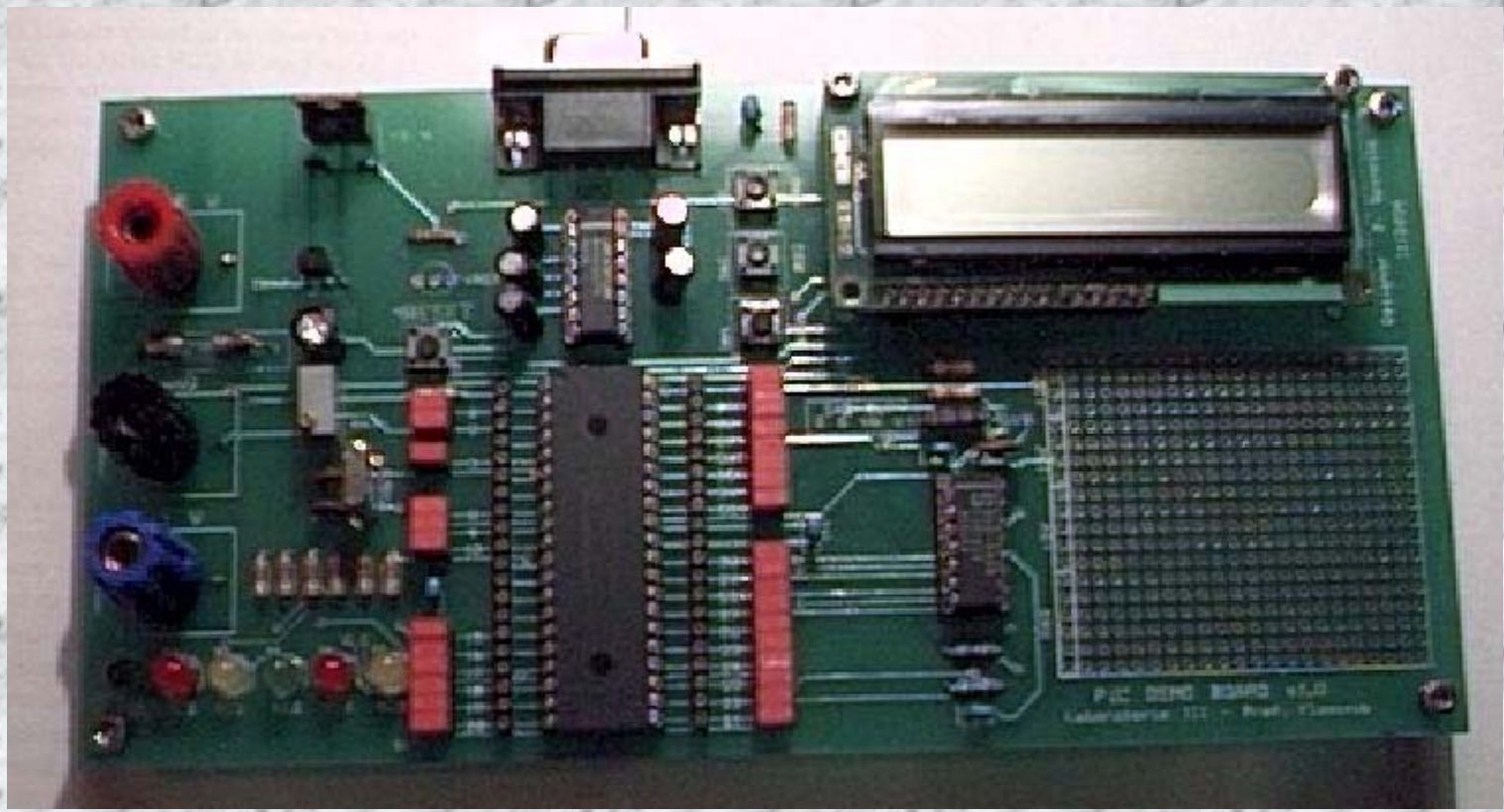
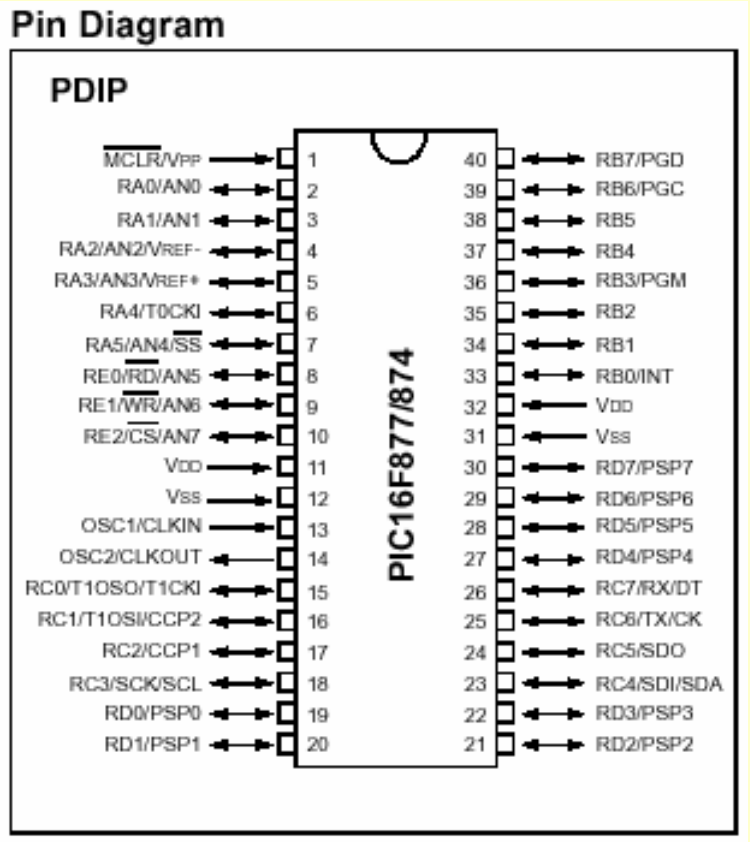
CORSO INTRODUTTIVO ALLA PROGRAMMAZIONE DEI MICROCONTROLLORI (PIC16F877)



Un dispositivo che puo` essere programmato per eseguire una serie di operazioni di varia natura e` un microcontrollore. Per esempio posso programmare il microntrollore per misurare la temperatura in una stanza ed accendere una sirena non appena la temperatura supera una certa soglia.

Il microntrollore che utilizzeremo è il PIC 16F877 della ditta Microchip. Le operazioni del microntrollore sono sincronizzate utilizzando un clock di frequenza 4 MHz. Il Pic esegue 1 istruzione elementare ogni ciclo macchina che corrisponde a 4 cicli di clock .

PIC == Programmable Intelligent Computer



Cosa è un microcontrollore ?

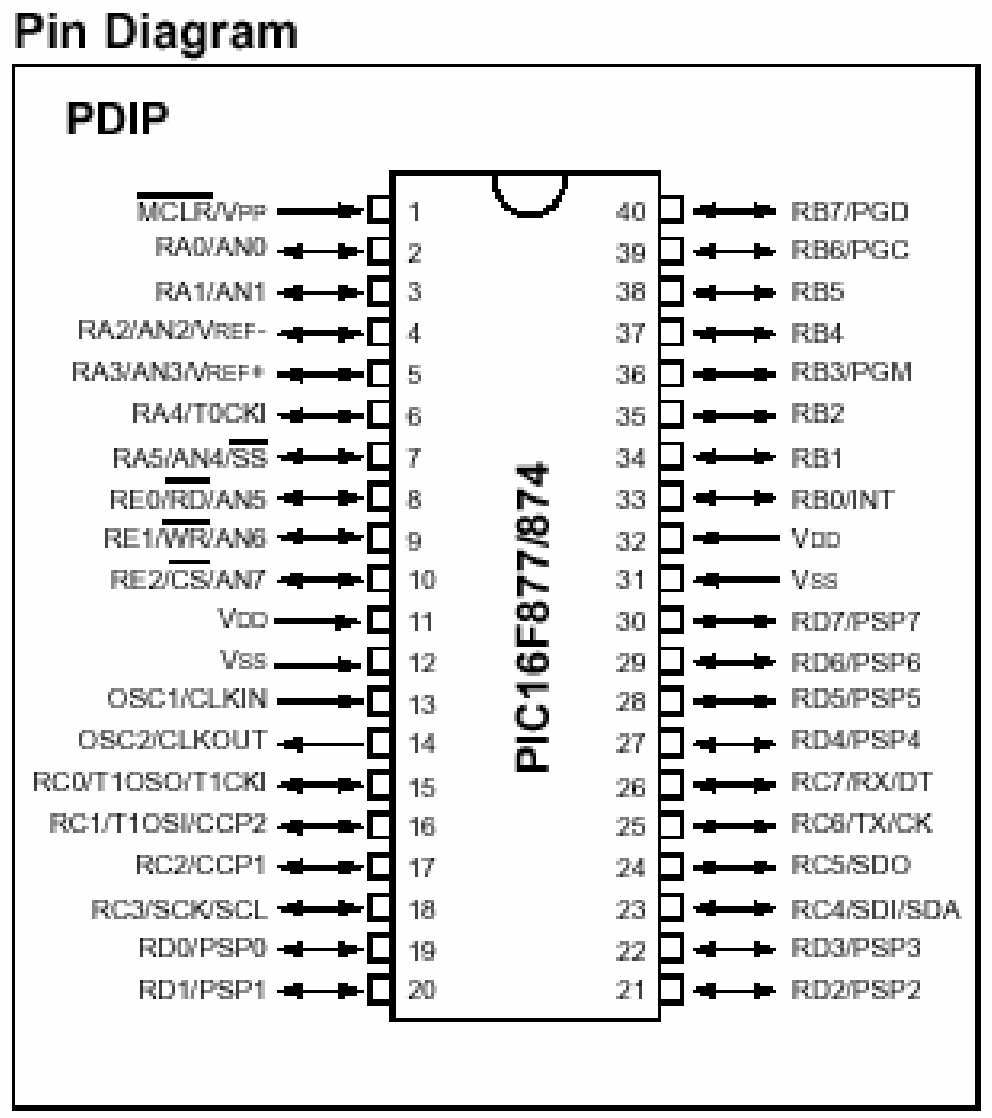
- Un piccolo computer, contenente al suo interno tutti i circuiti necessari al suo funzionamento, senza necessita` di circuiti integrati esterni.
- Il microprocessore vero e proprio (core) e` il cuore del sistema e si occupa di eseguire le operazioni matematiche (ALU) , di spostare i dati fra le varie parti della memoria, di incrementare i numerosi contatori necessari al funzionamento.
- Caratteristica peculiare del microntrollore che utilizziamo è che dati e istruzioni si muovono su bus diversi (architettura di Harvard).
- Tutti i dispositivi, interni, controllati dal microprocessore prendono il nome di **periferiche**

Vedremo durante le esercitazioni la funzione di molti dei 40 pin del Microcontrollore. Vediamone intanto alcuni con funzioni semplici :

Riferimento 0 logico → pin 12, 31

Riferimento 1 logico → pin 11, 32

Pin 1: inizializzazioni. Collegando questo pin a 0 logico il microcontrollore arresta l'esecuzione del programma in corso ed esegue la routine di reset (carica in tutti i registri i valori iniziali ed inizia l'esecuzione del programma in memoria).

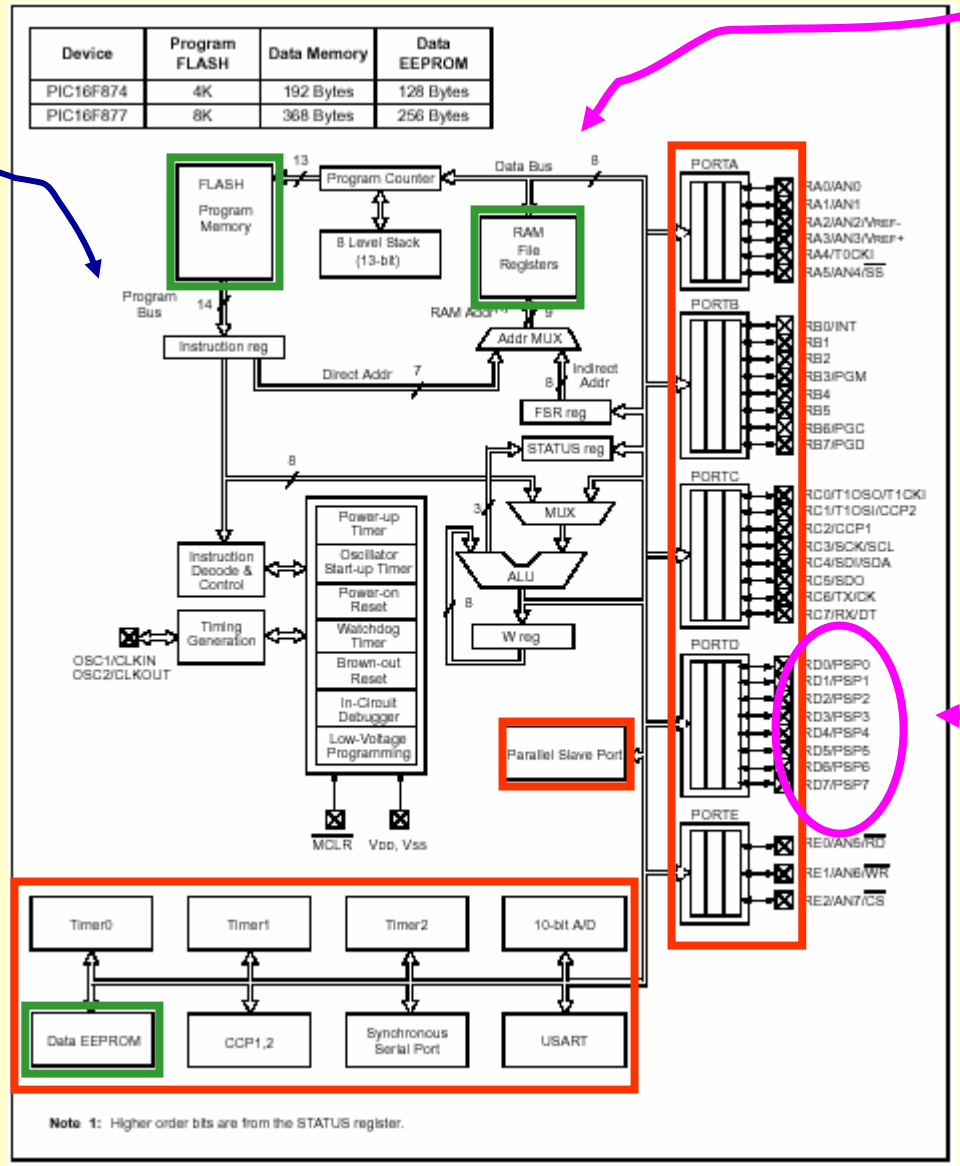


Architettura PIC 16F877

Program bus

Data bus

Microprocessore



Memorie interne

Periferiche interne

Notate che qui vengono specificati i nomi che vengono dati ai vari pin, sono gli stessi che troveremo piu` avanti sullo schema elettrico

Memorie interne al 16F877

- **FLASH Memory** programmi: contiene il “ programma’’ da eseguire, ha la caratteristica di mantenere la programmazione anche quando l’alimentazione al micro viene spenta, e di poter essere cancellata e riscritta con uno speciale apparecchio chiamato programmatore. E’ una particolare EEPROM in cui non si può riscrivere un singolo byte per volta.

