INTERRUPT

Interrupt

un segnale asincrono che indica il bisogno di attenzione oppure un evento sincrono che consente l’interruzione di un processo qualora si verifichino determinate condizioni

Siamo a casa e abbiamo un telefono: per vedere se qualcuno ci sta chiamando è necessario che ogni tanto alziamo la cornetta?

No, che fesseria direte voi!

Il telefono squilla avvisandoci che qualcuno ci sta chiamando, quindi alzeremo la cornetta solo quando sta squillando
(vi sembra una banalità eh? Non lo è affatto).

Immaginate: siamo a casa ad effettuare le nostre faccende, non abbiamo bisogno di alzare ogni tanto la cornetta, perchè il telefono ci avviserà con uno squillo, consentendoci di interrompere ciò che stiamo facendo e dedicarci quindi alla telefonata.

Ecco il concetto chiave:

lo squillo del telefono è il nostro interrupt

ci consente di interrompere momentaneamente ciò che stavamo facendo per dedicarci alla situazione che ha generato l'interrupt fino a quando non decidiamo di riprendere le nostre faccende nel punto in cui le avevamo lasciate.

Un interrupt periodico
(come è quindi quello scatenato da un timer),
legato a vari contatori, ci aiuta a creare sistemi
multitasking che operano in tempo reale:

ovvero più funzioni eseguite in “contemporanea”,
indipendentemente dal programma principale!

(in realtà in contemporanea non si può, ma siamo
abbastanza veloci per dar l’impressione di…)

- Quando si verifica un interrupt, si scatenano i seguenti eventi:
  - Il programma principale viene interrotto.
  - Viene salvato il valore numerico del PC (Program Counter) nello Stack.
  - Il PIC salta ad una determinata locazione di memoria (denominata Interrupt Vector) nella quale vengono appunto memorizzate le istruzioni da eseguire in caso di interrupt.
  - Il PIC esegue le istruzioni contenute nell'interrupt vector.
  - Finito di eseguire tali istruzioni, il PIC preleva dallo Stack la riga del programma principale in cui si era fermato.
  - Il programma principale riprende dal punto in cui era stato interrotto dall'interrupt.

Da ciò si capisce che le istruzioni da eseguire durante un interrupt, vengono memorizzate in una locazione di memoria apposita

In C si utilizzerà una notazione particolare di funzione: basterà anteporre la parola “interrupt” davanti al nome della funzione che vogliamo eseguire per far si che venga posizionata nella zona di memoria dedicata all’interrupt

```
void __interrupt() ISR()
{
    // routine eseguita al ricevimento di un interrupt

    return;
}
```

8.2.1 INTCON Register

The INTCON Register is a readable and writable register which contains various enable and flag bits.

Note:	Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).This feature allows for software polling.

Register 8-1: INTCON Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

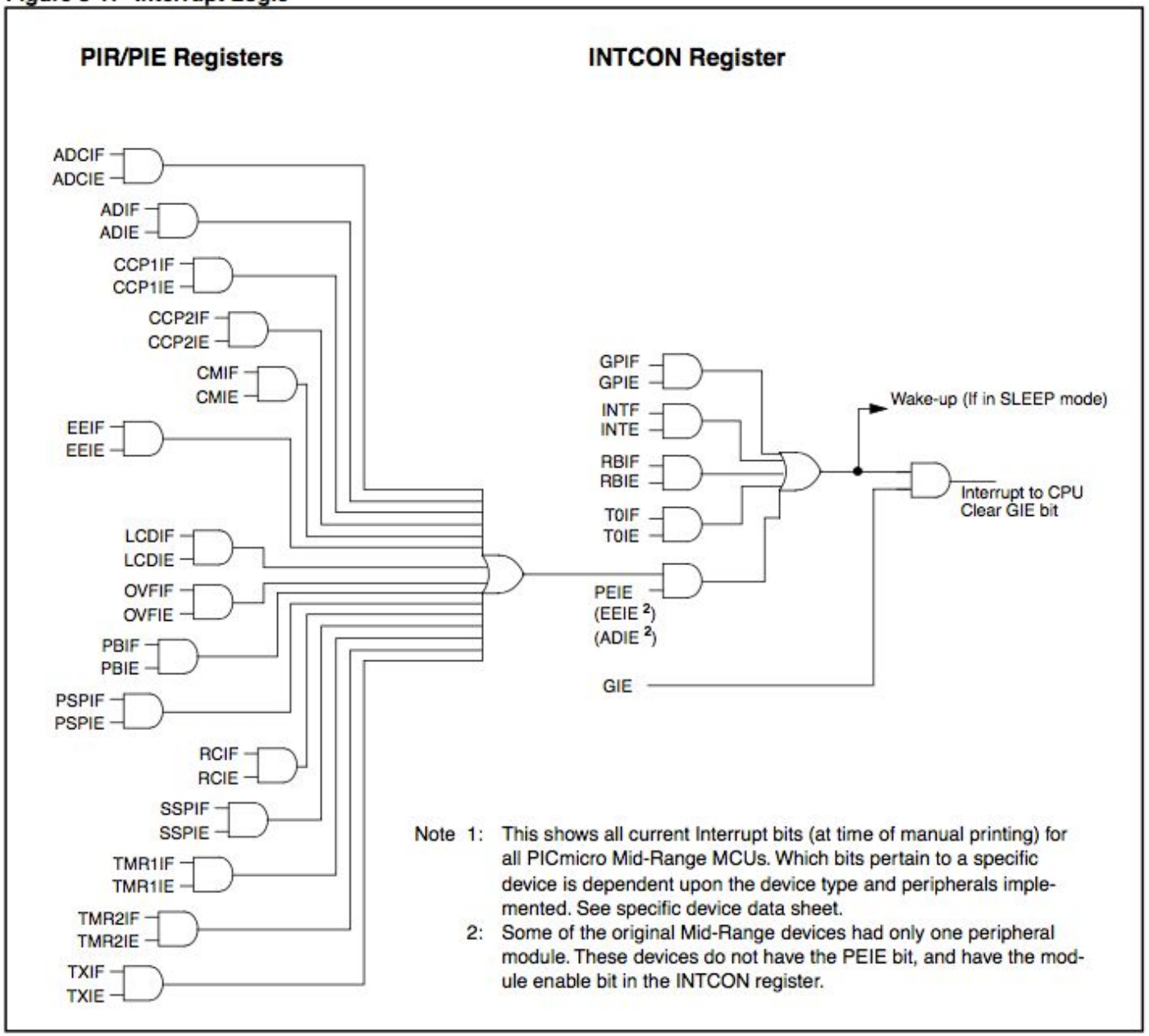
bit 3 **RBIE (1):** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

