TESINA N°250

Corso: Sistemi Elettronici

Docente: Prof. Falaschetti Laura

Assistente: Ing. Alessandrini Michele

A.A. : 2018/2019

Studente: Gavetti Francesco

Matricola: 1080057

Hardware utilizzato

- MCU: PIC16F887
- Scheda di sviluppo: CEDAR Solutions PIC board 1.1

Programmi utilizzati

- Microchip MPLAB X
- Microchip Serial Bootloader AN1310

Linguaggio di programmazione utilizzato

- Assembly

Descrizione della tesina

Si realizzi un programma che ogni minuto esca dallo sleep e stampi su porta seriale (EUSART) il numero totale di minuti passati, come numero a due cifre decimali.

Informazioni aggiuntive

Gestione eventi: microcontrollore in modalità sleep (dove possibile) in assenza di eventi da processare.

Diagramma di flusso

Il diagramma di flusso del programma è allegato all'ultima pagina.

CODICE

In primis occorre comunicare al programma assemblatore il tipo di microcontrollore che si intende utilizzare e ciò è possibile grazie alla direttiva "*list*":

```
list      p=16f887
```

In questo modo andiamo a definire il set di radix dedicate al nostro specifico processore, ossia quell'insieme ben definito di istruzioni elementari che il nostro processore è in grado di eseguire.

In un secondo momento è necessario rendere disponibile al programma, tramite la direttiva *#include*, tutte le definizioni dei simboli: nomi dei registri, nomi dei bit dei registri, ecc.

```
#include <p16f887.inc>
```

Un file *include* non è altro che un file di testo che contiene un set di definizioni di etichette che deve essere incluso all'inizio del programma.
Nel nostro caso il file include si chiama *p16f887.inc*.

Configuration bits:
I bit di configurazione settano alcune importanti impostazioni hardware del microcontrollore, ad esempio, la disattivazione del watchdog timer.

```
__CONFIG _CONFIG1, _INTRC_OSC_NOCLKOUT & _CP_OFF & _WDT_OFF & _BOR_OFF & _PWRTE_OFF & _LVP_OFF & _DEBUG_OFF & _CPD_OFF
```

Il watchdog timer (WDT) è un timer indipendente dal resto dell'hardware, ossia gode di un oscillatore proprio non appoggiandosi a quello del processore; se abilitato genera un reset all'overflow e pertanto il contatore del WDT deve essere periodicamente azzerato dall'utente.
Nel caso in cui non è stato possibile azzerare il contatore a causa di un blocco o di un malfunzionamento del programma, l'overflow del WDT genera un reset che tenta di risolvere il problema.

Inizializzazione hardware:

All'inizio del programma principale viene chiamata la subroutine initHw, nella quale viene inizializzato l'hardware necessario al funzionamento del programma.

1. Configurazione clock processore

   Impostiamo la frequenza interna del processore a 4MHz andando a settare il registro *OSCCON* (OSCILLATOR CONTROL REGISTER) in questo modo:

   ```
   setRegK     OSCCON, B'01100001'
   ```

   - I bit 6:4 (IRCF<2:0>) impostano la frequenza di clock del PIC e a tal fine sono settati = 110;
   - Il bit finale (bit0 = SCS) è settato a 1, in questo modo si comunica che verrà utilizzato l'oscillatore interno per il clock di sistema.

2. Configurazione prescaler

   Impostiamo il prescaler con un rapporto 1:8, settando i bit PS<2:0> = 010 del *OPTION_REG*:

   ```
   setRegK     OPTION_REG, B'00000010'
   ```

3. Configurazione interrupt:

   Abilitiamo il bit GIE (interrupt globale) e il PEIE (interrupt da periferiche) settando ad 1 i rispettivi bit nel registro INTCON:

```
setRegK     INTCON, B'11000000'
```

4. Impostazioni Timer1:

   Settiamo il registro *T1CON* del Timer1 in modo che quest'ultimo possieda un prescaler pari a 1:8 e lavori in modalità asincrona utilizzando il quarzo esterno con frequenza pari a 32768 Hz.
   Impostando il Timer1 in questo modo, esso potrà essere usato durante lo sleep e potrà contare il massimo periodo possibile, ossia 16s.

```
setRegK     T1CON, B'00111110'
```

5. Inizializzazione seriale:

   Settiamo la seriale in questo modo:
   - High speed della baud rate (BRGH = 1, BRG16 = 0);
   - Abilitiamo la trasmissione: TXEN = 1 (bit5 del TXSTA);
   - Connessione asincrona: SYNC = 0 (bit4 del TXSTA);
   - Abilitiamo ricezione: SPEN = 1 (bit7 del RCSTA);

```
setRegK    TXSTA, B'00100100'
setRegK    RCSTA, B'10010000'
setReg0    BAUDCTL
setRegK    SPBRG, .12
```

   NOTA: anche se utilizziamo la porta seriale unicamente in trasmissione, in quanto dobbiamo solamente stampare, abilitiamo comunque il bit *SPEN* della ricezione del *RCSTA*.