Questa memoria e` profonda 8192 parole (indirizzo di 13 bit) ed ogni parola è di 14 bit (potete controllare la larghezza del “program bus”) $\Rightarrow$ il programma da eseguire al piu` puo` essere composto da una sequenza di 8192 operazioni.

- Altri tipi di microprocessore hanno invece una memoria programma di tipo PROM: questa puo` essere scritta soltanto una volta.

Memorie interne al 16F877

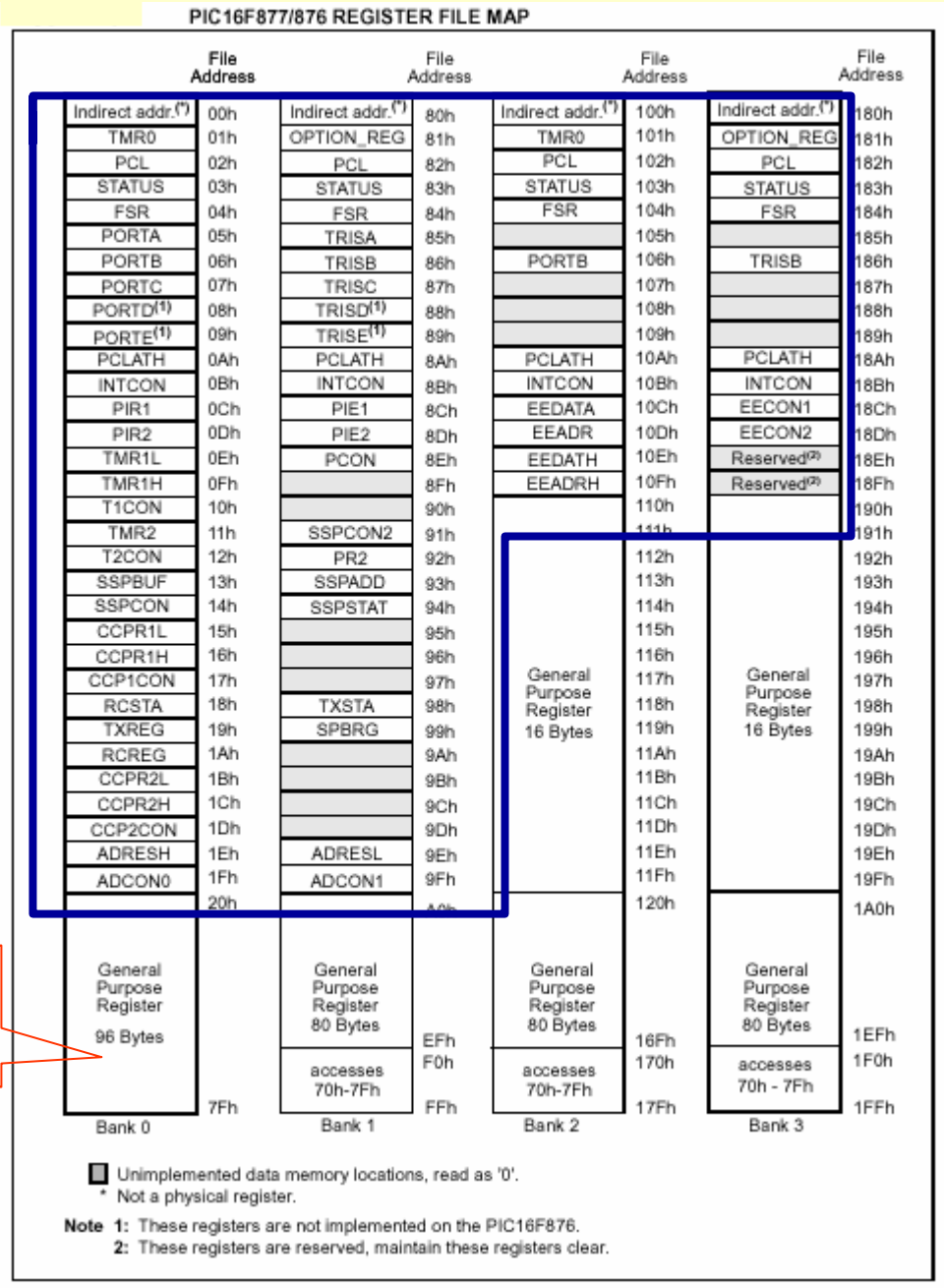
- **RAM** dati: il programma, in esecuzione, non puo` scrivere sulla FLASH programmi ( ~ vero …) ⇒ l'area di memoria RAM dati e` scrivibile in esecuzione e contiene le variabili.

Il contenuto di questa memoria viene perso quando si spegne il circuito. Questa memoria è organizzata in 4 banche da 128 byte
(indirizzi prima banca 00h fino a 7Fh). La larghezza del "data bus" è quindi pari a otto bit.
Le locazioni di memoria più basse sono riservate a registri per funzioni speciali dedicate a definire la configurazione del microcontrollore od a istruirlo ad eseguire determinate operazioni. Le locazioni di memoria più alte sono a disposizione dell'utente.
Quando si programma in un linguaggio diverso dal linguaggio macchina è il compilatore che si occupa della gestione della RAM.

- Esiste un'altra memoria FLASH per i dati (EEPROM) una specie di HardDisk che non utilizzermo.

Mnemonica Registri di uso speciale

Registri di uso generale



ffh == ff hexadecimal

$11h = 0x11 = 00010001 = 2^0 + 2^4$

ffh = 0x f f = 1111 1111

0000 0100 = 0x04 = 4h

Ricapitolando:

1) FLASH programmi: 8192 parole da 14 bit.
2) RAM per le variabili: 368 bytes (dei totali 511)
3) FLASH dati: 256 bytes

Le singole locazioni della RAM vengono chiamate registri.
Abbiamo visto che i registri della RAM possono essere di due tipi :

1) General Purpose: uso generale, tipo per contenere le variabili del nostro programma.
2) Special Function: scrivendo in queste locazioni si istruisce il micro ad eseguire determinate operazioni. Per esempio se scrivo il dato 0x4 nel registro 1Fh (mnemonica ADCON0) $\Rightarrow$ l'ADC interno al microprocessore comincia la conversione dal valore di tensione analogico a quello digitale.

Istruzioni macchina utilizzare dal 16F877 RISC

La famiglia dei microntrollori di tipo RISC (Reduced Instruction Set Computer) supporta un nro di codici operativi limitato, nel nostro caso 35, altamente ottimizzati in termini di velocità di esecuzione.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Istruzioni macchina utilizzare dal 16F877 RISC

Sono operazioni (AND OR ADD …) per lo più eseguite nella ALU che utlizzano il registro W (accumulatore).

Sono operazioni che modificano o dipendono da un singolo bit.

Sono istruzioni che contengono al loro interno il valore di una costante, esempio l'indirizzo di un registro a cui saltare.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff		
NOP		No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Istruzioni macchina utilizzare dal 16F877 RISC

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
[illegible]	f, d	[illegible]ment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
[illegible]		[illegible]	1	[illegible]	C	1,2

MOVWF f – copia il contenuto della locazione di memoria f nel registro accumulatore W

Tre righe in linguaggio macchina…

```
1: ;-- questo è una riga di commento
2: testreg equ 0xC
3: label1 movwf testreg
4:        btfss testreg,2
```

Le righe che iniziano per ; sono commenti.

L'operazione della riga due assegna (equ) 0xC a testreg.

La riga tre è preceduta da una etichetta "label1" e copia il valore di testreg nell'accumulatore W.

Nell'ultima riga viene testato se il valore del 2 bit di testreg è 1 ed in caso positivo salta l'istruzione che segue …

Noi non abbiamo bisogno di conoscere veramente il linguaggio macchina, useremo il C.

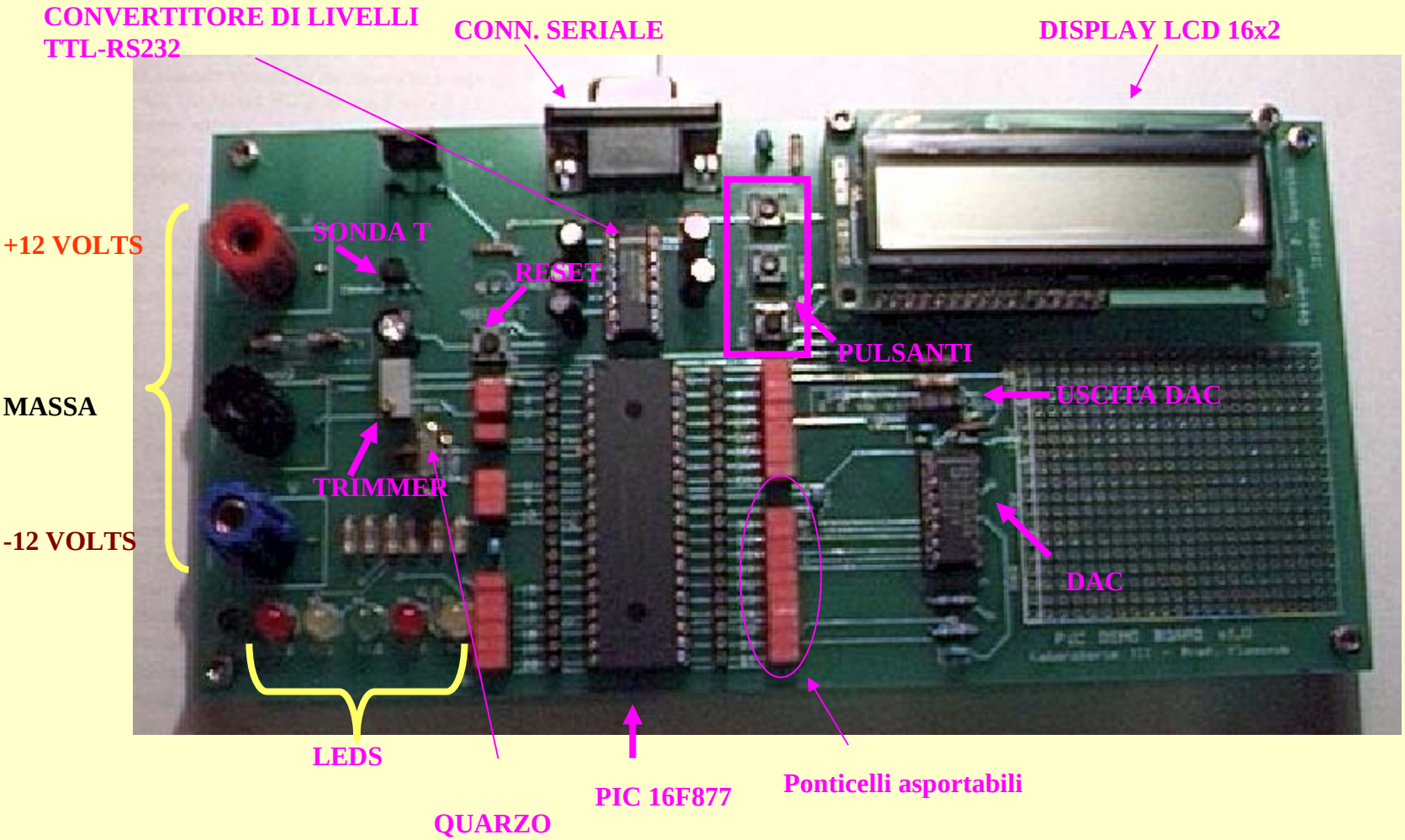
Elenco delle periferiche interne

solo quelle che utilizzeremo

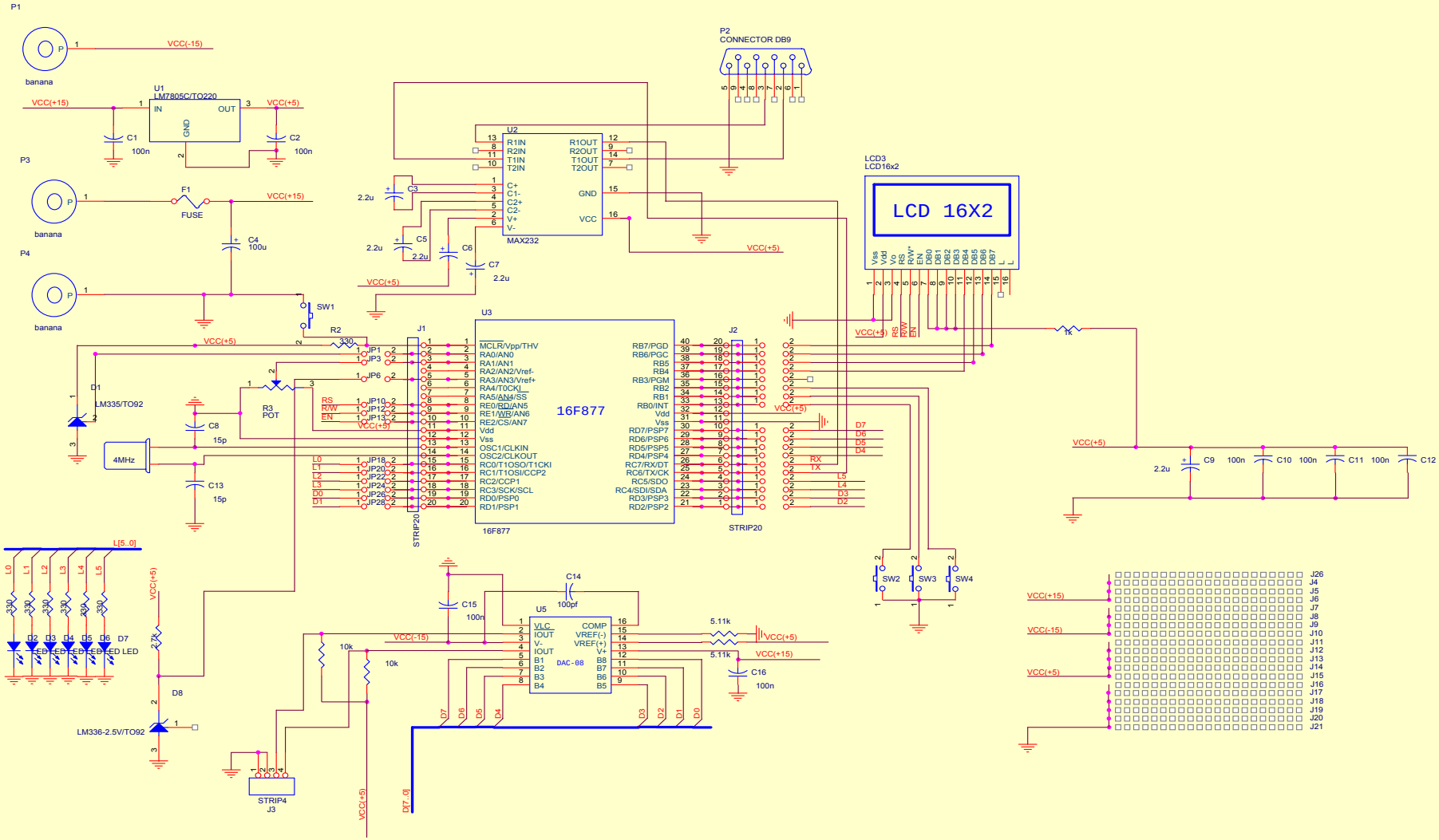
- Porte di I/O : 33 pin . Possono essere configurati come: ingressi ed uscite TTL oppure come ingressi analogici;
- USART: porta seriale, per programmare il PIC e per colloquiare col PC;
- ADC: 10 bit ADC, 9 ingressi multiplexati. Freq: ~ 30 Khz;
- Interrupt generator: generatore di interrupts.

