Lezione 2

Inizializzazione porte

Il primo argomento che tratteremo è la prima operazione da effettuare quando si programma un microcontrollore ovvero settare correttamente le porte ed inizializzare i registri dedicati che poi andremo ad utilizzare.

Ma prima di far ciò, ricordiamo che l'assemblatore impiegato è l'MPASM, fornito gratuitamente dalla Microchip e copiabile senza incorrere in sanzioni penali. A tale proposito ricordiamo che è possibile accedere a tale eseguibile tramite internet all'indirizzo: http://www.microchip.com.

In figura 1 troviamo il circuito che serve per eseguire le prove di questa lezione.

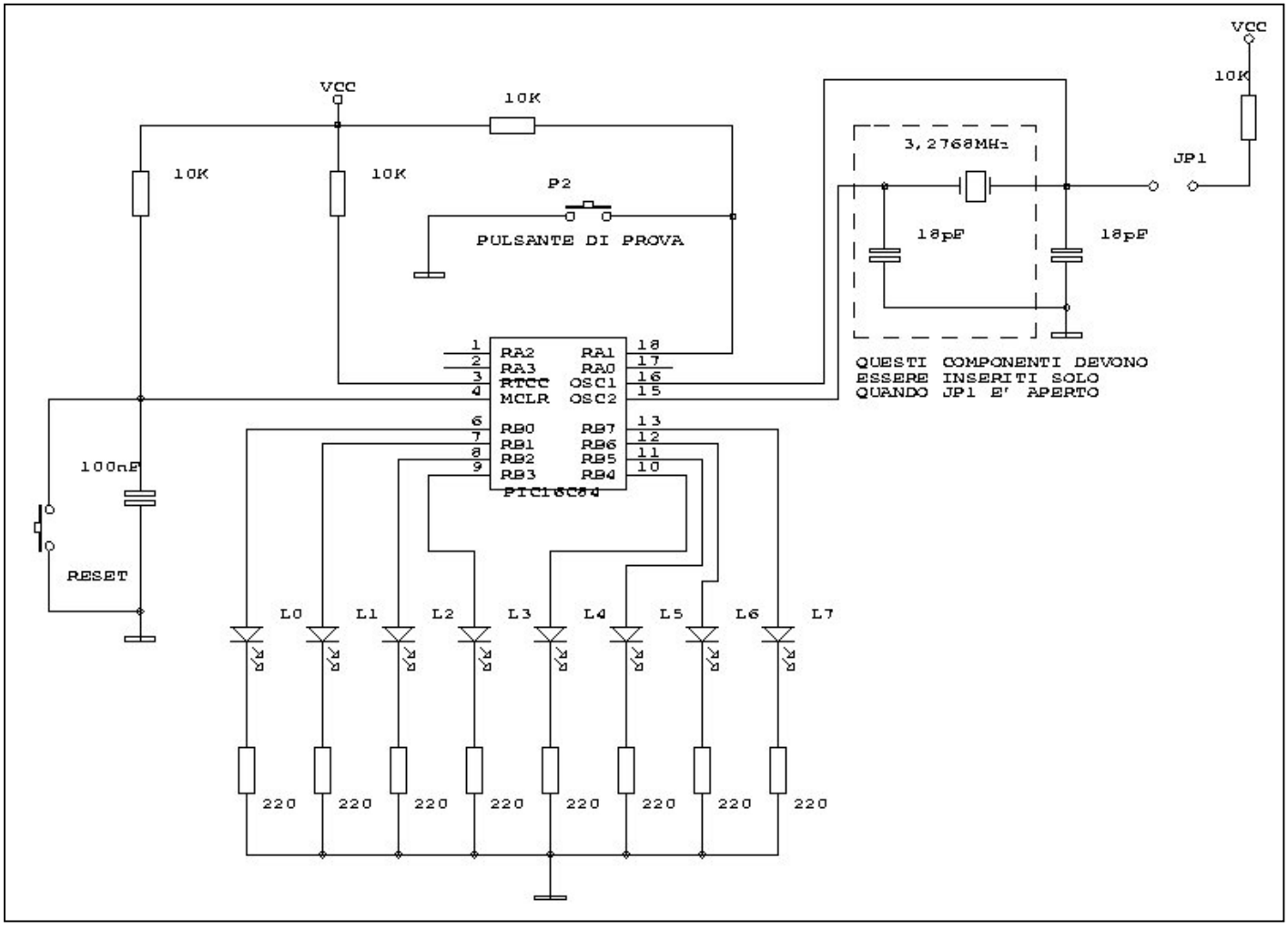


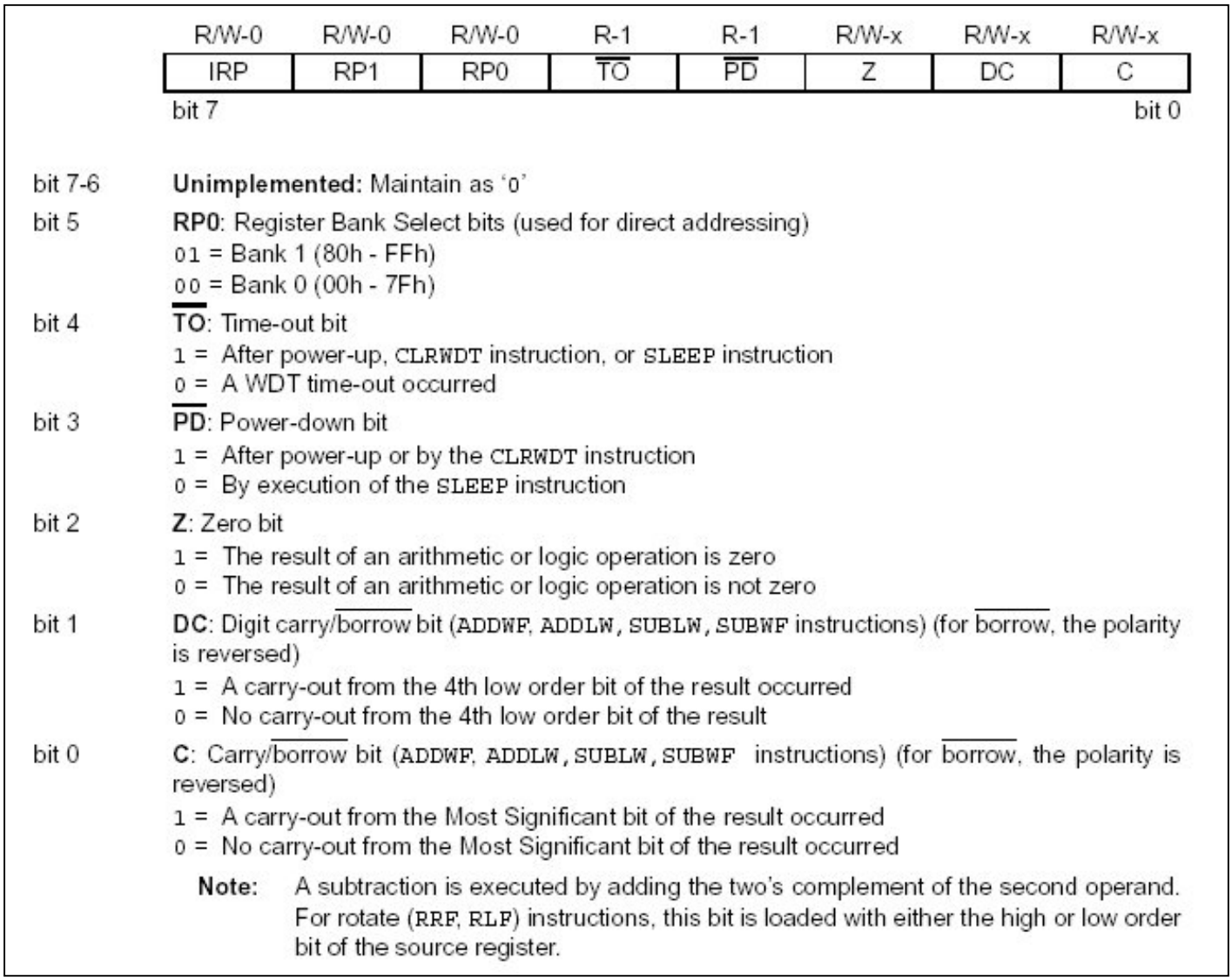
Figura 1. Schema del circuito necessario per le prove