Definizioni variabili e costanti:

A questo punto occorre definire le variabili, le quali permettono di stabilire una quantità precisa di memoria a disposizione nella RAM, in questo modo è possibile memorizzare temporaneamente dei valori indispensabili nel programma.
A queste celle di memoria viene assegnata una etichetta (LABEL) al fine di garantire una maggiore comprensione della loro funzionalità.

Nel codice sono state definite le seguenti variabili nella memoria condivisa (*UDATA_SHR*):

```
canSleep        RES             .1
```

La variabile *canSleep* costituisce un flag che indica quando la CPU può andare in sleep (solo il bit 0 viene utilizzato).

```
w_temp              RES             .1
status_temp         RES             .1
```

Le variabili *w_temp* e *status_temp* permettono il salvataggio dei registri (context saving) durante la gestione di un'interruzione, per poi effettuare il loro ripristino al termine dell'interruzione.

```
flags               RES             .1
```

La variabile *flags* viene utilizzata per salvare il valore di *IRP*, cioè il bit che indica la selezione del banco di memoria nell'indirizzamento indiretto (se *IRP*=0 vengono selezionati i banchi 0 e 1, se *IRP*=1 vengono selezionati i banchi 2 e 3).

```
unita_minuti        RES         .1
decine_minuti       RES         .1
```

Le variabili *unita_minuti* e *decine_minuti* sono dedicate al conteggio dei minuti per la stampa del tempo trascorso dall'accensione del PIC.

```
pointer             RES         .1
```

La variabile *pointer* viene utilizzata nell'indirizzamento indiretto per indicare il successivo byte che dovrà essere stampato.

```
counter             RES         .1
```

La variabile *counter* indica il numero di caratteri che rimangono da stampare.

```
possible_trans      RES         .1
```

La variabile *possible_trans* costituisce un flag che indica quando è possibile trasmettere (la trasmissione è possibile se il bit 0 di *possible_trans* è uguale ad 1, diversamente non è ancora trascorso un minuto).

```
tmr1Counter         RES         .1
```

La variabile *tmr1Counter* permette il conteggio del numero degli overflow del timer1; il valore di *tmr1Counter* viene incrementato di 1 al verificarsi di ogni overflow, ossia ogni 12s.
Una volta che *tmr1Counter* raggiunge il valore 5 significa che è trascorso un minuto e quindi è possibile incrementare il conteggio dei minuti e trasmetterlo.

```
printBuffer         RES         .3
```

La variabile *printBuffer* costituisce il buffer di stampa.

Le costanti, invece, permettono di associare ad un'etichetta una quantità numerica costante.
Le uniche costanti definite nel programma sono:

```
tmr_12s             EQU         (.65536 - . 49152)
```

La costante *tmr_12s* viene definita in modo che il timer sia in grado di contare un periodo di 12s, al termine del quale si verifica l'overflow.
Il contatore di timer1 è a 16 bit, per cui occorre impostare il contatore inizialmente a .65536 - .49152, in quanto non è necessario l'intero periodo di timer1.
Il valore 49152 corrisponde al numero di incrementi del clock necessari al conteggio di 12s ed è appunto pari al prodotto tra il numero dei secondi (12) e il numero di incrementi corrispondenti ad un secondo (4096).

```
length_string       EQU         .3
```

La variabile *lenght_string* indica il numero totale di caratteri da stampare.

Programma principale:

1. Inizializzazione hardware

   Per prima cosa viene chiamata la subroutine *initHw*, la quale inizializza l'hardware come illustrato precedentemente:

```
pagesel     initHw
call        initHw
```

2. Inizializzazione variabili

```
clrf        flags
clrf        unita_minuti
clrf        decine_minuti
clrf        counter
clrf        tmr1Counter
clrf        possible_trans
movlw       .1
movwf       tmr1Counter
```

   Una volta inizializzate le variabili definite precedentemente, vengono settati diversi parametri all'interno dei registri *PIE1* ed *INTCON*.
   Ciò viene fatto al fine di poter imporre uno stato di sleep al dispositivo, con un conseguente risparmio energetico, e risvegliarlo ogni qual volta si verifichi un overflow e quindi sia necessario trasmettere ed eventualmente stampare il conteggio dei minuti.

```
banksel     PIE1
bsf         PIE1, TMR1IE
bsf         INTCON,PEIE
bsf         INTCON, GIE
bsf         canSleep,0
```

   Più nello specifico, nel registro *PIE1* viene settato il bit *TMR1IE* che permette l'interrupt del timer1, mentre nel registro *INTCON*, vengono settati ad 1 il bit *PEIE* che abilita l'interrupt delle periferiche, tra cui il timer1, ed il bit *GIE* (General Interrupt Enable) che abilita la gestione da parte della CPU di qualsiasi evento di interrupt che si verifica durante l'esecuzione del programma.