Il 16F877 contiene anche molte altre periferiche (PWM, CCP, FLASH dati, parallel port) che non utilizzeremo

Basetta di test PICDEMO



Schema elettrico



Utilizzo delle porte

Prendete lo schema elettrico e controllate come sono connessi i pin delle varie porte :

ad esempio porta B : pin 0, 1, 2 $\rightarrow$ pulsanti;

porta B : pin 4 –7 $\rightarrow$ LCD;

porta C : pin 0 – 5 $\rightarrow$ LEDs;

porta C : pin 6, 7 $\rightarrow$ porta seriale IN e OUT;

porta D : pin 0 – 7 $\rightarrow$ DAC;

porta E : pin 0 – 2 $\rightarrow$ LCD.

Individuate sulla basetta i vari collegamenti.

Periferiche esterne presenti sulla demo board:

- Display LCD 16 caratteri per 2 righe, HD44780
- Convertitore di livello TTL-RS232
- Digital to analog converter DAC-08
- 6 LEDs
- 3 pulsanti + pulsante di reset
- Potenziometro da 20Kohm o 100 ohm a seconda delle schede
- Sonda di temperatura LM35
- Quarzo da 4 MHz

La documentazione completa di ogni chip in formato pdf e lo schema elettrico in formato jpg sono nella pagina web del corso

Comunicazione tra tastiera/schermo e PIC

Per poter inviare e ricevere caratteri al PIC utilizziamo un programma che si trova sempre nel folder “Collegamento a PIC ” e si chiama **pic**. Quando fate partire **pic** si stabilisce una connessione attraverso la porta seriale tra il vostro schermo/tastiera ed il PIC.

Per fare il modo che la comunicazione avvenga nel modo corretto i parametri del terminale devono essere definiti correttamente (ad esempio COM1, come porta per comunicare, e 9600 baud per la velocita` di comunicazione). L’insieme di specifiche su come PIC e porta comunicano e` il protocollo di comunicazione.

La porta che utilizzeremo (1/2)

- La porta che useremo nella programmazione del microprocessore e` una porta seriale a 9 pin che utilizza il protocollo di trasmissione RS-232 ... vediamo cosa significa :

- una porta seriale utilizza una sola linea per inviare i dati che vengono tramessi un bit dopo l'altro – quindi uno dei pin della porta corrisponde alla linea che trasmette i dati. Nel nostro caso il pin che trasmette i dati al PIC e` il C7.

- il protocollo **RS-232 e' full-duplex asincrono**.

Full duplex significa che i dati vengono ricevuti e trasmessi su due linee diverse, quindi un secondo pin e' dedicato alla linea che riceve i dati (nel nostro caso C6).

Asincrono specifica la logica con cui i due devices devono comunicare. Nel caso di trasmissione asincrona la comunicazione avviene come una conversazione tra due persone : CPU indirizza ad es. il terminale, il terminale dice ci sono, la CPU invia un segnale di inizio trasmissione dati, i dati e una segnale di fine trasmissione, ed il terminale risponde ho ricevuto, la CPU chiude la comunicazione con il terminale.
Altri pin possono venire utilizzati per la logica di trasmissione.

La porta che utilizzeremo (2/2)

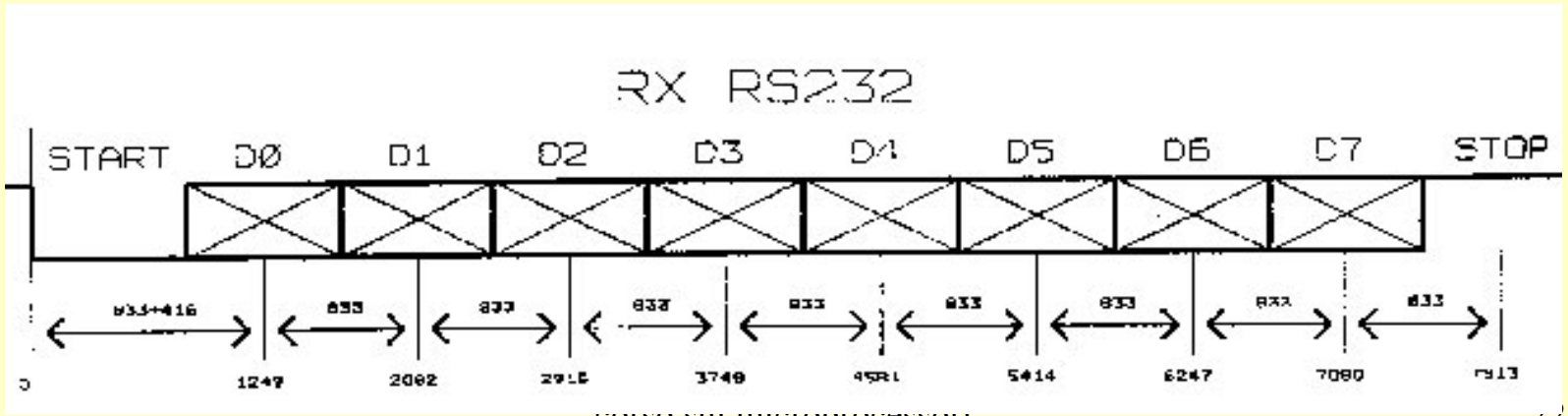
RS-232 specifica anche le caratteristiche HW che devono avere le device in comunicazione ad es.:

Livelli logici : livello 1 deve essere tra [+5,+15] Volts e livello 0 tra [-5,-15] Volts, noi useremo +12, -12 Volts;

Altre specifiche HW input, output load, slew rate ...;

Velocita` di trasmissione in baud=nro di bit/sec RS-232 funziona fino a 115200 e noi utilizziamo 9600.

I dati vengono trasmessi a 8 per volta, 7 bits di dati e l'ultimo bit puo` o meno essere utilizzato come controllo di parita', noi non lo utilizzeremo. Il treno di 8 bits e` preceduto da un bit di start e seguito da un bit di stop. Se la velocita' di trasmissione e' 1200 baud→ 833 μs/bit.



Esempio di utilizzo della PICDEMO

Sul PIC della picdemo board e` precaricato un programma che esegue il test di tutte le periferiche esterne presenti sulla schedina.

Per provare:

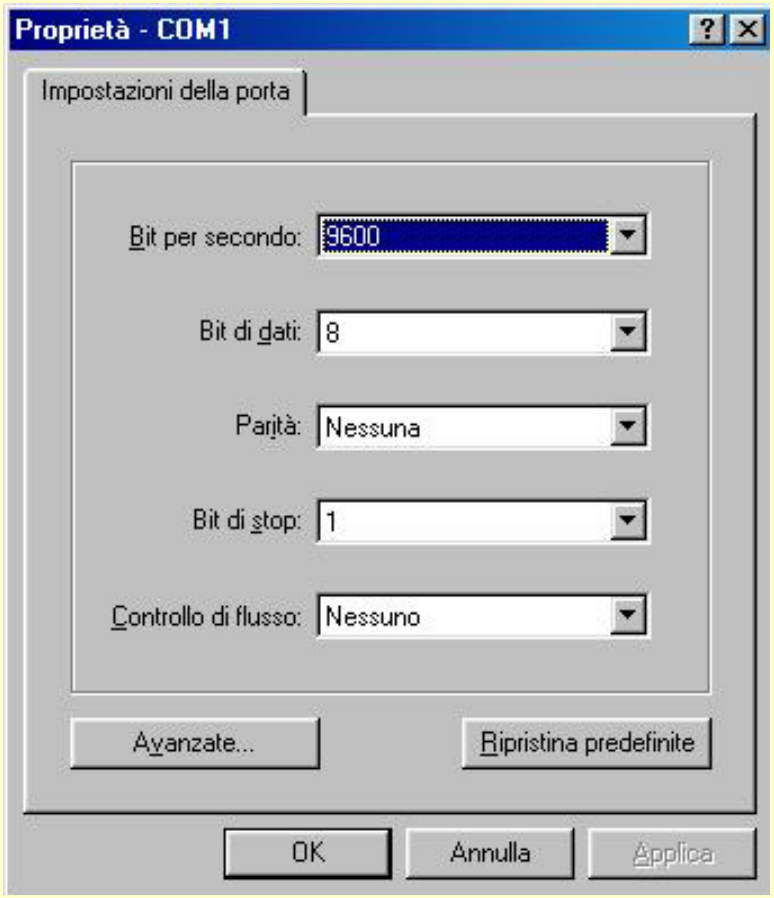
1. Accendere l'alimentatore: e` richiesta una alimentazione duale di +12 e – 12 volts (attenzione che se una delle due manca il DAC si puo` rompere);
2. collegare il cavo seriale dalla PICDEMO al PC (ci sono due porte seriali utilizzare quella superiore);
3. far partire l'emulatore di terminale cliccando due volte sull'icona con la lampadina che si trova all'interno del folder "Collegamento a pic" sul desktop;
4. premete "?" e return sulla tastiera (ATTENZIONE: i tasti che premete non verrano visualizzati sulla tastiera), il PIC risponde con :

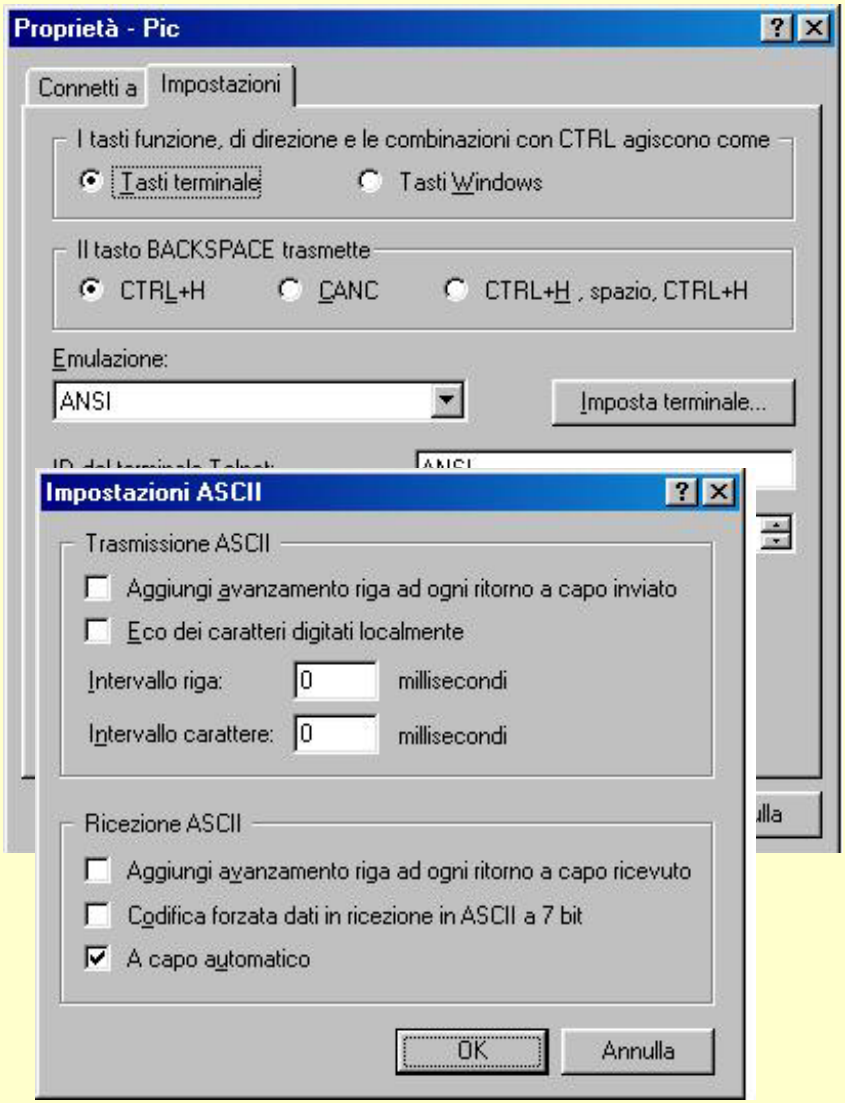
```
F1 ==> LED test
F2 ==> ADC test
F3 ==> DAC test
F4 ==> SWT test
```

NOTA BENE : F1 e` davvero F e poi 1 non il tasto F1 ....

5. date da tastiera i comandi al PIC F1 … F4

Definizioni iniziali per l'Hyperterminal





F1 : accende i 6 LEDS in cascata

F2 : test di due canali dell'ADC -> canale 0 temperatura in Celsius
-> canale 1 tensione in millivolt sul piedino centrale del trimmer
provate a scaldare il sensore di temperatura con le dita o a ruotare il potenziometro

F3 : test del DAC esterno -> si genera una rampa di circa 300 Hz di frequenza (guardarla con l'oscill.)
L'uscita del DAC e` indicata con DAC out sulla scheda di test

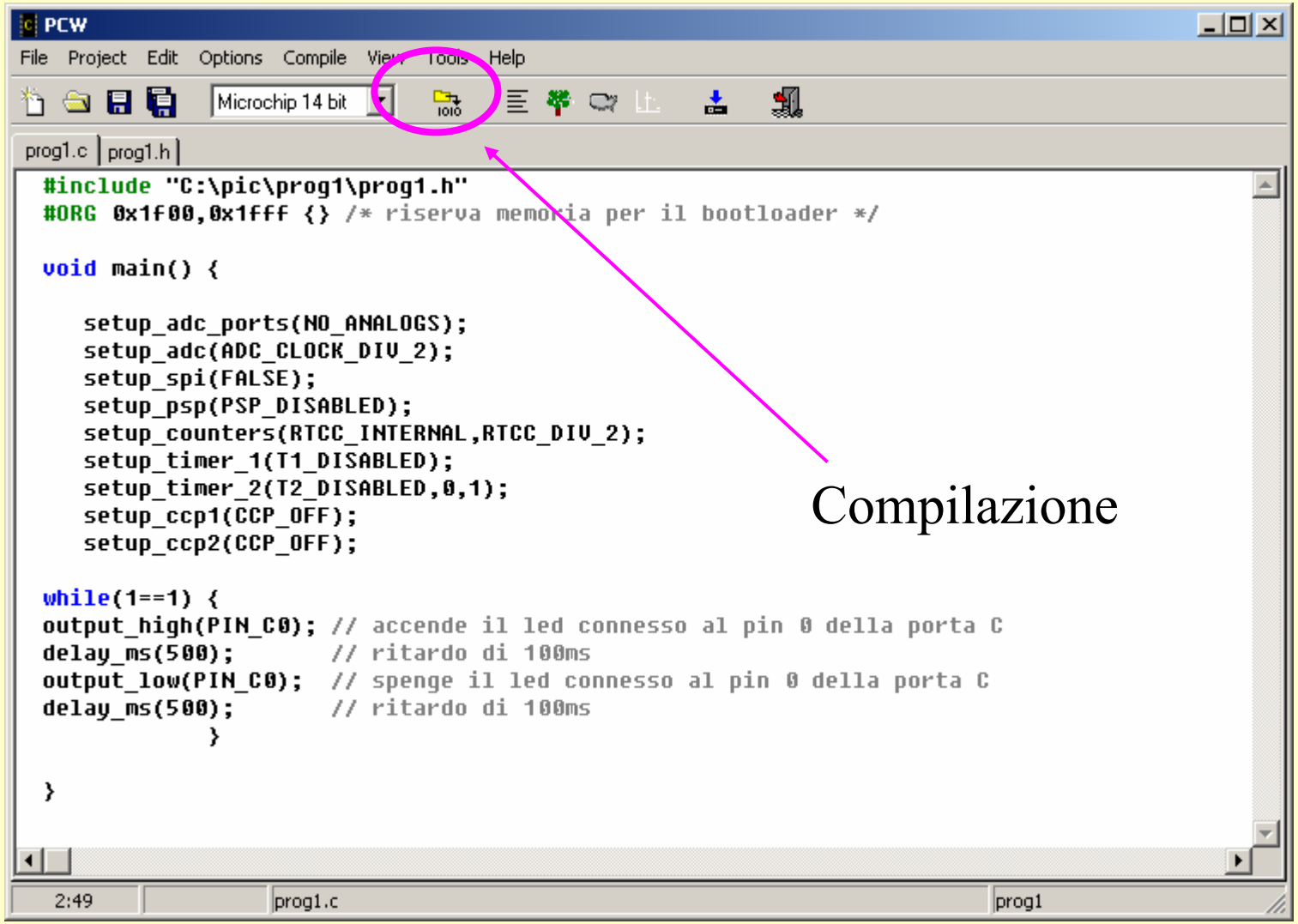
F4 : test dei pulsanti -> premendo i pulsanti si accendono LEDS e si scrive sull'LCD

avete individuato qualche comportamento errato del programma ?