Con il primo programma vedremo inoltre alcune direttive proprie dell'assemblatore. Analizziamo allora il programma PROG01:

```
TITLE 'PROG01: Prova inizializzazione porte'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
STAT   EQU 03H                 ; Registro di stato
PORTA  EQU 05H                 ; Porta A
PORTB  EQU 06H                 ; Porta B
INTCON EQU 0BH                 ; Registro abilitazione interrupt
TR_A   EQU 85H                 ; Tris A
TR_B   EQU 86H                 ; Tris B
;-----------------------------------------------------------------
#define LED0     PORTB,0       ; Led 0 sulla porta B pin 0
;-----------------------------------------------------------------
       ORG       0
       goto      START         ;Reset vector
       nop
       nop
       nop
       goto      INTERP        ;Interrupt vector
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START  bsf       STAT,5        ; Seleziona SRAM banco 1
       movlw     b'0010'       ; RA0,RA2,RA3 out   RA1 in
       movwf     TR_A          ;
       movlw     b'00000000'   ; RB0...RB7 out
       movwf     TR_B          ;
       clrf      INTCON        ; Disabilita interrupt
       bcf       STAT,5        ; Seleziona SRAM banco 0
       clrf      PORTA         ; Uscite della porta A tutte a 0
       clrf      PORTB         ; Uscite della porta B tutte a 0
       bsf       LED0          ; Accensione led 0
;---------------------- main program ----------------------------
MAIN
       clrwdt                  ; Corpo del programma
                               ; .
                               ; .
       goto      MAIN          ; .
;-----------------------------------------------------------------
INTERP                         ; Vettore interruzioni
       retfie                  ;
;-----------------------------------------------------------------
       END
```

La prima riga ovviamente serve per ricordarci il titolo del programma che andiamo a scrivere. La seconda invece dà due direttive all'assemblatore: la prima indica che il formato del file di uscita per il programmatore è di tipo INHX8S, la seconda che il PIC da programmare è del tipo 16C84.
Le sei righe successive sono associazioni di label (etichette) ai registri: se dobbiamo richiamare il registro di stato, è molto più facile ricordarsi "STAT" che non il suo indirizzo esadecimale (03h).
La riga che inizia con la gratella, assegna alla label LED0 il pin 0 della porta B. In questo modo, cambiando in seguito l'hardware, sarà molto facile riassegnare i pin di I/O.

ORG 0 è una direttiva che fa in modo che la istruzione successiva sia posta all'indirizzo 0 della memoria di programma. Nel nostro caso si esegue un "goto START", cioè la prossima istruzione del programma sarà la "bsf STAT,5".
Questa istruzione, pone a 1 il bit numero 5 dello status register. In figura 2 possiamo vedere che in questo modo viene selezionato il banco 1 della SRAM.



R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 $\overline{\textbf{TO}}$: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{\textbf{PD}}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Figura 2. Significato dei bit del registro di stato

Con l'istruzione successiva, viene caricato in W il valore binario "0100". L'istruzione "movwf" poi, copia tale valore nel registro TRISA. In questo modo si definiscono le direzioni dei pin della porta A: dove abbiamo messo uno "0", il pin corrispondente sarà configurato come uscita, dove abbiamo messo un "1", il pin corrispondente sarà configurato come ingresso.

Lo stesso dicasi per le istruzioni relative alla porta B.
Con l'istruzione "clrf INTCON" si disabilitano tutte le sorgenti di interrupt del PIC.
Si ripassa poi al banco 0 della SRAM ponendo a 0 il bit 5 del registro di stato.
Con le due istruzioni successive, si pongono a zero tutti i pin di uscita, quindi, ripassando alla figura 1, i led saranno tutti spenti.
Poichè il programma è terminato, per vedere se effettivamente ha funzionato, abbiamo inserito l'istruzione "bsf LED0", che farà accendere il solo led 0.
Le ultime due righe sanciscono la fine del programma.
Ovviamente questo non è un programma vero e proprio, però tramite questo banale esempio sarete in grado di inizializzare i PIC16C84 in funzione del vostro hardware. In figura 3 troviamo lo stato di tutti i registri del PIC16C84 dopo un reset. I valori dei registri general pourpose rimangono inalterati.