   Infine viene settato ad 1 il bit0 della variabile *canSleep*, indicando che la CPU può andare in sleep.

3. Main loop

```
bcf         INTCON, GIE
btfsc       canSleep, 0
```

   Per prima cosa si disabilita l'interupt globalmente (*GIE*=0) e si controlla se il dispositivo può andare in sleep. Il controllo avviene tramite l'istruzione condizionale di bit test (*btfsc*) che salta l'istruzione seguente nel caso il bit in esame sia "clear", ossia pari a zero.

```
btfsc       canSleep, 0
goto        goSleep
bsf         INTCON, GIE
goto        main_loop
```

Nel caso in cui il bit0 di *canSleep* sia uguale a zero la CPU non può andare in sleep, per cui si salta l'istruzione successiva, si setta nuovamente il bit *GIE* e si torna al *main_loop* mediante l'istruzione *goto*, per poi effettuare una nuova verifica.

In caso contrario (bit0 di *canSleep*=1), sempre grazie l'istruzione *goto*, si salta alla label *goSleep* che contiene il set di istruzioni che provoca l'ingresso in sleep del dispositivo.

4. Go sleep

```
pagesel     reload_timer1
call        reload_timer1
banksel     T1CON
bsf         T1CON,TMR1ON
sleep
```

In un primo momento, chiamando la routine *reload_timer1*, viene caricato il tmr1 con il valore iniziale in modo che questo conti un periodo di 12s (e non il periodo massimo di 16s) e si abilita lo stesso timer1 settando il bit *TMR1ON* del registro *T1CON*.
Fatto ciò la CPU entra in modalità sleep.

```
bsf         INTCON, GIE
```

A questo punto la CPU si è risvegliata per via di un interrupt che, nel nostro caso, è quello dovuto all'overflow del timer1 e viene riabilitato il bit GIE.

```
btfss       possible_trans, 0
goto        main_loop
```

Quindi si verifica il bit0 della variabile flag *possible_trans*: se è uguale a 0, non è possibile la trasmissione e si torna al main loop, se è uguale ad 1 è possibile trasmettere e viene chiamata la routine di trasmissione, saltando l'istruzione *goto main_loop*.

```
pagesel     transmission
call        transmission
pagesel     print
call        print
pagesel     uart_tx_end
call        uart_tx_end
goto        main_loop
```

5. Transmission

Inizialmente si somma lo 0ASCII a *decine_minuti* in modo da ottenere il valore ascii effettivo di *decine_minuti* per la trasmissione e viene fatta la stessa cosa per unita_minuti.
Successivamente, si salva il valore ascii di *decine_minuti* all'interno del byte0 della variabile *printBuffer*, in modo che le decine di minuti vengano stampate per prime, e il valore ascii di *unita_minuti* all'interno del byte1, così che le unità di minuti vengano stampate come seconda cifra.
Inoltre, all'interno del byte2 di printBuffer, viene salvato il carattere invio, corrispondente al valore 10 nella tabella ascii, in modo tale che ogni nuovo valore stampato venga scritto su una nuova riga.

```
movlw       '0'
addwf       decine_minuti,w
```

```
movwf       (printBuffer+0)

movlw       '0'
addwf       unita_minuti,w
movwf       (printBuffer+1)

movlw       .10
movwf       (printBuffer+2)
```

In seguito, si sposta il valore di *length_string* in *W*, ossia il numero totale di caratteri da stampare, e viene memorizzato nella variabile *counter* che indica quanti caratteri mancano da stampare.
Per cui, prima di iniziare a stampare, la variabile *counter* varrà proprio 3.

```
movlw       length_string
movwf       counter
```

A questo punto, grazie all'utilizzo della funzione "low", viene caricato l'indirizzo del byte0 di *printBuffer*, quello relativo alle decine di minuti, all'interno di *W*, e poi salvato all'interno della variabile *pointer* che, come detto inizialmente, indica l'indirizzo del byte da stampare.

```
bankisel    printBuffer
movlw       low printBuffer
movwf       pointer
```

Infine si resetta il bit0 della variabile *flags* e si effettua il bit test sul bit *IRP* del registro *STATUS*: se IRP è uguale a 1 il bit0 di *flags* viene settato ad 1, altrimenti rimane al valore 0 e si torna al punto del programma in cui era stata chiamata la routine *transmission*.
In questo modo si esegue un backup del valore del bit di *IRP* all'interno del bit0 di *flags*.

```
bcf         flags, 0
btfsc       STATUS, IRP
bsf         flags, 0
return
```