Esempio di flusso completo da un programma C fino alla programmazione del PIC

- Fate partire il compilatore CCS (icona PIC C compiler) sempre nel folder “Collegamento a pic” sul desktop;
- Caricate il progetto:
  project => open => c:\pic\prog1\prog1.pjt
- Il file di progetto contiene i nomi di tutti i files che dovranno essere compilati (nel nostro caso un file .c e uno .h, se volete vederli potete utilizzare open all files)
- Compilate il progetto:
  compile
  Il risultato della compilazione e` un file binario che deve essere caricato sul PIC 16F877 (prog1.hex)

Finestra del compilatore



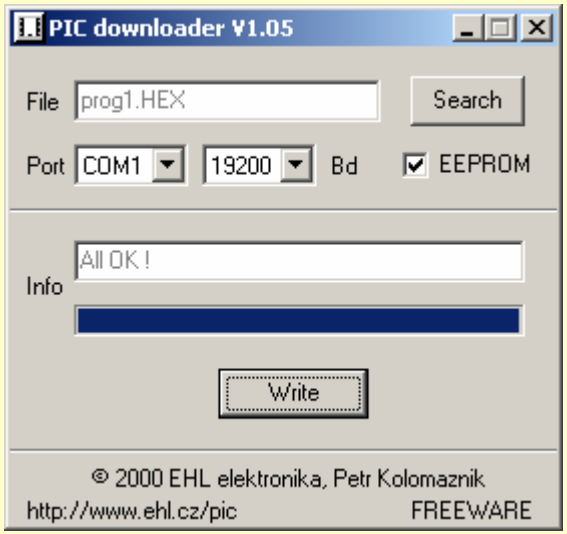
Come programmare il PIC

Per programmare il PIC:

- Disconnettere il terminale del PIC. Il terminale ed il programma che carica i programmi sul PIC utilizzano le medesime linee della porta seriale.
- Far partire il programma per programmare (bootloader) sempre nel folder "Collegamento a pic".
  Controllare che Port sia su Com1 e che la velocita' di comunicazione sia 19200 Bd
- Selezionare il file .hex da caricare sul PIC
  Tasto search e poi c:\pic\prog1\prog1.hex
- Per cominciare la programmazione:
  - selezionare WRITE sulla finestra del downloader
  - premere subito il tasto Mreset sulla Picdemo board

Alla fine della programmazione, dopo ½ secondo, il PIC comincia ad eseguire il programma automaticamente e si vedra` il LED lampeggiante.

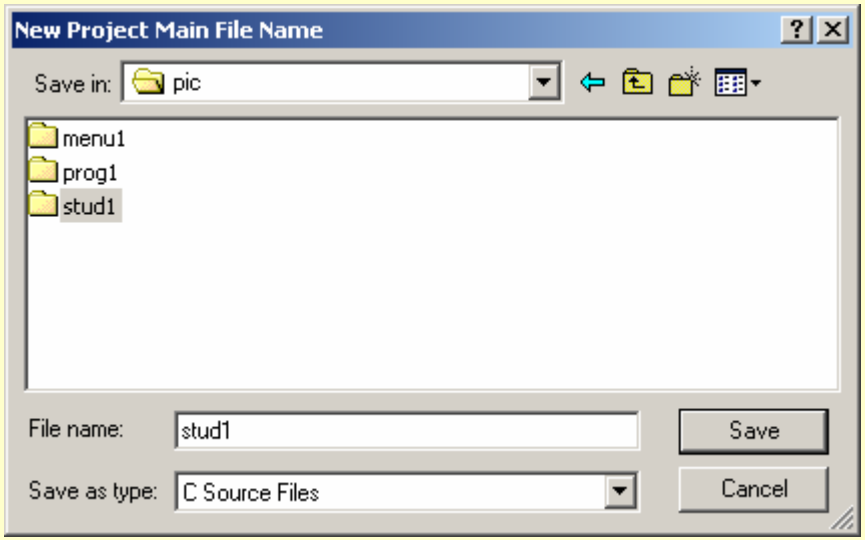
Finestra del PIC downloader



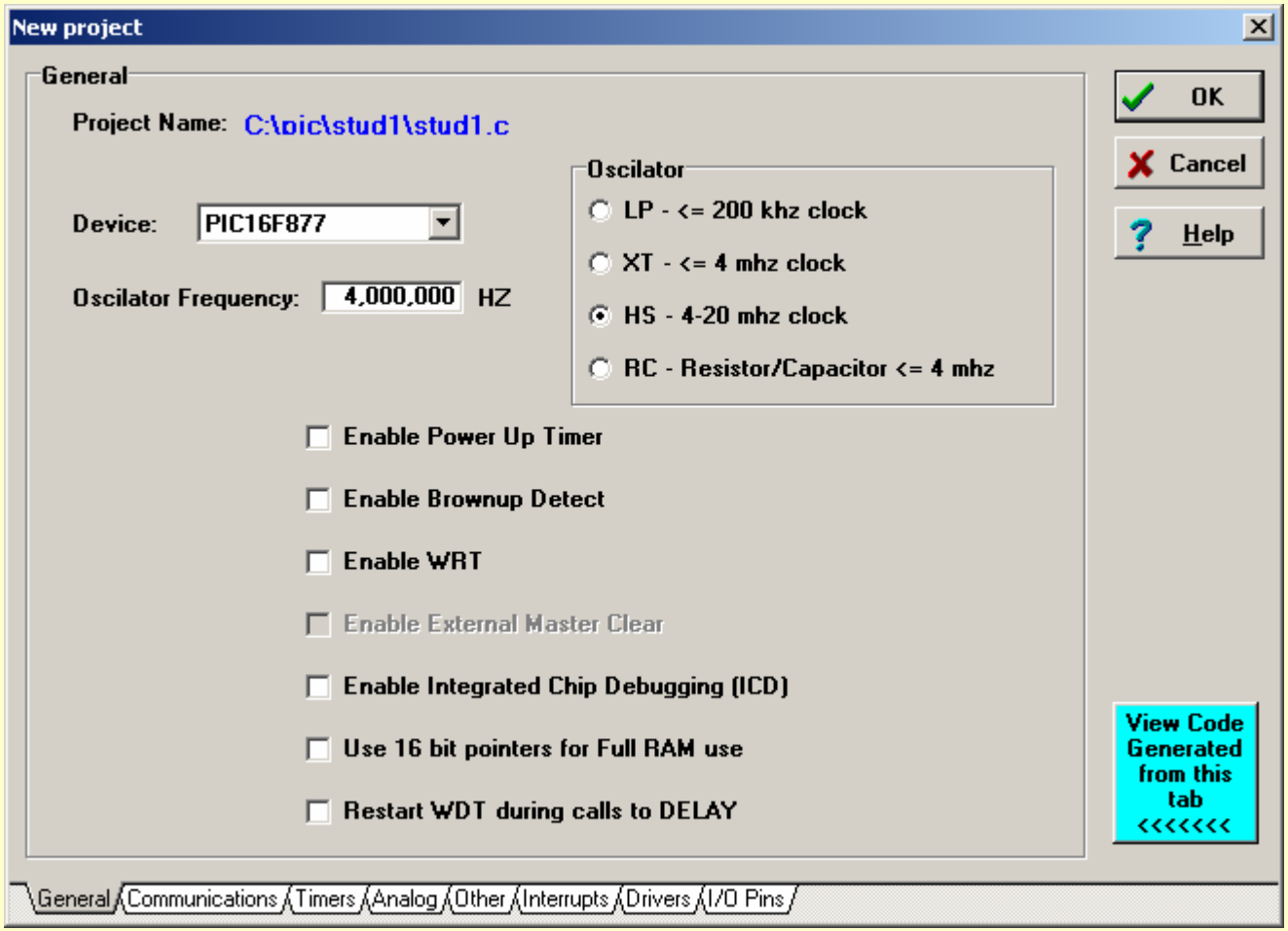
Attenzione : dovete scollegare dalla porta seriale l'emulatore di terminale per poter caricare il programma che utilizza la stessa porta seriale.

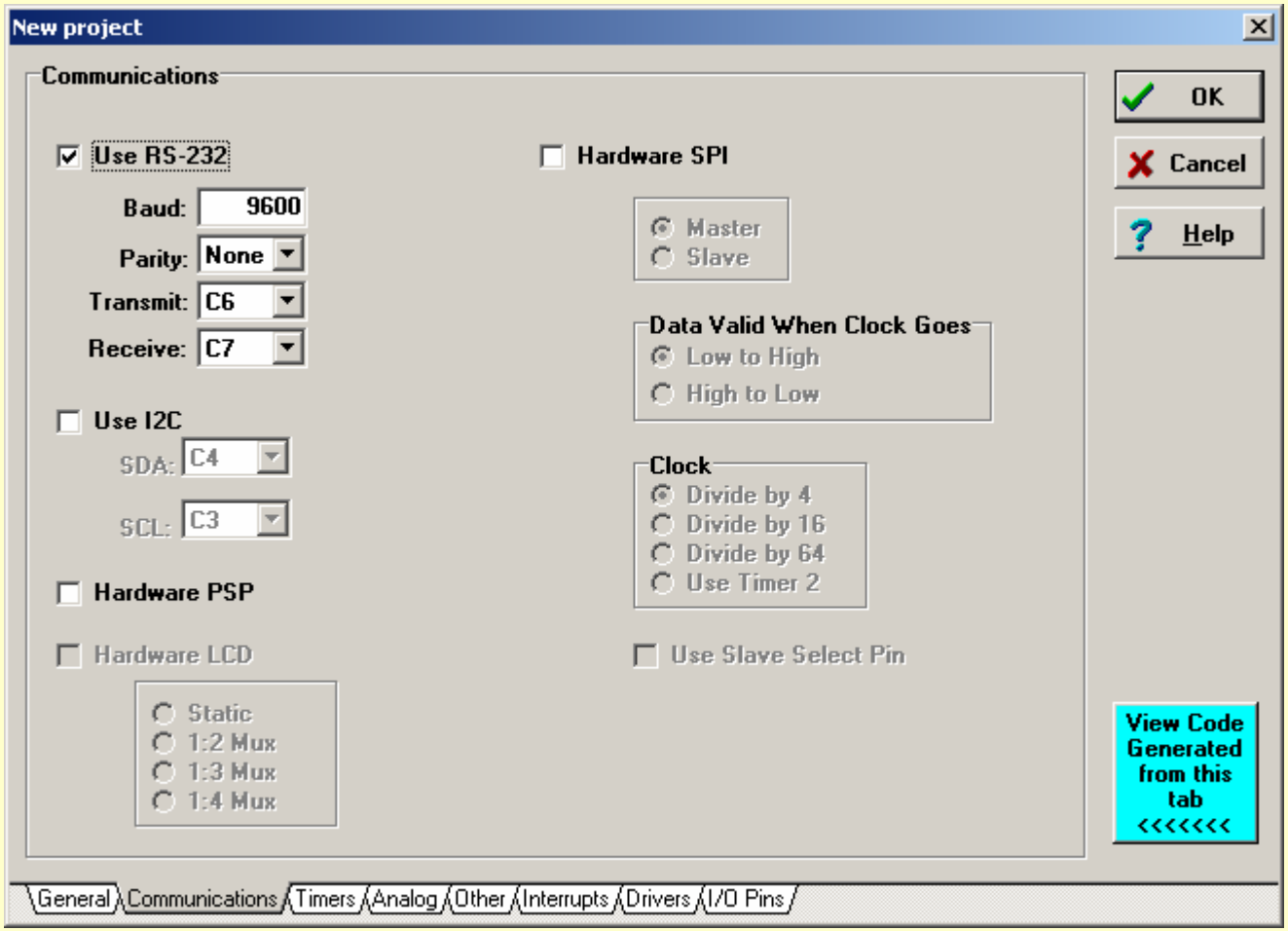
Implementazione di un progetto

1) aprire il compilatore C
2) definite una opportuno folder ‘PIC’ che conterra` tutti i folder per i progetti relativi al pic. Create quindi un’altro folder ‘prog1’ che conterra` il progetto prog1 cioe` tutti file relativi a questo progetto (.c, .h., .hex ....)
3) project => new => pic wizard
4) file name prog1 in prog1 ... (nell’esempio il progetto e` stato chiamato stud1)