Register	Address	Power-on Reset	$\overline{\text{MCLR}}$ during: – normal operation – SLEEP WDT Reset during normal operation	Wake-up from SLEEP: – through interrupt – through WDT Time-out
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000 0000	0000 0000	PC + 1(2)
STATUS	03h	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA(4)	05h	---x xxxx	---u uuuu	---u uuuu
PORTB(5)	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu(1)
INDF	80h	---- ----	---- ----	---- ----
OPTION_REG	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000 0000	0000 0000	PC + 1(2)
STATUS	83h	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	---1 1111	---u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 x000	---0 q000	---0 uuuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu(1)

I timer software

Vediamo ora come implementare un controllo di tempo molto semplice con il programma PROG02:

```
 TITLE 'PROG02: Prova temporizzazione 1'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
STAT   EQU 03H                 ; Registro di stato
PORTA  EQU 05H                 ; Porta A
PORTB  EQU 06H                 ; Porta B
INTCON EQU 0BH                 ; Registro abilitazione interrupt
TR_A   EQU 85H                 ; Tris A
TR_B   EQU 86H                 ; Tris B
;-----------------------------------------------------------------
FR01   EQU 0CH                 ; Ritardo 20mS
FR02   EQU 0DH                 ; Ritardo 20mS
FR03   EQU 0EH                 ; Ritardo 1S
FR04   EQU 0FH                 ; Ritardo N secondi
;-----------------------------------------------------------------
#define LED0     PORTB,0       ; Led 0 sulla porta B pin 0
#define LED1     PORTB,1       ; Led 1 sulla porta B pin 1
;-----------------------------------------------------------------
       ORG       0
       goto      START         ;Reset vector
       nop
       nop
       nop
       goto      INTERP        ;Interrupt vector
; -------------------- subroutine RITARDO 20 mS ---------------------------
; con frequenza 3,579545 MHz
; -------------------------------------------------------------------------
DELA2M movlw     .20           ; Carica 20 in W
       movwf     FR01          ; Copia W in FR01
       movlw     .177          ; 888 operazioni a 1,117 uS cd.
DEL0   movwf     FR02          ; Copia W in FR02
DEL1   clrwdt                  ; Azzera watchdog
       nop                     ; Nessuna operazione
       decfsz    FR02          ; Decrementa FR02, salta se 0
       goto      DEL1          ; Vai a DEL1
```

Figura 3. Settaggio dei registri dopo un reset

```
decfsz    FR01          ; Decrementa FR01, salta se 0
       goto      DEL0          ; Vai a DEL0
       return                  ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START  bsf       STAT,5        ; Seleziona SRAM banco 1
       movlw     b'0010'       ; RA0,RA2,RA3 out   RA1 in
       movwf     TR_A          ;
       movlw     b'00000000'   ; RB0...RB7 out
       movwf     TR_B          ;
       clrf      INTCON        ; Disabilita interrupt
```

```
        bcf       STAT,5        ; Seleziona SRAM banco 0
        clrf      PORTA         ; Uscite della porta A tutte a 0
        clrf      PORTB         ; Uscite della porta B tutte a 0
        bsf       LED0          ; Accensione led 0
;---------------------- main program ----------------------------
        movlw     .5            ; Carica 5 in W
        movwf     FR04          ; Copia W in FR04
RIT1    movlw     .50           ; Carica 50 in W
        movwf     FR03          ; Copia W in FR03
RIT2    call      DELA2M        ; Vai alla subroutine DELA2M
        decfsz    FR03          ; Decrementa FR03, salta se 0
        goto      RIT2          ; Vai a RIT2
        decfsz    FR04          ; Decrementa FR04, salta se 0
        goto      RIT2          ; Vai a RIT1
        bsf       LED1          ; Accensione led 1
MAIN                            ; Corpo del programma
        clrwdt                  ;
                                ;
        goto      MAIN          ;
;----------------------------------------------------------------
INTERP                          ; Vettore interruzioni
        retfie                  ;
;----------------------------------------------------------------
        END
```