bit 0 **RBIF (1):** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Figure 8-1: Interrupt Logic



Note 1: This shows all current Interrupt bits (at time of manual printing) for all PICmicro Mid-Range MCUs. Which bits pertain to a specific device is dependent upon the device type and peripherals implemented. See specific device data sheet.

2: Some of the original Mid-Range devices had only one peripheral module. These devices do not have the PEIE bit, and have the module enable bit in the INTCON register.

Timer

il Timer è un contatore che si autoincrementa (indipendentemente dall'esecuzione del programma principale) ogni tot di tempo. Possiamo stabilire noi ogni quanto tempo deve essere incrementato tramite appositi settaggi

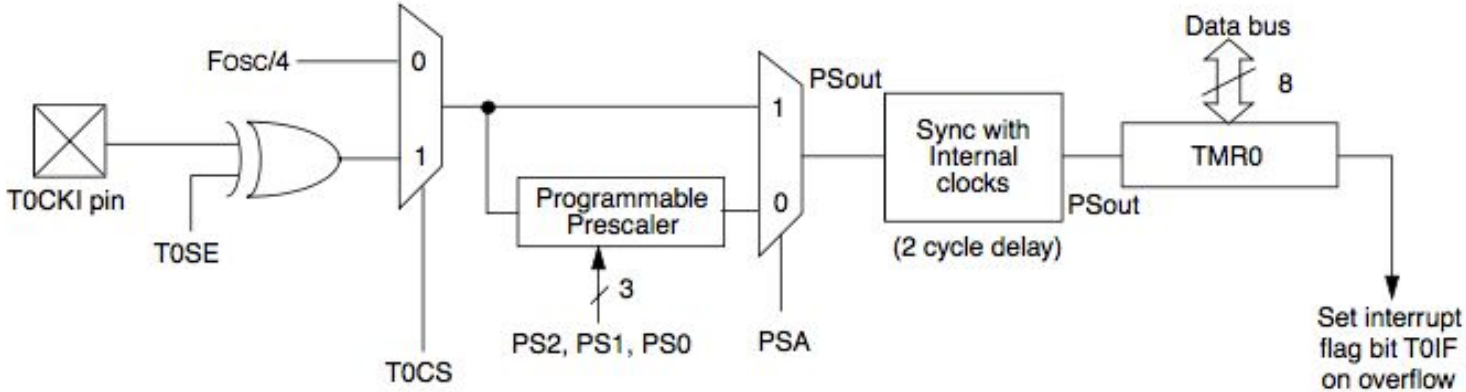
Timer nel PIC16F628A

- Contiene 3 timer indipendenti
- 2 Timer a 8 bit
- 1 Timer a 16 bit
- Il tempo di incremento dipende dal clock di sistema e dalle impostazione del registro di configurazione

Timer0

- 8-bit timer/counter
- Read/write capabilities
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock
- Register is Shared between Timer0 and WDT

Figure 11-1: Timer0 Block Diagram



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 11-6 for detailed block diagram).

bit 2:0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

TABLE 6-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on All Other Resets
01h, 101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h, 181h	OPTION(2)	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	1111 1111

Legend: - = Unimplemented locations, read as '0', u = unchanged, x = unknown. Shaded cells are not used for Timer0.