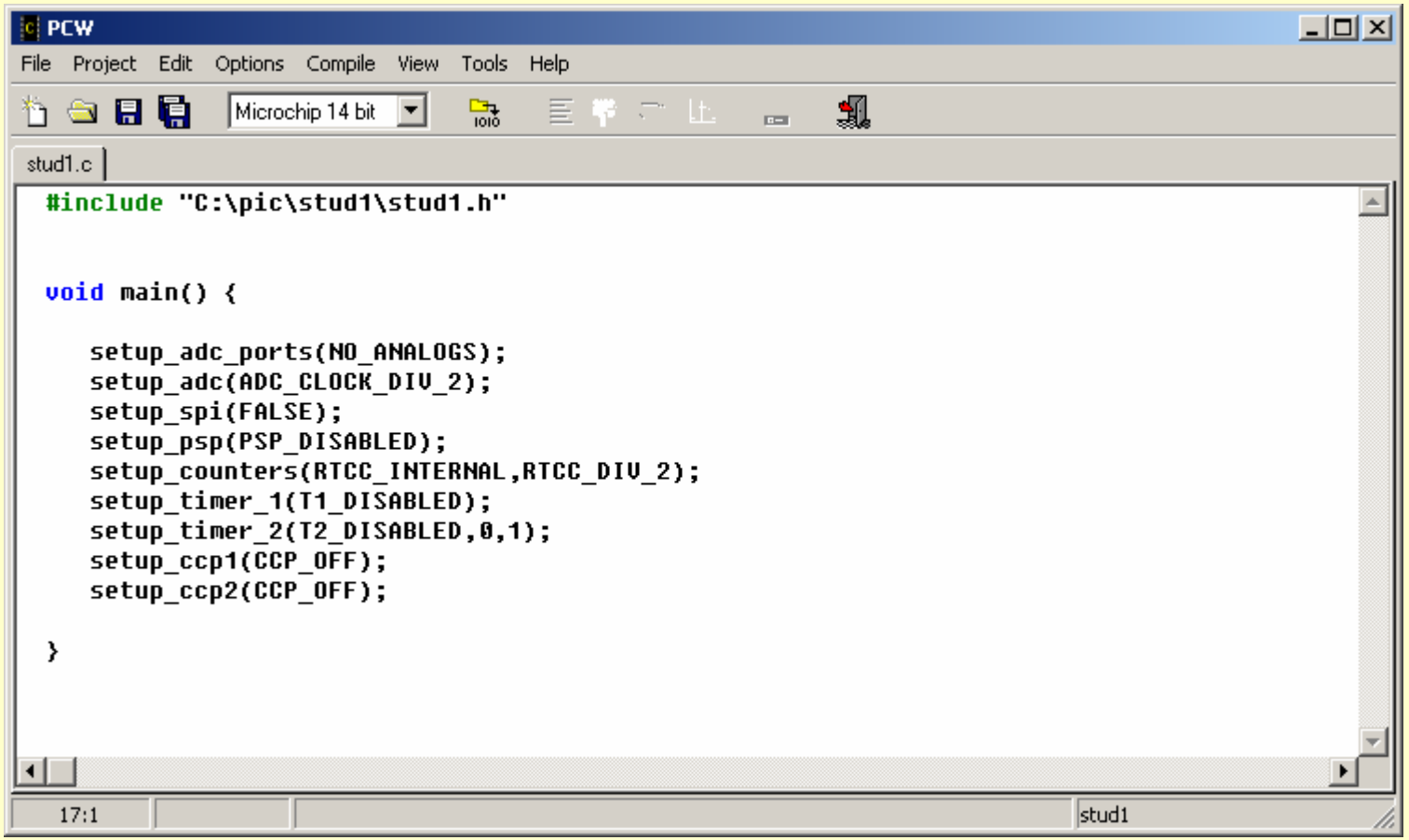
5) definite i parametri del progetto:





in questo progetto non e` necessario cambiare altri parametri …
=> premere OK

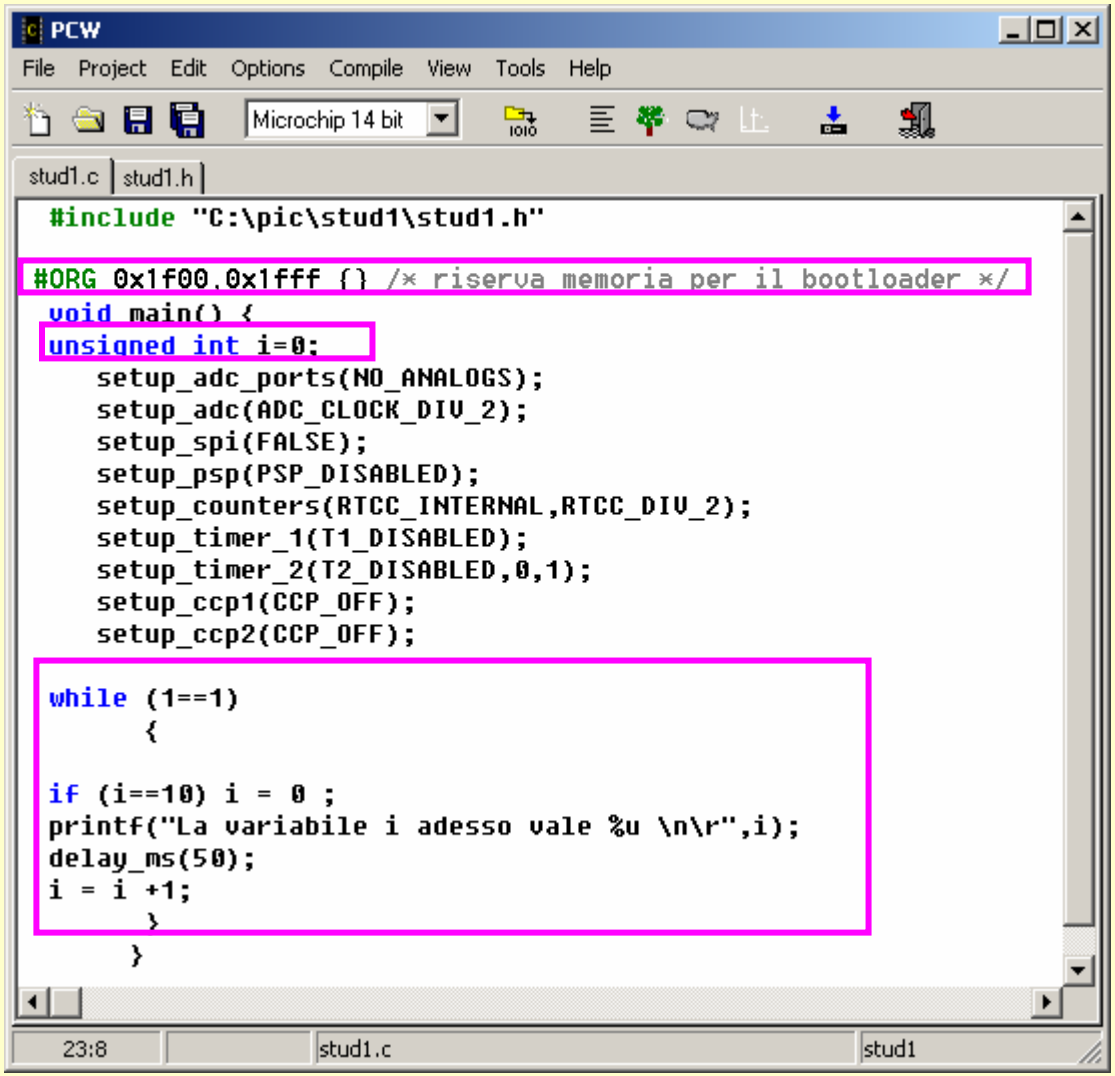
il compilatore si presenta cosi` :



queste righe sono generate automaticamente (setup periferiche).

aggiungiamo adesso il codice del nostro programma :
vogliamo scrivere su un emulatore di terminale una stringa.

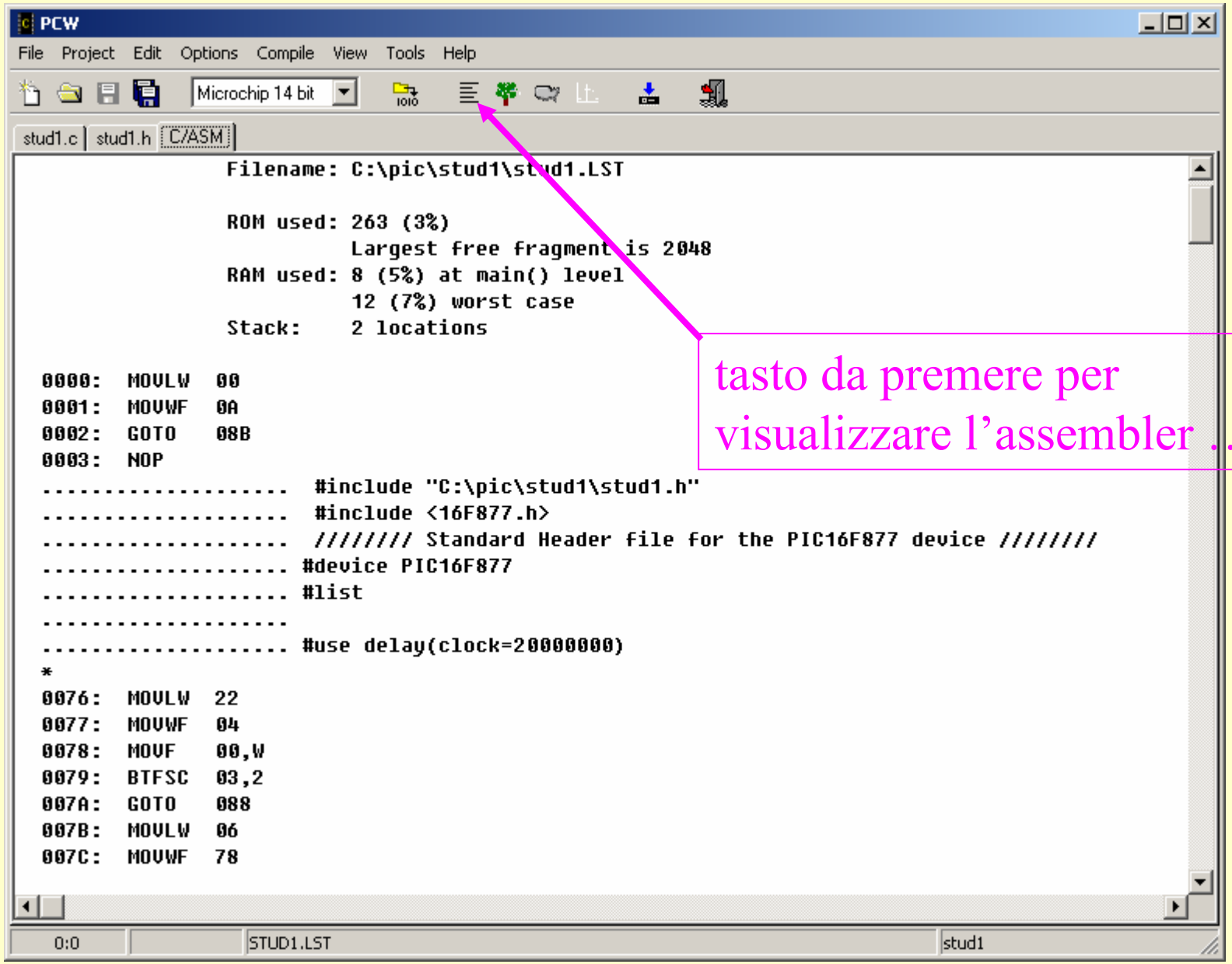
Attenzione:
Questa riga
riserva memoria
Per il bootloader:
metterla sempre



Compilate e caricate sul PIC il programa stud1.hex ottenuto.
Il programma scrive dalla porta seriale del PIC alla porta seriale del PC. Per osservare il flusso di dati sulla porta seriale, utilizziamo l'hyperterminal emulatore (icona lampadina).
Se le scritte non si vedono, puo' darsi che il PIC debba essere resettato.

Provate a vedere la conversione in assembler di questo semplice listato C ….

sono 10 pagine circa ….


PCW
File Project Edit Options Compile View Tools Help
Microchip 14 bit
stud1.c stud1.h C/ASM
Filename: C:\pic\stud1\stud1.LST
ROM used: 263 (3%)
Largest free fragment is 2048
RAM used: 8 (5%) at main() level
12 (7%) worst case
Stack: 2 locations
0000: MOVLW 00
0001: MOVWF 0A
0002: GOTO 08B
0003: NOP
..... #include "C:\pic\stud1\stud1.h"
..... #include <16F877.h>
..... //////// Standard Header file for the PIC16F877 device ////////
..... #device PIC16F877
..... #list
.....
..... #use delay(clock=20000000)
*
0076: MOVLW 22
0077: MOVWF 04
0078: MOVF 00,W
0079: BTFSC 03,2
007A: GOTO 088
007B: MOVLW 06
007C: MOVWF 78
0:0
STUD1.LST
stud1
tasto da premere per visualizzare l'assembler ...

Analisi del codice

```
#include "C:\pic\stud1\stud1.h"


void main() {
unsigned int i=0;
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

while (1==1)
      {

if (i==10) i = 0 ;
printf("La variabile i adesso vale %u \n\r",i);
delay_ms(50);
i = i +1;
      }
     }
```

Esercizio (da consegnare con commmenti):

scrivere un programma che:

1) accende tutti e 6 i LEDS per 200 ms
2) scrive su seriale “tutti accesi”
3) ne spegne uno per volta ogni 100 ms
4) scrive su seriale “tutti spenti”
5) aspetta 200 ms
6) riesegue il ciclo in loop

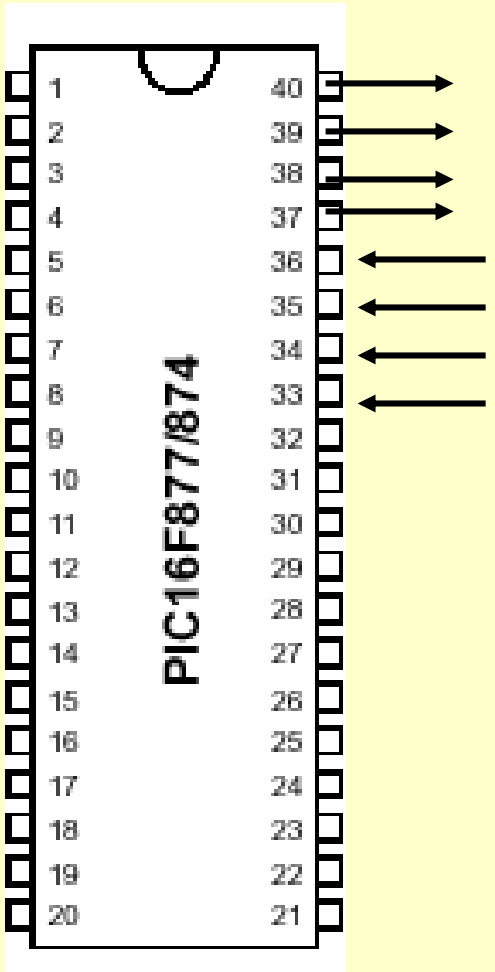
Dove sono definiti PIN_C0, PIN_C1 ... ?

Le “porte” per entrare ed uscire dal PIC….

• Ogni pin di ogni porta può essere configurato come ingresso o come uscita del microcontrollore.

• Il microntrollore conosce la direzione di ogni pin di ogni porta dal valore del registro di stato indicato con il nome mnemonico TRISx. Ad esempio la direzione del bit 0 della porta B è definito dal valore del bit 0 del registro di stato TRISB.

• 1 logico = = INGRESSO 0 logico = = USCITA ad esempio: TRISB = 0Fh allora i bit 0,1,2,3 sono definiti come ingressi e 4,5,6,7 come uscite.



Le “porte” per entrare ed uscire dal PIC….

• Ogni pin di ogni porta può essere configurato come ingresso o come uscita del microcontrollore.