6. Print

All'inizio della routine di stampa ("print") resetto il bit IRP e controllo il bit0 della variabile flags: se è impostato a 0, i valori dei due bit coincidono e salto direttamente all'istruzione successiva, altrimenti setto il bit IRP allo stesso valore del bit0 di flags e procedo all'istruzione seguente.

```
bcf         STATUS, IRP
btfsc       flags, 0
bsf         STATUS, IRP
```

Una volta verificato che siano uguali i valori del bit *IRP* e del bit0 di *flags*, viene caricato in *W* l'indirizzo del byte di *printBuffer* puntato dalla variabile *pointer* e salvato all'interno del registro *FSR* per permettere l'indirizzamento indiretto.
Poi, tramite l'invocazione del registro "fittizio" *INDF*, si salva in *W* il valore effettivo contenuto nell'indirizzo di memoria precedentemente puntato da *FSR*.

```
movf        pointer, w
movwf       FSR
movf        INDF, w
```

Successivamente viene caricato il valore appena salvato in *W* nel registro di trasmissione *TXREG*.

```
banksel     TXREG
movwf       TXREG
```

Dunque, grazie all'utilizzo della tecnica del POLLING, il microcontrollore interroga ciclicamente lo stato della periferica per verificare se è avvenuta o meno la trasmissione.
Quindi, se il bit *TXIF* del registro *PIR1* è settato ad 1 significa che la trasmissione è avvenuta, ma solamente verificando il bit *TRMT* di *TXSTA* si capisce se tutti i byte siano stati effettivamente trasmessi.
In caso il bit *TRMT* non valesse 1, significa che lo shift register non è vuoto, cioè che la trasmissione non è ancora terminata, pertanto occorre aspettare il termine della trasmissione effettuando un'ulteriore verifica.
Contrariamente, il byte è già stato completamente trasmesso, per cui si decrementa il valore di *counter* e si salva il valore ottenuto all'interno di *counter* stesso.

```
banksel     PIR1
btfss       PIR1,TXIF

goto        $-1
banksel     TXSTA
btfss       TXSTA,TRMT

goto        $-1
decf        counter, f
```

In conclusione, si verifica se la trasmissione è terminata, cioè se sono stati trasmessi tutti e tre i byte (*decine_minuti*, *unita_minuti* ed il carattere invio), controllando il bit *Z* di *STATUS*.
Se *Z*=0 allora il *counter* è diverso da 0 e quindi la trasmissione non è ancora terminata, pertanto viene incrementato il *pointer* in modo da puntare il byte successivo e si riesegue la routine di stampa. Altrimenti, se Z=1, il *counter* vale 0, i byte da trasmettere sono terminati e si torna tornare all'istruzione successiva alla chiamata della routine di stampa nel *main_loop*.

```
btfsc       STATUS, Z
return

incf        pointer, f
movf        pointer, w
goto        print
```

7. Uart_tx_end

Una volta terminata la stampa viene chiamata la routine *uart_tx_end* all'interno della quale viene resettata la variabile *possible_trans* e viene settato ad 1 il bit0 della variabile *canSleep*.
In questo modo si indica che non è più possibile trasmettere e che il dispositivo può tornare allo stato di sleep.

```
clrf        possible_trans
bsf         canSleep, 0
return
```

8. Reload timer1

La routine *reload_timer1* contiene il set di istruzioni necessario alla ricarica del contatore timer1 per riavviare il conteggio una volta avvenuto l'overflow.
Con il timer1 impostato in modalità asincrona, occorre arrestare il timer prima di aggiornare i due registri del contatore, perciò tramite l'istruzione *bcf* (bit clear) si resetta il bit *TMR10N* del registro *T1CON*.

```
banksel     T1CON
bcf         T1CON, TMR1ON
```

A questo punto del programma si caricano i due registri di timer1, ossia *TMR1L* e *TMR1H*, grazie all'uso delle funzioni "*high*" e "*low*" che forniscono rispettivamente il byte più e meno significativo di una costante maggiore di 8 bit.

```
banksel     TMR1L
movlw       low  tmr_12s
movwf       TMR1L
movlw       high  tmr_12s
movwf       TMR1H
```

Una volta ricaricato il contatore di timer1 viene resettato il bit *TMR1F* del registro *PIR1*, in quanto l'interrupt relativo all'overflow di timer1 è già stato gestito, quindi si riattiva il timer1.

```
banksel     PIR1
bcf         PIR1, TMR1IF
banksel     T1CON
bsf         T1CON, TMR1ON
return
```

Infine si torna al *main_loop*, all'istruzione successiva alla chiamata della routine *reload_timer1*.