Note 1: Option is referred by OPTION_REG in MPLAB® IDE Software.

```c
#include <xc.h>

#define _XTAL_FREQ 8000000

static bit led;
int count = 0;

void main(void) {
    TRISB = 0x00;
    INTCON = 0xA0;              //GIE = 1; PEIE = 0; TMR0IE = 1;
    OPTION_REG = 0x07;          //PS0 = 1; PS1 = 1;  PS2 = 1;
    while(1)
    {

    }
    return;
}

void __interrupt() ISR()
{
    if (INTCON&0x04)
     {
        INTCON &= ~0x04; // ~0x04 è la negazione di 0x04 → 0xFB
          // INTCON = INTCON & 0xFB -> 11111011
        count++;
        if (count==30)
        {
            PORTB = ~PORTB;  // nego il valore di tutti i bit di PORTB
            count = 0;
        }
    }
    return;
}
```

INTCON

1	0	1	0	0	0	0	0

OPTION_REG

0	0	0	0	0	1	1	1

INTCON

1	1	1	1	1	0	1	1

Basta poco per ottenere la sequenza:

10101010
01010101
10101010
01010101
10101010
01010101

.

.

.

```
#include <xc.h>

#define _XTAL_FREQ 8000000

static bit led;
int count = 0;

void main(void) {
    TRISB = 0x00;
    PORTB = 0xAA;
    INTCON = 0xE0;              //GIE = 1; PEIE = 1; TMR0IE = 1;
    OPTION_REG = 0x07;          //PS0 = 1; PS1 = 1;  PS2 = 1;
    while(1)
    {

    }
    return;
}

void __interrupt() ISR()
{
    if (INTCON&0x04)
     {
        INTCON &= ~0x04; // ~0x04 è la negazione di 0x04 → 0xFB
        count++;
        if (count==30)
        {
            PORTB = ~PORTB;  // nego il valore di tutti i bit di PORTB
            count = 0;
        }
    }
    return;
}
```

Calcolo dei tempi

Chiaramente è possibile definire con precisione i tempi di lampeggio (o di esecuzione di qualunque funzione) modulando i parametri del PRESCALER e imponendo delle precariche in TMR0

Calcolo dei tempi

Facciamo un esempio pratico: stiamo utilizzando un quarzo da 8Mhz, abbiamo assegnato il Prescaler al Timer0 e abbiamo impostato il prescaler a 256. In tali condizioni, la frequenza con cui il Timer0 si incrementa è pari a:

$$\frac{\left\{ \frac{\text{Fosc}}{4} \right\}}{\text{PS}}$$

Calcolo dei tempi

(ricordo ancora che con Fosc indico la frequenza del Quarzo, con PS indico invece il valore del PreScaler)

Ovvero: (8Mhz/4)/256 = 8.8125KHz

$$\frac{\left\{\frac{Fosc}{4}\right\}}{PS}$$

Calcolo dei tempi

Ma noi vogliamo ragionare in termini di tempo, perchè così ci risulta più facile lavorare, quindi prendiamo la formula precedente e facciamone il reciproco per ottenere il valore di tempo:

$$\left\{ \frac{4}{\text{Fosc}} \right\} \bullet \text{PS}$$

Calcolo dei tempi

ovvero: (4/8)*256 = 128 µS

Quindi in tali condizioni, il nostro Timer0 incrementerà di una unità ogni 128 µS, andrà quindi in overflow dopo (128*256) = 32,768 mS
Come vedete il tempo è già aumentato.

Calcolo dei tempi

Supponiamo adesso di voler ottenere un tempo ben preciso

Diciamo che vogliamo ottenere un interrupt ogni 1 millisecondi, ovvero ogni 1000μS.

Calcolo dei tempi

In questi casi si ricorre ad un trucchetto estremamente semplice: basta semplicemente non far partire il Timer0 dal valore 0:

impostiamo noi il valore da cui partire

Calcolo dei tempi

Abbiamo appena detto che con queste impostazioni, il Timer0 incrementa di una unità ogni 128 µS (ovvero il suo ciclo di istruzioni, grazie al prescaler, adesso è di 128 µS/ciclo), ciò significa che un tempo di 1000 µS lo raggiunge dopo:

1000/128 = 7,8125 cicli

Calcolo dei tempi

teniamo però conto del valore intero: 7 (i cicli sono finiti, non possiamo certo usare il decimale). Bene, se con tali impostazioni, dopo 7 cicli otteniamo un millisecondo, è facile far generare l'interrupt giusto ad un millisecondo: basterà far partire il Timer0 da: 256-7 = 249

Calcolo dei tempi

Partendo da 249, il Timer0 andrà in overflow dopo 7 cicli, che ci darà appunto il tempo di 1 millisecondo!

Calcolo dei tempi

c’è da dire che 1000/128, non fa 7, ma 7,8125
E quel 0,8125 genererà un errore.

Il trucco sta nel bilanciare il valore del prescaler e il valore da caricare in TMR0 per ridurre al massimo l’errore.

Scrivere un codice che generi

il lampeggio di un solo led della porta B

alla frequenza di 125Hz

Ricordo che f = 1/T