programma ed aprendo il file con estensione.lst.

```
sbit LED1       at RB1_bit;
sbit LED2       at RB2_bit;
sbit PULSANTE   at RB0_bit;

void interrupt(){          //ROUTINE DI INTERRUPT
  if(INTCON.INTF == 1 ){  //controllo se l'interrupt viene da RB0
    if(LED2==0) LED2=1;  //eseguo la funzione richiesta
    else        LED2=0;
    INTCON.INTF = 0;    //spengo il flat di interrupt
  }
  INTCON.INTE=1;
}

void main() {
  ANSEL  = 0;    //disabilito i convertitori
  CMCON=0x07;    //disabilito i comparatori
  OSCCON=0xFC;  //imposto il clock interno ad 8Mhz
  INTEDG_bit=0;      //attivo il fronte di discesa
  INT0IE_bit=1;      //attivazione dell'interrupt su RB0
  GIE_bit=1;         //attivare l'interrupt generale
  TRISB=0x01;
  LED1=0;
  LED2=0;
  while(1){            //loop continuo
    LED1=1;
    delay_ms(500);
    LED1=0;
    delay_ms(500);
  }
}
```

Questo è solo un semplice esempio, e cambiando tipologia di microcontrollore l'hardware interno potrebbe essere anche più complesso, come ad esempio nei PIC della serie 18, dove può essere gestita anche la priorità dei vari Interrupt.

Il principio di funzionamento rimane comunque simile, ma in ogni caso è indispensabile leggere il datasheet ed analizzare ogni singolo bit e registro coinvolto.

1 www.danielepostacchini.it