Routine di gestione interrupt:

La routine di gestione degli interrupt ammissibili dal programma inizia con il salvataggio dello stato della CPU, ossia del valore dei registri, in modo che questi possano essere ripristinati una volta gestito l'interrupt e il programma possa riprendere dall'esatto punto in cui si è interrotto.
In caso contrario il programma non sarebbe in grado di riprendere la normale esecuzione, in quanto i valori dei registri risulterebbero modificati dalla routine di interrupt.

```
movwf       w_temp
swapf       STATUS, w
movwf       status_temp
```

Subito dopo avviene il test dell'evento di overflow di timer1, nello specifico si verificano i valori dei bit *TMR1IE* e *TMR1IF*.
Se *TMR1IE* è settato ad 1 significa che è possibile che sia avvenuto un interrupt dovuto all'overflow di timer1, per cui si procede a controllare il bit *TMR1IF*.
Se anche quest'ultimo bit risulta essere settato significa che l'interrupt del timer1 è effettivamente avvenuto, cioè che quest'ultimo è andato in overflow, pertanto si deve controllare se occorre aumentare il contatore dei minuti *tmr1Counter*, o se non è ancora trascorso un minuto.

```
banksel     PIE1
btfss       PIE1,TMR1IE
```

```
goto        irq_end
banksel     PIR1
btfss       PIR1,TMR1IF
goto        irq_end
```

Quindi si carica in *W* il valore 5, si calcola la differenza tra il valore della variabile *tmr1Counter* e *W* e si verifica il valore del bit *Z* all'interno del registro *STATUS*.
Nel caso in cui la differenza valga 0 il bit *Z* vale 1, ciò significa che l'interrupt appena verificatosi è quello relativo al quinto overflow, ossia che è appena trascorso un minuto ed occorre incrementare il conteggio dei minuti, dunque si salta alla label *trans_totMinutes*.
Viceversa, il bit *Z* vale 0, cioè non è ancora trascorso un minuto; quindi viene incrementato di 1 il valore di *tmr1Counter* e si salta a *irq_end*.

```
movlw       .5
subwf       tmr1Counter, w
btfsc       STATUS, Z

goto        trans_totMinutes

incf        tmr1Counter, 1

goto        irq_end
```

1. Trans_totMinutes:

   All'inizio della label *trans_totMinutes* viene resettato il valore del bit0 della variabile *canSleep*, in modo che la CPU non possa andare in sleep.

   ```
   bcf     canSleep,0
   ```

   Fatto ciò si verifica se sono trascorsi o meno 10 minuti e quindi se occorre incrementare il contatore delle decine di minuti o quello delle unità.

   Successivamente, si carica in *W* il valore 9 e si calcola la differenza tra il valore della variabile *unita_minuti* e il valore appena caricato in *W*.
   Se la sottrazione restituisce 0, il bit *Z* del registro *STATUS* vale 1; ciò significa che si è arrivati alla decina di minuti e viene chiamata la label *inc_decine_minuti*.
   Diversamente, se la sottrazione restituisce un valore differente da 0, significa che non sono ancora trascorsi dieci minuti, perciò viene saltata l'istruzione successiva (*goto inc_decine_minuti*), si incrementa il valore delle unità dei minuti, si setta ad 1 la variabile *possible_trans* per indicare che è possibile trasmettere e si salta ad *irq_end*.

   ```
   movlw       .1
   movwf       tmr1Counter
   movlw       .9
   subwf       unita_minuti, w
   btfsc       STATUS, Z
   goto        inc_decine_minuti
   incf        unita_minuti, 1
   bsf         possible_trans, 0
   goto        irq_end
   ```

2. Inc_decine_minuti:

   Viene incrementato di 1 il valore della variabile *decine_minuti* e ovviamente viene azzerata la variabile *unita_minuti*.
   Infine si salta ad *irq_end*.

```
incf        decine_minuti, 1
clrf        unita_minuti
bsf         possible_trans, 0
goto        irq_end
```

3. Irq_end:

   La label *irq_end* termina la routine di gestione degli interrupt; in essa vengono ripristinati i valori dei registri precedenti alla gestione dell'interrupt, in modo che il programma possa riprendere la normale esecuzione dal punto in cui si era interrotto.

```
swapf       status_temp, w
movwf       STATUS
swapf       w_temp, f
swapf       w_temp, w
retfie
```