Con PROG02, dopo aver dato l'alimentazione, vedremo accendere il led 2 dopo un tempo di circa 5 secondi, ovviamente impiegando un quarzo da 3,579545 MHZ. Sono state introdotte nuove label, corrispondenti ai quattro registri impiegati e al led numero 1.
Prendiamo in esame la subroutine DELA2M. Poichè sappiamo che il PIC divide per 4 la frequenza di clock, se gli applichiamo un oscillatore da 3,579545 MHz, ogni istruzione verrà eseguita in 1,117 microsecondi. Quindi per ottenere un millisecondo saranno necessarie circa 888 istruzioni. Il ciclo più interno di questa subroutine (relativo a DEL1) esegue esattamente 888 istruzioni, prima di passare al ciclo più esterno (DEL0) che conterà 20 cicli da 1mS per ottenere i 20 mS richiesti.
Andando ora a vedere il main program, dopo aver acceso il led 0, si creano due cicli con cui implementare il ritardo richiesto: con RIT2 si realizzano ritardi da 1 secondo, perchè la subroutine da 20 mS viene richiamata per 50 volte (20mS x 50 = 1.000 mS), mentre con RIT1 vengono contati i secondi di attesa veri e propri. Modificando i valori caricati nei registri, è possibile modificare tutte le tempistiche.
Al termine dei 5 secondi, si accenderà il led 1.
Questo modo di calcolare tempi, può andar bene se non abbiamo particolari esigenze di precisione, ma ci accorgiamo che per piccoli errori di alcuni microsecondi ad ogni ciclo, su alcuni secondi possiamo trovare diversi millisecondi di scarto e, su minuti, troveremmo sicuramente secondi di differenza.
In più, durante queste attese, il chip non può eseguire altri lavori, impegnato come è nel contare il tempo!
Allora si impone un altro modo di lavorare, e precisamente impiegando un registro, l'RTCC (Real Time Clock Counter) che viene incrementato automaticamente ogni n istruzioni eseguite. Il valore di questo n dipende da un prescaler che può essere abilitato o meno.
In figura 4 vediamo come si deve configurare il registro OPTION, cioè il registro che definisce i parametri dell'RTCC e del prescaler. Il bit 7 ed il bit 6 settano altre due funzioni, e cioè l'abilitazione delle resistenze di pull-up sulla porta B e la selezione dell'interrupt sul fronte di salita oppure sul fronte di discesa. Il bit 5 indica se l'incremento dell'RTCC deve avvenire per transizione di livello sul pin RTCC, oppure in relazione al ciclo

di clock interno. Il bit 4 setta se l'incremento deve avvenire su fronte di salita oppure di discesa. Il bit 3 invece assegna il prescaler al watchdog oppure all'RTCC.
Con i bit 2, 1 e 0, si imposta il valore di divisione del prescaler secondo la tabellina riportata a fianco.
In figura 5 è possibile vedere il diagramma a blocchi della condivisione del prescaler tra watchdog e RTCC.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