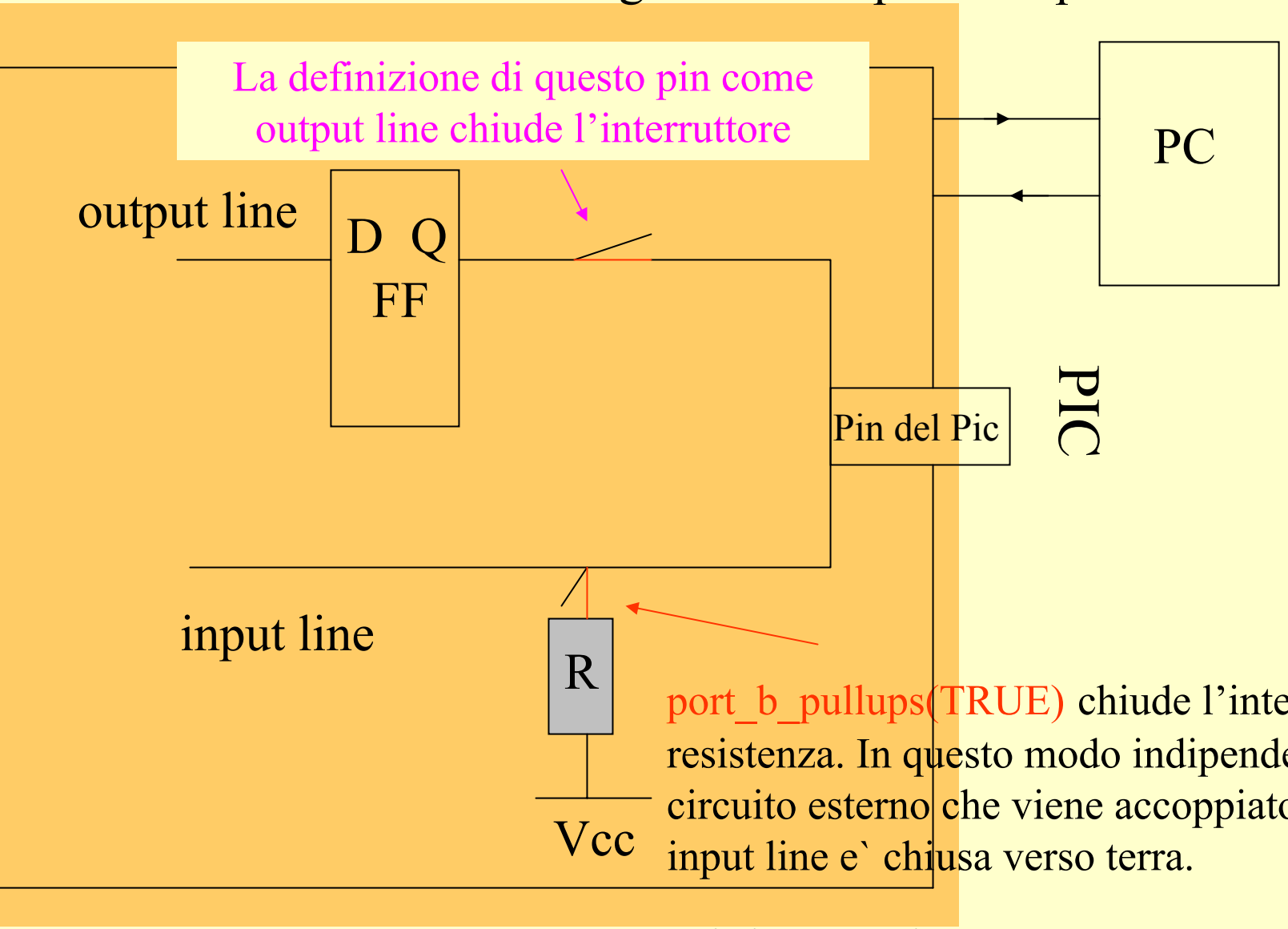
• Il microntrollore conosce la direzione di ogni pin di ogni porta dal valore del registro di stato indicato con il nome mnemonico TRISx. Ad esempio la direzione del bit 0 della porta B è definito dal valore del bit 0 del registro di stato TRISB.

• 1 logico = = INGRESSO 0 logico = = USCITA ad esempio: TRISB = 0Fh allora i bit 0,1,2,3 sono definiti come ingressi e 4,5,6,7 come uscite.

• Un pin configurato come ingresso può essere collegato a massa da un circuito ad alta impedenza mentre un pin configurato come uscita è collegato ad un circuito di latch.

• Ad ogni porta è associato anche un registro DATI indicato con PORTx (PORTB per la porta B). Per ogni pin configurato come uscita il valore del bit corrispondente indica il valore inviato al pin dal microcontrollore. Per ogni pin configurato come ingresso il valore del bit indica il valore rivelato sul pin.

Schema di un pin di una porta del PIC in configurazione input o output



port_b_pullups(TRUE) chiude l'interruttore sulla resistenza. In questo modo indipendentemente dal circuito esterno che viene accoppiato al pin, la input line e` chiusa verso terra.

Metodi di utilizzo delle porte di I/O

Per leggere o scrivere un bit di una particolare porta si possono utilizzare vari metodi:

1.funzioni interne al compilatore CCS :

output_high(<pin name>);

`output_high(PIN_C0)`mette 1 sul pin 0 della porta C

output _low(<pin name>);

input(<pin name>);

Queste funzioni sono descritte nell' help online del compilatore C.

Metodi di utilizzo delle porte di I/O

2. Scrittura diretta negli opportuni registri:
   set_tris_c(0xff) => tutti i pin della porta C configurati come input
   0x0 tutti output, 0x11 bit 0 e 4 come input
   bit_test(registro,bit);
   bit_clear(registro,bit);
   bit_set(registro,bit);
3. mappatura del registro:

   richiede set_tris_x(value) per definire se i bit sono ingressi o uscite: ad esempio set_tris_b(0xff).

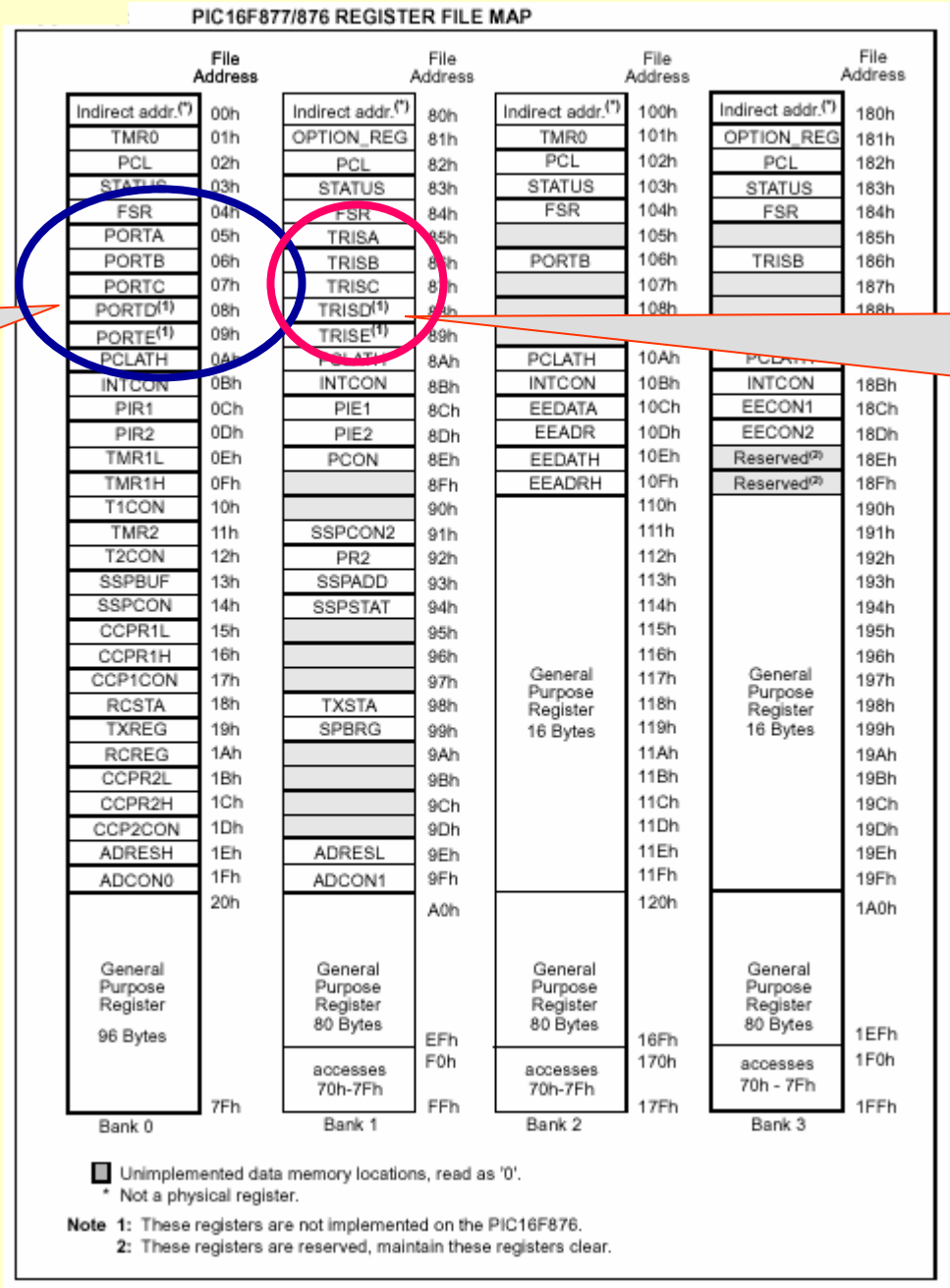
   #byte myport = 0x06 porta B (0x07 per la C, 0x08 per la D 0x09 per la E) definisce la posizione del registro dove scrivo o leggo.

   invalue = myport ; per leggere i valori di pin definiti come ingressi

   myport = outvalue; per assegnare i valori ai pin definiti come uscite

Indirizzi per i registri delle porte

Registri per definire la direzione delle porte



Esempio 1

```
#include "C:\pic\stud2\stud2.h"


void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

// abilito i pullup sulla porta B
port_b_pullups(TRUE);


while(1==1) {
if (!input(PIN_B0)) output_high(PIN_C0); else output_low(PIN_C0);
if (!input(PIN_B1)) output_high(PIN_C1); else output_low(PIN_C1);
if (!input(PIN_B2)) output_high(PIN_C2); else output_low(PIN_C2);
            }
}
```

I nomi dei PIN sono definiti nel file .h del pic

Esempio 2

```
#include "C:\pic\stud3\stud3.h"

#byte port_b = 0x6
#byte port_c = 0x7

void main() {


   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

// abilito i pullup sulla porta B
port_b_pullups(TRUE);
//definisco i bits della porta B tutti in input
// pulsanti: su RB0,EB1,RB2

set_tris_b(0xff);

// definisco i bits della porta C 0-6 in output  il 7 in input
set_tris_c(0x0);

while(1==1) {

if (bit_test(port_b,0)) bit_clear(port_c,0) ; else bit_set(port_c,0);
if (bit_test(port_b,1)) bit_clear(port_c,1) ; else bit_set(port_c,1);
if (bit_test(port_b,2)) bit_clear(port_c,2) ; else bit_set(port_c,2);
            }
}
```

Errore dovrebbe essere: set_tris_c(0x80)

Esempio 3

```
#include "C:\pic\stud4\stud4.h"
#byte port_b = 0x6
#byte port_c = 0x7

void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

// abilito i pullup sulla porta B
port_b_pullups(TRUE);
//definisco i bits della porta B tutti in input
// pulsanti: su RB0,EB1,RB2

set_tris_b(0xff);

// definisco i bits della porta C 0-6 in output  il 7 in input
set_tris_c(0x80);

while(1==1)        {
// la pressione del tasto RB1 fa accendere tutti e 6 i leds

 if (bit_test(port_b,1)) port_c = 0 ; else port_c = 0x3f;

                    }
            }
```

Esercizio (da consegnare con commenti):

scrivere un programma che :

accende un led in piu` ogni volta che si preme e rilascia il pulsante RB0

spegne un led in piu` ogni volta che si preme e rilascia il pulsante RB1

spegne tutti i led quando si preme e rilascia il pulsante RB2.

Attenzione: dovete fare in modo di capire quando un pulsante viene premuto e rilasciato.

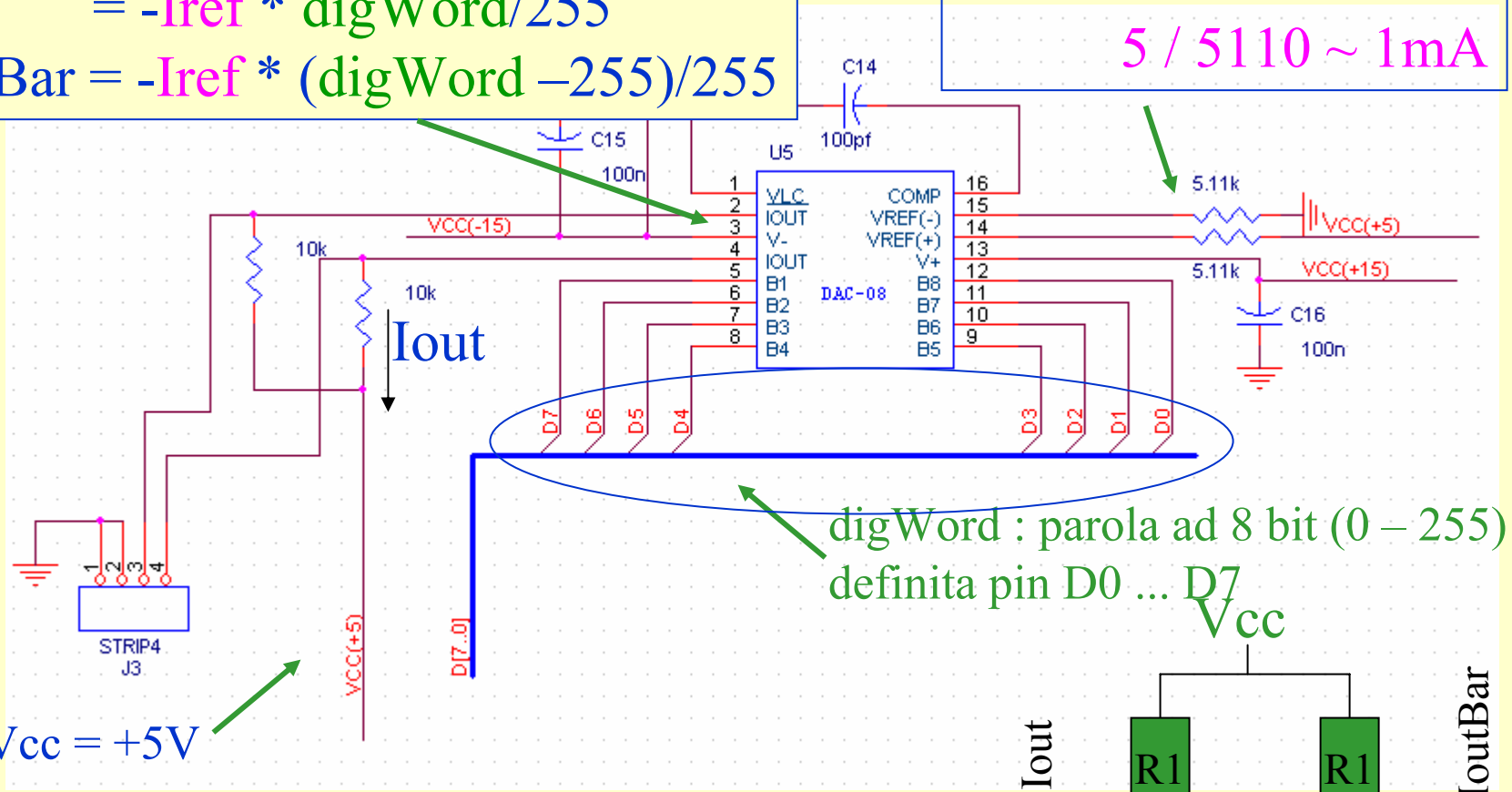
Utilizzo del DAC esterno (DAC 08)