CODICE COMPLETO

```
; Si realizzi un programma che ogni minuto esca dallo sleep e stampi su porta seriale
; (EUSART) il numero totale di minuti passati, come numero a due cifre decimali.

                list            p=16f887        ; direttiva che definisce il tipo di processore utilizzato
                #include        <p16f887.inc>   ; file che contiene le definizioni dei simboli
                                                ; (nomi registri, nomi bit dei registri, ecc).
                #include        <macro2.inc>

; *** Configuration bits ***
                __CONFIG        _CONFIG1, _INTRC_OSC_NOCLKOUT & _CP_OFF & _WDT_OFF & _BOR_OFF &
_PWRTE_OFF & _LVP_OFF & _DEBUG_OFF & _CPD_OFF


; TIMER1:
; In base alle impostazioni del timer, un periodo di 12s corrisponde a 49152 incrementi.
; Il contatore di timer1 e' a 16 bit, per cui occorre
; impostare il contatore inizialmente a 65536 - 49152
tmr_12s         EQU             (.65536 - .49152)

; *** variabili in RAM (shared RAM) ***
                UDATA_SHR
canSleep        RES             .1      ; flag che indica quando la CPU puo' andare in sleep (solo il bit 0 e' utilizzato)
w_temp          RES             .1      ; salvataggio registri (context saving)
status_temp     RES             .1      ;  "          "           "
flags           RES             .1      ; flag bits
unita_minuti    RES             .1      ; contatore per le unità di minuti
decine_minuti   RES             .1      ; contatore per le decine di minuti
pointer         RES             .1      ; indirizzo prossimo byte da stampare
counter         RES             .1      ; quanti caratteri mi rimangono da stampare?
possible_trans  RES             .1      ; flag di possibile trasmissione
tmr1Counter     RES             .1      ; contatore tmr1
printBuffer     RES             .3      ; buffer di stampa

; *** DEFINIZIONE COSTANTI ***
length_string   EQU             .3      ;numero totale di caratteri da stampare


; VETTORE DI RESET:
RST_VECTOR      CODE  0x0000
                pagesel         start   ; imposta la pagina della PM in cui si trova l'indirizzo della label start
                goto            start   ; salta all'indirizzo indicato dalla label star


; **** PROGRAMMA PRINCIPALE ****
MAIN            CODE
start
; inizializzazione hardware
                pagesel         initHw
                call            initHw          ; inizializzazione hardware

; inizializzazione variabili
                clrf            flags           ; flag bits = 0
                clrf            unita_minuti
                clrf            decine_minuti
                clrf            counter
                clrf            tmr1Counter
                clrf            possible_trans
                movlw           .1
```

```
                movwf       tmr1Counter
                ; abilita interrupt timer1
                banksel     PIE1
                bsf         PIE1, TMR1IE

                ; Abilita gli interrupt delle periferiche aggiuntive (tra cui timer1)
                bsf         INTCON,PEIE     ; abilita interrupt periferiche
                ; Abilita gli interrupt globalmente
                bsf         INTCON, GIE

                ; Inizialmente la CPU puo' andare in sleep
                bsf         canSleep,0      ; la CPU può andare in sleep quando
                                            ; il bit0 di canSleep è settato a 1

                ; RD0 = 1 --> Sleep
                banksel     PORTD
                bsf         PORTD,RD0

main_loop

                banksel     PORTD
                bcf         PORTD,RD1
                bcf         INTCON, GIE     ; disabilita interrupt globalmente
                                            ; prima di controllare il canSleep
                btfsc       canSleep, 0     ; sleep possibile?
                ; se canSleep,0 = 0 --> non può andare in sleep --> torno al main_loop
                ; se canSleep,0 = 1 --> può andare in sleep --> goto goSleep
                goto        goSleep
                bsf         INTCON, GIE     ; se non può andare in sleep
                                            ; prima di tornare al main_loop
                                            ; abilito il GIE
                goto        main_loop

goSleep
                ; carica contatore timer1 con valore iniziale per contare 12s
                ; (e non 16s, ossia il massimo contabile con tmr1 a 16bit)
                pagesel     reload_timer1
                call        reload_timer1
                banksel     T1CON
                bsf         T1CON,TMR1ON
                sleep
                ; a questo punto la CPU si e' risvegliata per via di un
                ; interrupt, che nel nostro caso è il timer1.
                bsf         INTCON, GIE
                ; Avendo riabilitato gli interrupt (bit GIE), viene subito
                ; eseguita la routine di interrupt, quindi il programma
                ; continua.

                ; sleep --> LED0 acceso, LED1 spento
                banksel     PORTD
                bsf         PORTD,RD0
                ; controllo se è possibile trasmettere
                ; è possibile trasmettere se possible_trans,0 = 1
                ; in tal caso chiamo transmission
                ; in caso contrario torno al main loop
                btfss       possible_trans, 0
                goto        main_loop


                ; TRASMISSIONE
                pagesel     transmission
                call        transmission
```

```
                ; STAMPA
                pagesel         print
                call            print
                ; FINE STAMPA
                pagesel         uart_tx_end
                call            uart_tx_end