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

Figura 4. Il registro OPTION

Vediamo allora come implementare dei ritardi precisi avvalendosi dell'RTCC, seguendo il programma PROG03:

```
 TITLE 'PROG03: Prova temporizzazione 2'
 list F=INHX8M,P=16C84
;-----------------------------------------------------------------
RTCC   EQU 01H                  ; Real Time Clock Counter
STAT   EQU 03H                  ; Registro di stato
PORTA  EQU 05H                  ; Porta A
PORTB  EQU 06H                  ; Porta B
INTCON EQU 0BH                  ; Registro abilitazione interrupt
TR_A   EQU 85H                  ; Tris A
TR_B   EQU 86H                  ; Tris B
OPTIO  EQU 81H                  ; Registro OPTION
;-----------------------------------------------------------------
FR01   EQU 0CH                  ; Ritardo 20mS
FR02   EQU 0DH                  ; Ritardo 20mS
FR03   EQU 0EH                  ; Ritardo 1S
FR04   EQU 0FH                  ; Ritardo N secondi
;-----------------------------------------------------------------
#define LED0     PORTB,0        ; Led 0 sulla porta B pin 0
#define LED1     PORTB,1        ; Led 1 sulla porta B pin 1
;-----------------------------------------------------------------
       ORG       0
       goto      START          ;Reset vector
       nop
       nop
       nop
       goto      INTERP         ;Interrupt vector
; -------------------- subroutine RITARDO 20 mS ---------------------------
; con frequenza 3,2768 MHz
; -------------------------------------------------------------------------
DELA2M clrwdt                   ;
       movf      RTCC,0         ; Controllo se finiti 20mS
       SKPZ                     ;
       goto      DELA2M         ;
       movlw     .128           ;
       movwf     RTCC           ;
       return                   ; Ritorna dopo la CALL
;-----------------------------------------------------------------
; START PROGRAM
;-----------------------------------------------------------------
START  bsf       STAT,5         ; Seleziona SRAM banco 1
       movlw     b'0010'        ; RA0,RA2,RA3 out   RA1 in
       movwf     TR_A           ;
       movlw     b'00000000'    ; RB0...RB7 out
       movwf     TR_B           ;
       clrf      INTCON         ; Disabilita interrupt
       movlw     b'11000110'    ; Prescaler 1:128
       movwf     OPTIO          ; Copia W in OPTION
       bcf       STAT,5         ; Seleziona SRAM banco 0
       clrf      PORTA          ; Uscite della porta A tutte a 0
       clrf      PORTB          ; Uscite della porta B tutte a 0
       bsf       LED0           ; Accensione led 0
;---------------------- main program ----------------------------
       movlw     .128           ; Settaggio iniziale RTCC
       movwf     RTCC           ;
```

```
        movlw     .5            ; Carica 5 in W
        movwf     FR04          ; Copia W in FR04
RIT1    movlw     .50           ; Carica 50 in W
        movwf     FR03          ; Copia W in FR03
RIT2    call      DELA2M        ; Vai alla subroutine DELA2M
        decfsz    FR03          ; Decrementa FR03, salta se 0
        goto      RIT2          ; Vai a RIT2
        decfsz    FR04          ; Decrementa FR04, salta se 0
        goto      RIT1          ; Vai a RIT1
        bsf       LED1          ; Accensione led 1
MAIN
        clrwdt                  ;
                                ;
        goto      MAIN          ;
;-----------------------------------------------------------------
INTERP                          ; Vettore interruzioni
        retfie                  ;
;-----------------------------------------------------------------
        END
```

La subroutine di attesa fine 20mS è stata completamente sostituita da un'altra che si avvale del conteggio interno dell'RTCC. Vediamo che il registro RTCC deve essere caricato con un valore, 128, e che il prescaler deve dividere per 128. Questa combinazione è del tutto casuale, perchè potremmo avere dei valori diversi con altri quarzi o con altre tempistiche. Ad esempio, se volessimo con la stessa subroutine ottenere 5mS, sarebbe sufficiente settare il prescaler per una divisione di 32.
Come calcolare questi valori? Se abbiamo il quarzo da 3,2768MHz, la frequenza di lavoro sarà di 3.276.800 Hz / 4 = 819.200 Hz.
Supponendo di prescalerizzare tale frequenza con rapporto 1:32, allora per ottenere 5mS dovremo applicare la formula:

$$(256 - 128) \times (1/(819.200/32)) = 5 \text{ mS}$$

Con un quarzo da 4.096MHz invece dovremo caricare l'RTCC con il numero 96.
Da notare le due righe che sono state aggiunte per settare il registro OPTION con i valori da noi richiesti.
In questo modo otterremo dei tempi precisi al microsecondo, tanto che le applicazioni principali sono proprio timer, orologi e contasecondi.
Per esercitazione, consigliamo di modificare il programma PROG03 per far accendere il led 1 ogni secondo. Nella prossima lezione troverete una delle possibili soluzioni.