Iout = -Iref * digWord/255

IoutBar = -Iref * (digWord –255)/255

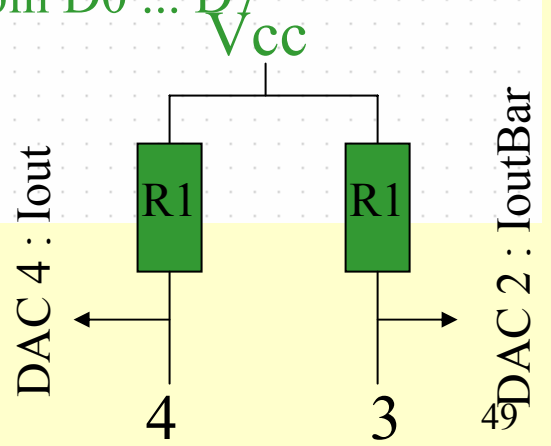
Iref = Vcc/R =

5 / 5110 ~ 1mA



digWord : parola ad 8 bit (0 – 255)
definita pin D0 ... D7

Vcc = +5V



Vout(4) = Iout R1 + Vcc =
(- (10000 * 0.001)/255 * digWord) + 5 Volts

Esempio generatore di rampa o sinusoide:

```
#include <math.h>
#include "C:\pic\dac\dac-test.h"
#ORG 0x1f00,0x1fff {} /* riserva memoria per il bootloader */
#byte port_D=8

void main() {
   unisgned int i;
   float x;
   int dac_value;


   set_tris_d(0x0);

   port_D = 0;
   while(1==1) {
   for (i=0;i<256;i++) {
   port_D = i;

   // x = sin(0.024639 * i);
   //dac_value =  floor (( x+1) * 127.5);
   //port_D = dac_value;


                     }
               }


}
```

Floor(x): ritorna l'intero piu` vicino non piu` grande di x

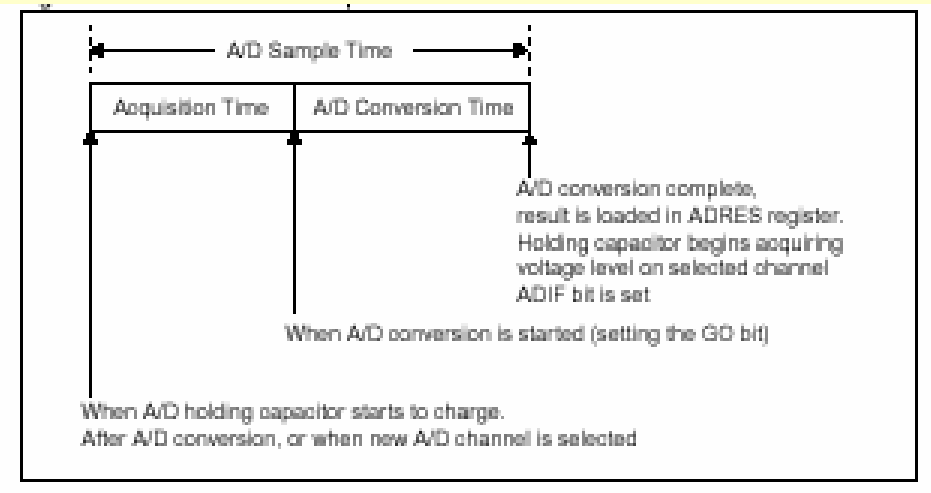
Perche' la sinusoide ha frequenza bassissima ?

Esercizio (da consegnare con commenti):

scrivere un programma che :

genera un'onda triangolare che abbia ampiezza massima e minima pari a +5 e –5 Volt rispettivamente.

Uso del ADC integrato

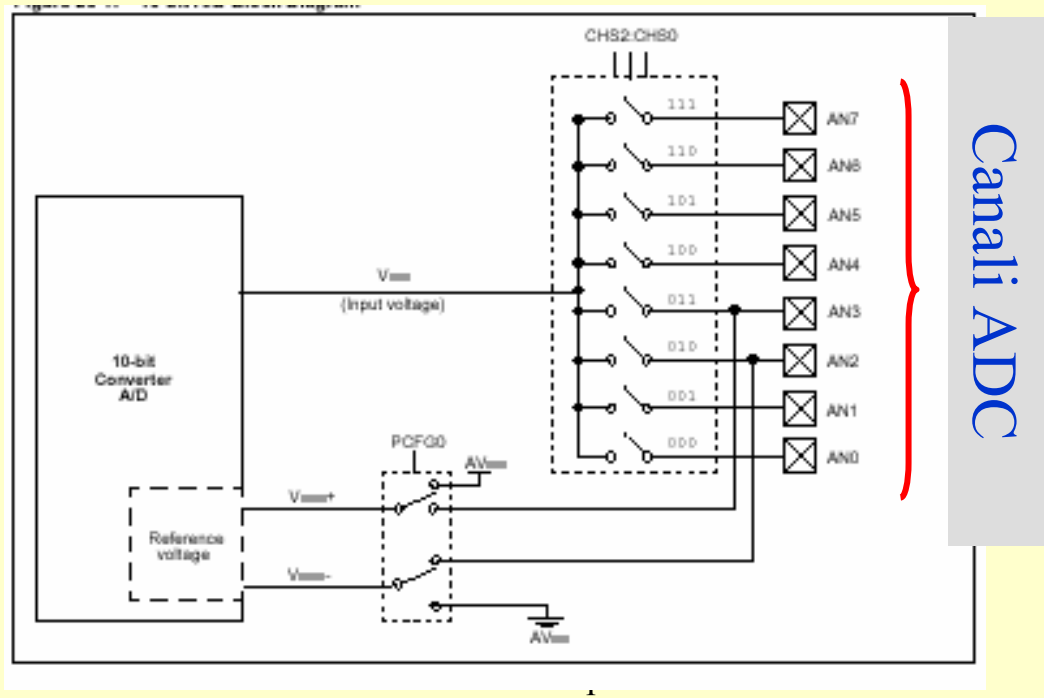


L'acquisizione avviene in due passaggi: prima si carica un condensatore (acquisition time) SAMPLE e il condensatore viene chiuso su alta impedenza ed il valore ai suoi capi viene misurato (conversion time) HOLD. La fase di conversione utilizza una tecnica ad opprossimazioni successive, con un suo ciclo di clock, piu` lento di quello del quarzo e ottenuto da questo per divisione.
setup_adc(ADC_CLOCK_DIV_8);

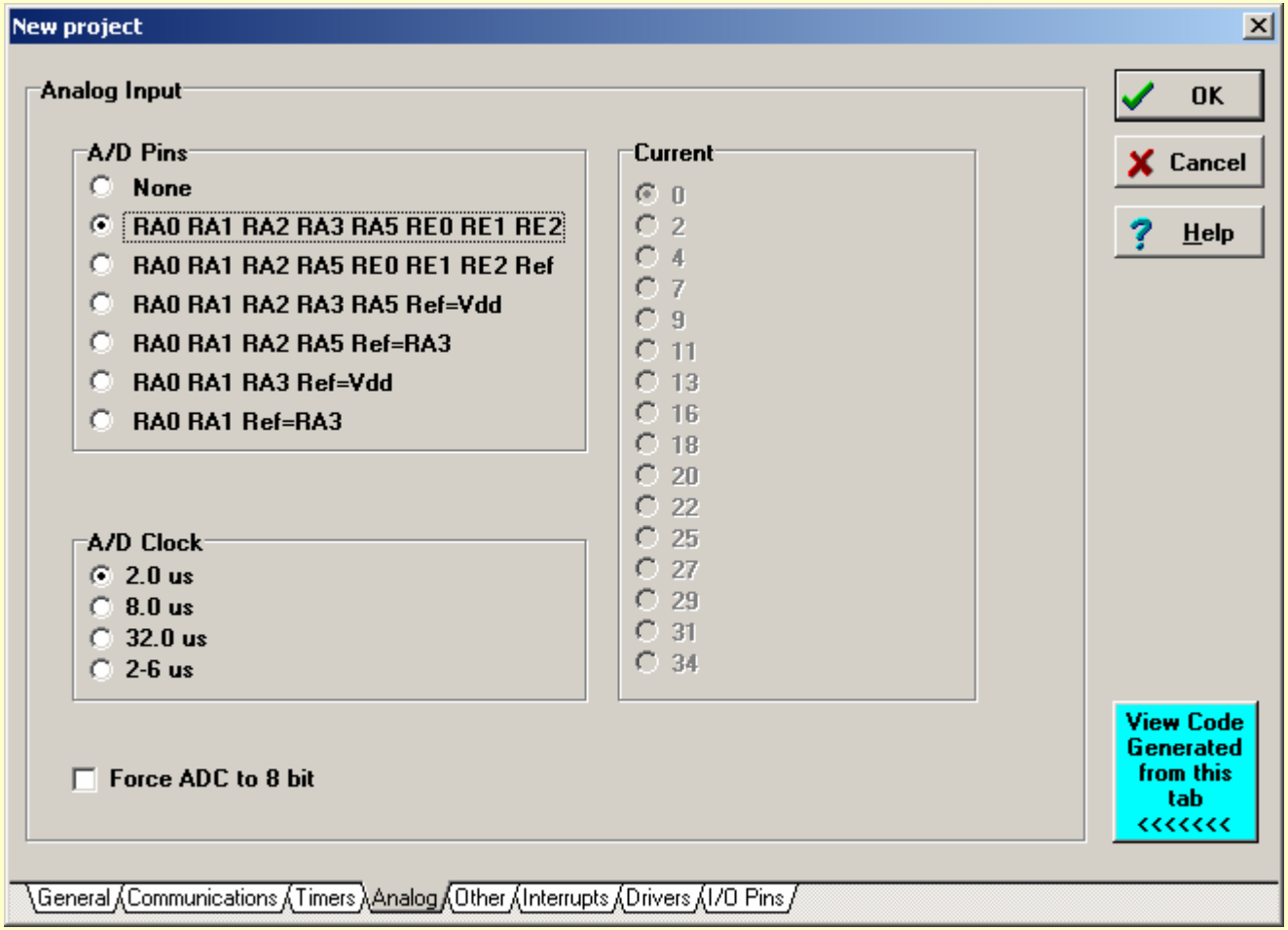
Tra una acquisizione e un`altra deve passare il tempo necessario per la carica del condensatore: ~50 usec.

Uso del ADC integrato

L'ADC e` a 10 bit ed e` collegato ad un multiplexer analogico a 8 ingressi: prima di leggere l'ADC e` quindi necessario scegliere il canale che si vuole acquisire **set_adc_channel(x).** Questa funzione imposta il valore di tre bit del registro ADCCON0 che sono dedicati alla selezione del canale ADC. Ad esempio il primo programma che abbiamo utilizzato acquisiva canale 0 e canale 1. La relazione pin del PIC-canale ADC e` fissata: RA0/CH0, RA1/CH1, RA2/CH2, RA3/CH3, RA5/CH4, RE0/CH5, RE1,CH6, RE2/CH7.



Nel setup selezionare le porte analogiche volute, i Vref, il clock dell'ADC per la conversione



Esempio: lettura del potenziometro, ADC canale1

```
#include "C:\pic\adc\adctest.h"

unsigned long value;
float voltvalue;
void main() {

   setup_adc_ports(ALL_ANALOG);
   setup_adc(ADC_CLOCK_DIV_8);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

 port_b_pullups(TRUE);
// voglio leggere il canale 0 dell'ADC
set_adc_channel(1);
while(1==1){
while ( !input(PIN_B1) ) {

delay_ms( 50);
   value = read_adc();
   voltvalue = value * 0.00489;
   printf("A/D value = %lu || Volt value = %1.2f \n\r", value,voltvalue);}


} }
```

value = read_adc();

5/1023 = 0.00489

Visualizzare il risultato sull'HyperTerminal …

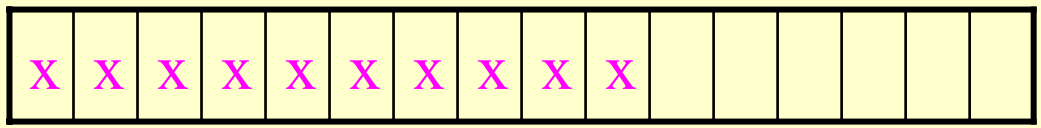
Attenzione tutte le funzioni che leggono canali di ADC devono essere modificate :

Il nuovo sistema operativo in combutta con un baco del compilatore ha prodotto un errore nelle funzioni di esempio che vi sono state proposte dove si utilizzava l'ADC.

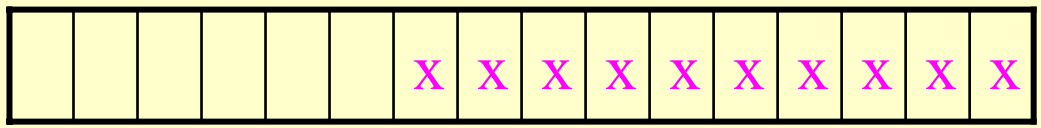
Come si produceva l'errore ?

Problema di allineamento dei 10 bits letti su una variabile composta da 16 bits

Come era :



Come deve essere :



Esempio: lettura del potenziometro, ADC canale1

```
#include "C:\pic\adc\adctest.h"

unsigned long value;
float voltvalue;
void main() {

   setup_adc_ports(ALL_ANALOG);
   setup_adc(ADC_CLOCK_DIV_8);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

 port_b_pullups(TRUE);
// voglio leggere il canale 0 dell'ADC
set_adc_channel(1);
while(1==1){
while ( !input(PIN_B1) ) {

delay_ms( 50);
   value = read_adc();
   voltvalue = value * 0.00489;
   printf("A/D value = %lu || Volt value = %1.2f \n\r", value,voltvalue);}


} }
```

#byte adcon1=0x9f

bit_set(adcon1,7);

Istruzione per definire il tipo di formato del dato in uscita

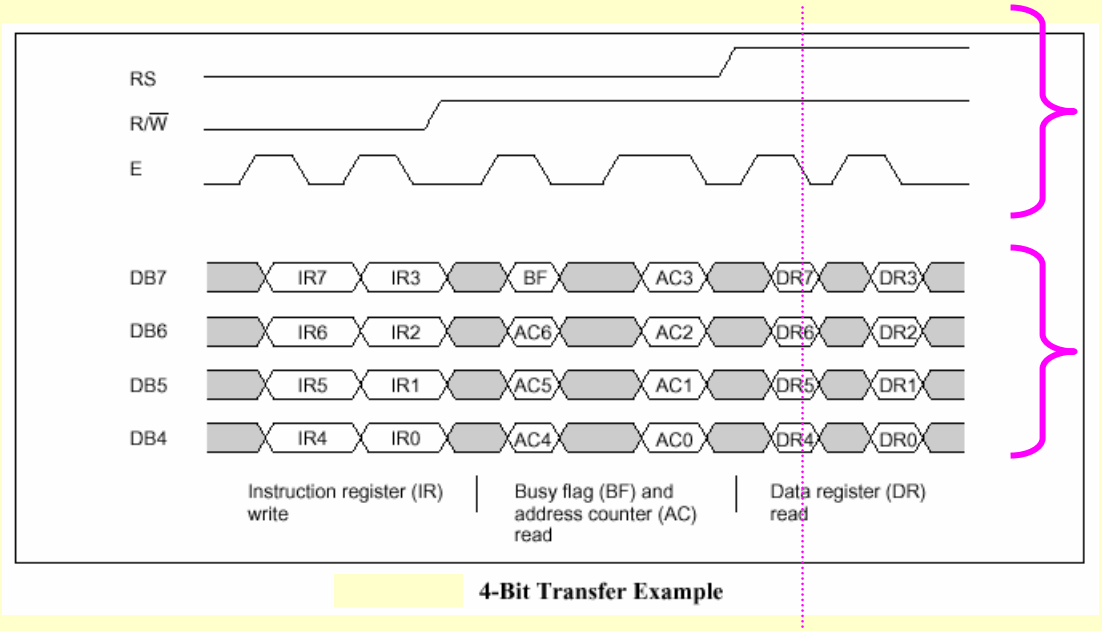
Esercizio (da consegnare con commenti):

Sulla board e` saldato un sensore di temperatura, che genera in uscita una tensione pari a 10 mV/C (es 250 mV = 25 gradi).