                goto            main_loop           ; ripete il loop principale del programma

; **** Subroutine initHw ****
initHw
                ; configura clock del PIC OSCCON: OSCILLATOR CONTROL REGISTER
                ; - IRCF<2:0>: 110 = 4 MHz (Internal Oscillator Frequency Select bits)
                ; - SCS: 1=Internal oscillator is used for system clock (System Clock Select bit)
                setRegK         OSCCON, B'01100001' ; 4 MHz internal oscillator

                ; registro OPTION_REG:
                ; - PS<2:0>: 010 (Prescaler Rate Select bits): prescaler impostato a 1:8
                setRegK         OPTION_REG, B'00000010'

                ; registro INTCON:
                ; - GIE abilitato (interrupt abilitato globalmente)
                ; - PEIE abilitato (interrupt da periferiche abilitato)
                setRegK         INTCON, B'11000000'

                banksel         TRISD
                movlw           B'11111100'
                movwf           TRISD

                ; Timer1
                ; Impostazioni:
                ; bit 5,4=11 --> prescaler = 8
                ; bit 2=1   --> modalita' asincrona (funziona con quarzo esterno anche durante sleep)
                ; bit 1=1   --> usa quarzo esterno (32768 Hz)
                ; bit 0=1   --> timer disattivato
                ; Con la frequenza del quarzo ed il prescaler a 8 si ha:
                ; - singolo tick ~= 244 us
                ; - periodo max = 16 s (contatore a 16 bit)
                setRegK         T1CON, B'00111110'

                ; EUSART
                ; baud rate = 19200 (BRGH = 1:high speed, BRG16 = 0)
                ; TXEN = 1
                ; SPEN = 1 : Serial port enabled
                ; CREN = 1 : Continuous Receive Enable bit : Asynchronous mode: 1 = Enables receiver
                setRegK         TXSTA, B'00100100'
                setRegK         RCSTA, B'10010000'
                setReg0         BAUDCTL
                setRegK         SPBRG, .12

                return                      ; uscita da subroutine e ritorno al punto del codice in cui era stata chiamata

reload_timer1
                ; ricarica contatore timer1 per ricominciare conteggio.
                ; In modalita' asincrona, occorre arrestare il timer prima
                ; di aggiornare i due registri del contatore
                banksel         T1CON
                bcf             T1CON, TMR1ON  ; arresta timer
                ; le funzioni "low" e "high" forniscono il byte meno e piu'
                ; significativo di una costante maggiore di 8 bit
                banksel         TMR1L
                movlw           low  tmr_12s
```

```
                movwf   TMR1L
                movlw   high  tmr_12s
                movwf   TMR1H
                banksel PIR1
                bcf     PIR1, TMR1IF            ; azzera flag interrupt
                banksel T1CON
                bsf     T1CON, TMR1ON           ; riattiva timer
                return

; ***ROUTINE GESTIONE INTERRUPT***
irq             code    0x0004
                ; context saving (vedere datasheet)
                movwf   w_temp
                swapf   STATUS, w
                movwf   status_temp
                ; testa evento overflow timer1 (TMR1IF + TMR1IE)
                banksel PIE1
                btfss   PIE1,TMR1IE     ; se IE è abilitato vado a controllare
                                        ; se anche IF è abilitato
                goto    irq_end
                banksel PIR1
                btfss   PIR1,TMR1IF     ; se sia l'IE che l'IF sono abilitati
                                        ; significa che è avvenuto l'interrupt
                                        ; del timer1, cioè che è andato in overflow
                                        ; quindi devo andare a controllare se
                                        ; occorre incrementare il contatore
                                        ; dei minuti (tmr1Counter) o se ancora
                                        ; non sono arrivato ad 1 minuto
                goto    irq_end
                ; *** CONTROLLO SE SONO ARRIVATO AD 1 MINUTO ***
                ; salvo 5 in W e faccio la differenza (tmr1Counter - W)
                ; - se (tmr1Counter - W) = 0
                ;   --> tmr1Counter = W --> Z = 1
                ;   --> sono arrivato ad 1 minuto
                ;   --> goto trans_totMinutes
                ; - se (tmr1Counter - W) ? 0
                ;   --> tmr1Counter ? W  --> Z = 0
                ;   --> non sono ancora arrivato ad 1 minuto
                ;   --> incremento tmr1Counter (perché comunque ho ricevuto un overflow dei 5 che mi servono)
                bcf     PIR1, TMR1IF

                movlw   .5
                subwf   tmr1Counter, w
                btfsc   STATUS, Z

                goto    trans_totMinutes

                incf    tmr1Counter, 1  ; incremento variabile tmr1Counter di 1

                ; fine evento timer1
                goto    irq_end

trans_totMinutes
                bcf     canSleep, 0
                ; *** INCREMENTO CONTATORE MINUTI E VERIFICO SE SONO ARRIVATO A 10 MINUTI ***
                ; salvo 9 in w e faccio la differenza (unita_minuti - W)
                ; - se (unita_minuti - W) = 0
                ;   --> (unita_minuti) = W --> Z = 1
                ;   --> sono arrivato a 9 e sto per incrementare (quindi sono arrivato a 10min)
                ;   --> goto    inc_totmin_byte0

                ; - se (unita_minuti) - W ? 0
```

```
                ;   --> (unita_minuti) ? w --> Z = 0
                movlw       .1
                movwf       tmr1Counter
                movlw       .9
                subwf       unita_minuti, w         ; sottrazione tra byte basso (unità minuti) e w