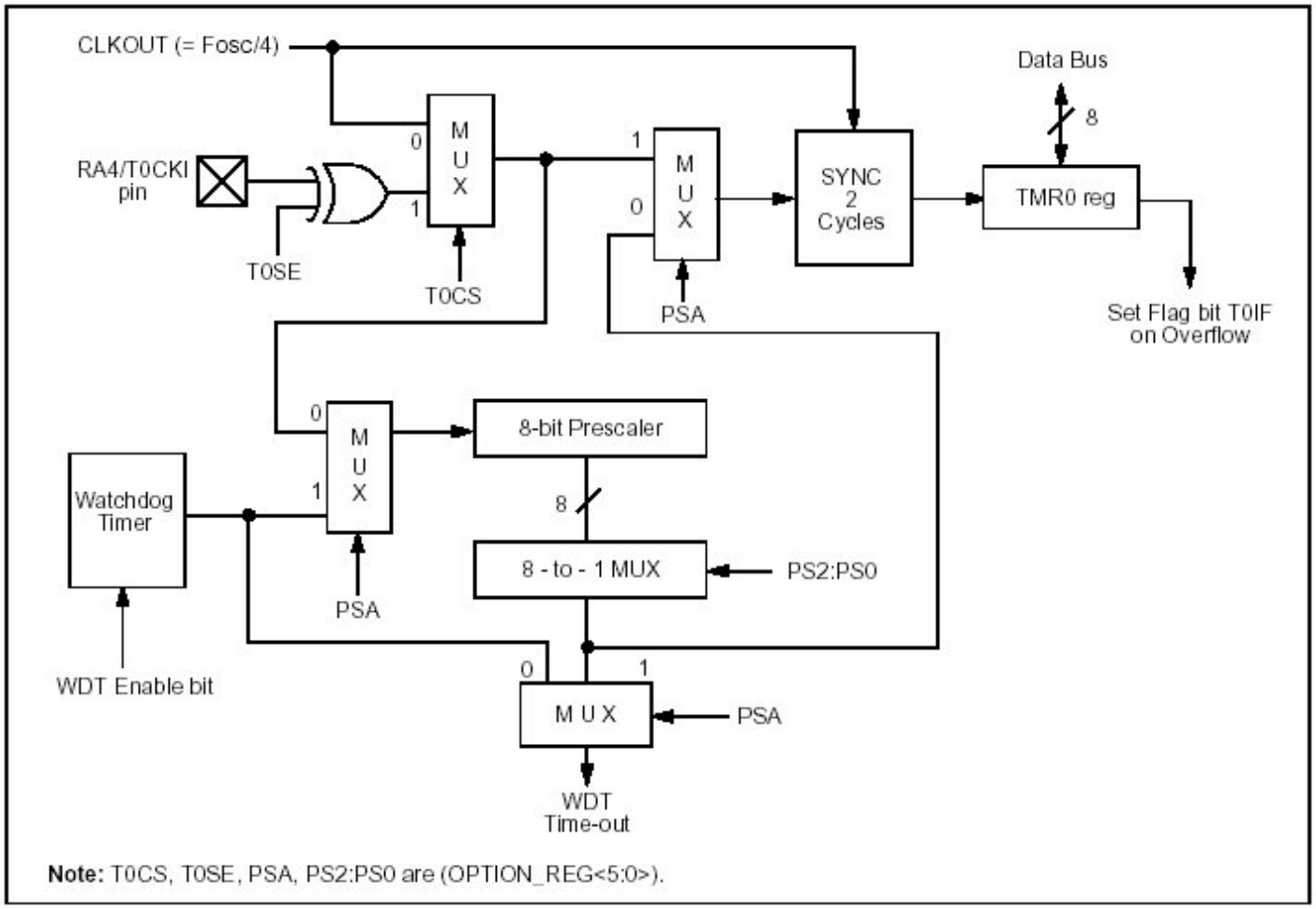


Figura 5. Schema a blocchi dell'RTCC e WDT con prescaler

-continua.