                btfsc       STATUS, Z
                goto        inc_decine_minuti       ;incremento byte più basso (unità minuti)
                incf        unita_minuti, 1
                bsf         possible_trans, 0
                goto        irq_end

inc_decine_minuti
                incf        decine_minuti, 1
                clrf        unita_minuti
                bsf         possible_trans, 0
                goto        irq_end

irq_end
                ; ripristino registri
                swapf       status_temp, w
                movwf       STATUS
                swapf       w_temp, f
                swapf       w_temp, w
                retfie

transmission
                banksel     PORTD
                bcf         PORTD,RD0
                bsf         PORTD,RD1
                movlw       '0'
                addwf       decine_minuti,w         ; sommo 0ASCII a decine_minuti
                                                    ; in modo da ottenere il valore ascii
                                                    ; effettivo di decine_minuti
                movwf       (printBuffer+0)         ; salvo il valore ascii decine_minuti
                                                    ; nel byte0 di printBuffer
                movlw       '0'
                addwf       unita_minuti,w
                movwf       (printBuffer+1)
                movlw       .10                     ;carattere invio
                movwf       (printBuffer+2)
                movlw       length_string           ; sposto lenght_string in W (= 3)
                movwf       counter                 ; memorizzo il numero di caratteri
                                                    ; da stampare nella variabile counter

                bankisel    printBuffer
                movlw low   printBuffer             ; ottengo l'indirizzo del buffer

                movwf       pointer                 ; metto l'indirizzo del primo byte di
                                                    ; printBuff nella variabile pointer

                bcf         flags, 0                ; metto bit0 di flags a 0
                ; - se il bit IPR di status è zero
                ;   --> return alla call transmission in go_sleep
                ;   --> dove chiamerò il print
                ; - se il bit IRP di status è uno
                ;   --> setto il bit0 di flags a 1
                btfsc       STATUS, IRP             ; INDIRIZZAMENTO INDIRETTO
                bsf         flags, 0                ; devo fare in modo che
                                                    ; IRP = bit0 di flags
                                                    ; bit0 di flags mi serve da backup

                return
```

```
print
                ; controllo bit0flags/IRP
                bcf             STATUS, IRP             ; IRP = 0
                btfsc           flags, 0
                bsf             STATUS, IRP             ; a meno che bit0 = 1
                movf            pointer, w              ; metto pointer(puntatore al byte
                                                        ; corrente di printBuffer) dentro a W
                movwf           FSR                     ; metto il valore di pointer in FSR
                                                        ; (per l'indirizzamento indiretto)
                                                        ; cioè l'indirizzo di memoria del byte
                                                        ; corrente di printBuffer puntato da pointer
                movf            INDF, w                 ; carico in W il valore vero

                ; inizio trasmissione dei byte
                banksel         TXREG
                movwf           TXREG                   ; metto il valore vero in TXREG

                ; controllo se è avvenuta la trasmissione, altrimenti aspetto (POLLING)
                banksel         PIR1
                btfss           PIR1,TXIF               ; se il bit è 1 allora il carattere è stato
                                                        ; trasmesso, sennò aspetto
                                                        ; anche se il TXIF = 1, non è detto che il
                                                        ; byte sia stato tutto trasmesso
                goto            $-1
                ; controllo se ho terminato la trasmissione di un byte
                banksel         TXSTA
                btfss           TXSTA,TRMT              ; se il bit è 1 allora lo shift register (SR)
                                                        ; è vuoto sennò aspetto --> verifico di nuovo
                goto            $-1
                decf            counter, f              ; se ho stapato un byte, decremento counter
                                                        ; (che mi dice il numero di byte da stampare)
                                                        ; e salvo il valore di counter in f
                btfsc           STATUS, Z               ; se Z=0 allora counter è diverso da 0 e si continua la
                                                        ; trasmissione dei byte
                return                                  ; se Z=1 allora counter=0 e quindi i byte da trasmettere sono
                                                        ; terminati
                incf            pointer, f              ; incrementa puntatore dati
                movf            pointer, w
                goto            print

uart_tx_end
                ; caso di dati da trasmettere terminati
                clrf            possible_trans
                bsf             canSleep, 0
                return

                end
```

DIAGRAMMA DI FLUSSO