Scrivere un programma che stampa sul terminale
la temperatura media di 10 letture eseguite ogni 100 ms.

Uso del display LCD 16x2

Il display a cristalli liquidi montato sulla scheda e` di tipo intelligente: LCD vero e proprio viene controllato da un complesso chip che si occupa di gestire il protocollo di comunicazione. Questo chip e` l'Hitachi HD44780 (vedi documentazione sul web)



portE
bit 2,1,0

portB
bit 7,6,5,4

Per scrivere un carattere sull'LCD si inviano all'LCD due gruppi di 4 bits ciascuno, cioe` 8 bits, che rappresentano il codice ascii del carattere da visualizzare.

Per semplicita`, utilizzeremo l'LCD facendo ricorso ad una libreria di funzioni, che si trovano nel file **c:\Pic\disp16x2.C**, quindi quando volete utilizzare l'LCD dovete sempre includere questo file.

Funzioni di uso comune:
init_lcd(); => da eseguire prima di tutto, resetta l'LCD
printf(disp_lcd, "FRANCO"); => si puo' utilizzare printf anche per scrivere sull'LCD. printf passa la stringa "FRANCO" alla funzione disp_lcd che la scrive sull'LCD
lcd_row_col(1,0); => sposta il cursore nella posizione 1,0 (inizio seconda riga)
clr_home(); => cancella l'LCD e porta il cursore a 0,0

Esempio uso LCD

```
#include "test-lcd.h"
#include "c:\Pic\Disp16x2.c"

#ORG 0x1f00,0x1fff {} /* reserves memory for the bootloader */

void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

   disable_interrupts(GLOBAL);
   port_b_pullups(TRUE);

   init_lcd();
   while(1==1) {
   printf(disp_lcd, "FRANCO");
   lcd_row_col(1,0);
   printf(disp_lcd,"SPINELLA");
   delay_ms(500);
   clr_home();
            }
}
```

attenzione: l'LCD usa la portaE => non si puo' usare ALL_ANALOG

Esercizio (da consegnare con commenti):
Scrivere un programma che legge la tensione applicata alla resistenza e quando questa e` maggiore di un valore di soglia, definito da voi, stampa sull'LCD la temperatura ambiente media su 100 misure.

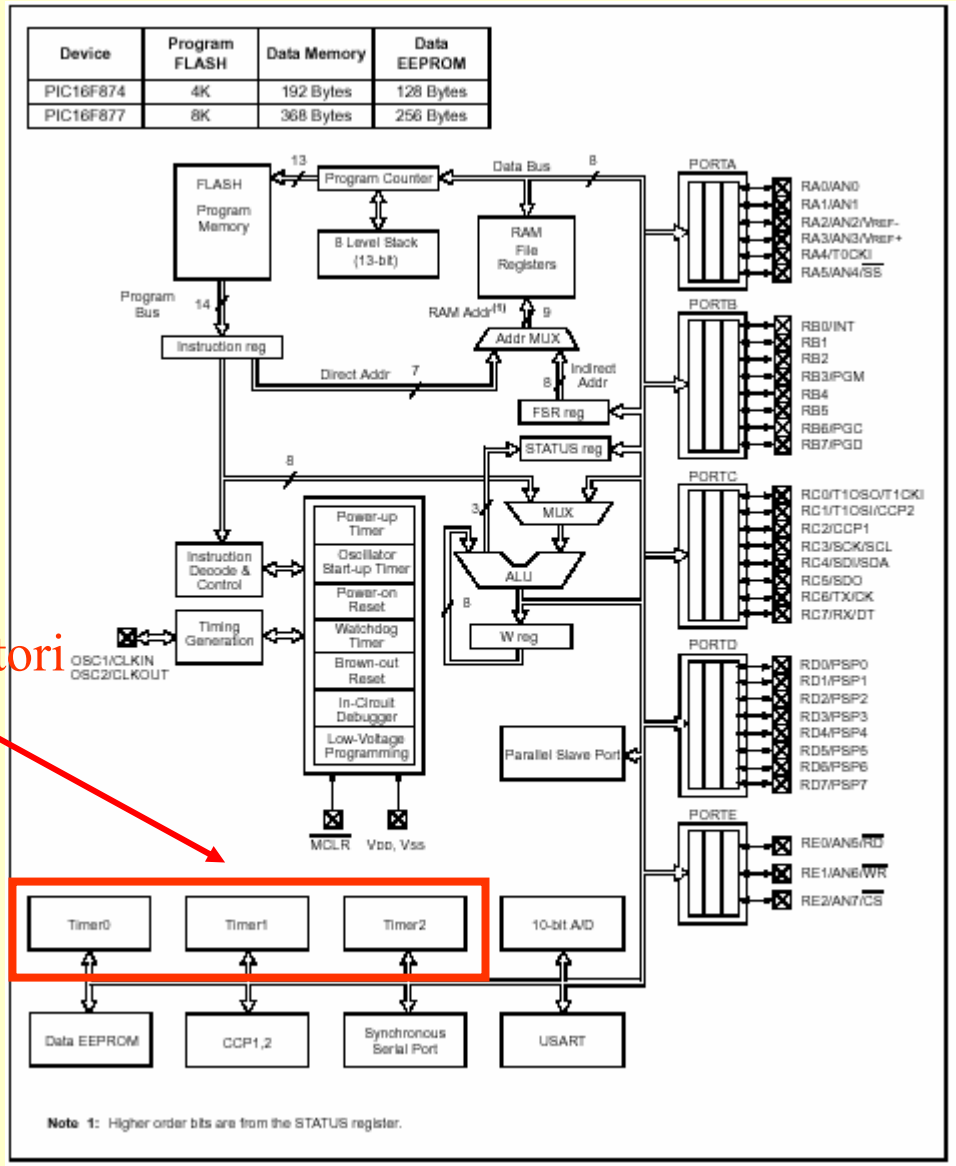
Problemi trovati dalla volta scorsa:

Il compilatore non vuole nomi di file o folder con molti caratteri (di sicuro 13 sono troppi...). Consiglio: utilizzatene al massimo otto.

Il file da includere per LCD deve contenere il path: C:\Pic\Display16x2.c

Architettura PIC 16F877

3 Timer = 3 contatori



I Timer

I timer sono precisi contatori, che possono essere configurati per incrementarsi su fronti di segnali esterni o su fronti di un segnale interno ottenuto dal clock. Il segnale che incrementa il contatore puo' eventualmente essere "prescalato", cioe` incrementato solo per multipli del periodo del clock. Il clock interno e` gia diviso per 4.

Il valore del prescaler viene definito con la funzione:

setup_counters(RTCC_INTERNAL, …..);

Ad esempio: setup_counters(RTCC_INTERNAL,RTCC_DIV_8);
divide il clock per 8 =>
clock = 4.000.000 /4 = 1.000.000 /8 = 125.000 => il timer si incrementa ogni 1/125.000 = 8 µsec.
Il contatore conta fino a 8 bit e dopo ricomincia => 8 µsec x 256 = 2048 µsec ~ 2 msec. Puo` quindi essere utilizzato per misure di tempo abbastanza accurate.
I Timer0, Timer2 sono a 8 bits mentre il Timer1 e` a 16 bits.

L’esempio che segue mostra un programma per misurare la durata di un impulso, dal fronte di
salita fino a quello di discesa, e stamparla sul terminale.
Provate ad eseguirlo inviando un segnale TTL di frequenza
1 KHz, con duty cycle di circa il 50 % e fate varie misure variando la frequenza e il duty cycle. Ricompilando il programma con prescaler diversi si puo’ variare il fondo scala.
Il programma si aspetta segnali TTL in ingresso nel PIN RB_3 (pin 36). Utilizzate il generatore e la breadboard bianca, portando il segnale sulla strippiera con uno degli aghi

ATTENZIONE !!!!!!

Inviate solo segnali TTL (0V–5V), **MAI** segnali che possano diventare negativi o superare i 5 Volts, perche’ il PIC si rompe di sicuro !!!!!

Esempio di uso di Timer0

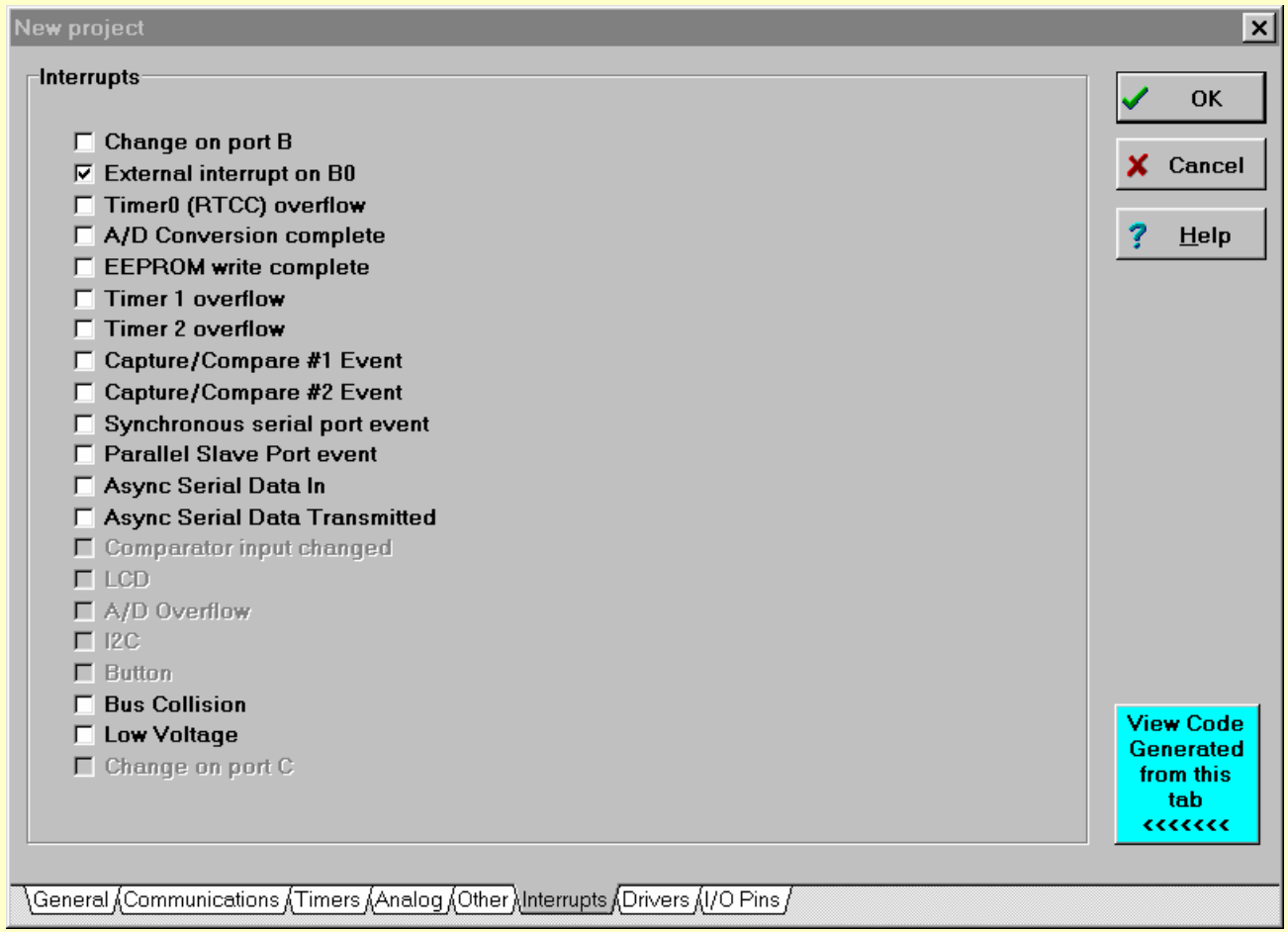
```
byte time;
void main() {



   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_8); // questo fissa il prescaler a
   setup_timer_1(T1_DISABLED);               // (fclock/4) /8
   setup_timer_2(T2_DISABLED,0,1);           // = 8 usec
   setup_ccp1(CCP_OFF);                      // quindi il  timer0 conta un tempo
   setup_ccp2(CCP_OFF);                      // pari a 8 x 256 = 2 msec

while(1==1) {
    printf("Aspetto un fronte...\n\r");
    while(input(PIN_B3)) ;        /* se il segnale e'  alto aspetta che diventi basso */
    delay_us(3);                  /* aspetta che si stabilizzi */
    while(!input(PIN_B3));        /* aspetta un fronte di salita */
    set_rtcc(0);                  /* setta il timer a 0 */
    delay_us(3);                  /* aspetta che si stabilizzi */
    while(input(PIN_B3));         /* aspetta un fronte di discesa */
    time = get_rtcc();            /* legge il valore del timer0 */
    printf("Valore del contatore: %2X   \n\n\r",time);

            }
}
```

Esercizio (da consegnare):

1) Modificare il programma precedente per misurare il periodo del segnale e non soltanto la parte di impulso positivo;
2) Fare quindi la media di 100 periodi e stamparla sul terminale.

Utilizzo degli INTERRUPTS

Gli interrupts sono dei componenti software-hardware, contenuti all'interno del PIC, in base ai quali il PIC, quando riceve un segnale di interrupt, interrompe immediatamente quello che stava facendo ed esegue una determinata funzione, detta funzione di Interrupt.
Per esempio possiamo configurare il PIC in modo che generi un segnale di interrupt non appena viene premuto uno dei nostri tasti, o quando l'ADC ha terminato la conversione analogico-digitale, o quando noi inviamo dalla tastiera del PC un carattere alla porta seriale del PIC.
Il PIC wizard del compilatore CCS e` molto comodo per configurare gli interrupts desiderati.



Richiedo un interrupt ogni qual volta si presenta un fronte di salita sul pin 0 della porta B (RB0)
=> questo wizard produce:

```
#include "C:\lab3\lab3\programmi\inter_rb0\interrb0.h"
#int_ext
ext_isr() {
  . . . .
}


void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);
   enable_interrupts(INT_EXT);
   enable_interrupts(global);


}
```

questa e' la routine di interrupt: il compilatore la prepara perche' noi la possiamo riempire. Viene eseguita ogni volta che il pin 0 della porta B rivela un fronte di salita

Abilitazione globale degli interrupts e di quello per il pin 0 della porta B

Le variabili globali, definite fuori da tutte le funzioni, possono essere utilizzate anche dentro le routines di interrupt

Esempio

```
#include "C:\elettronica\pic\lab3\programmi\int_rb0\intrb0.h"

int tmp; // variabile globale, fuori da main e dalla funzione di interrupt

#int_ext
ext_isr() {

if (tmp ==1) { // verifica se tmp vale 1, nel caso prosegue ...
// accende il led al fronte di salita di RB0
output_high(PIN_C0);
delay_ms(500);
output_low(PIN_C0);
delay_ms(500);
tmp =0;
}

else tmp = 1;
}


void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);
   enable_interrupts(INT_EXT);
   enable_interrupts(global);
   port_b_pullups(TRUE);
   tmp =1;
while(1) {}

}
```

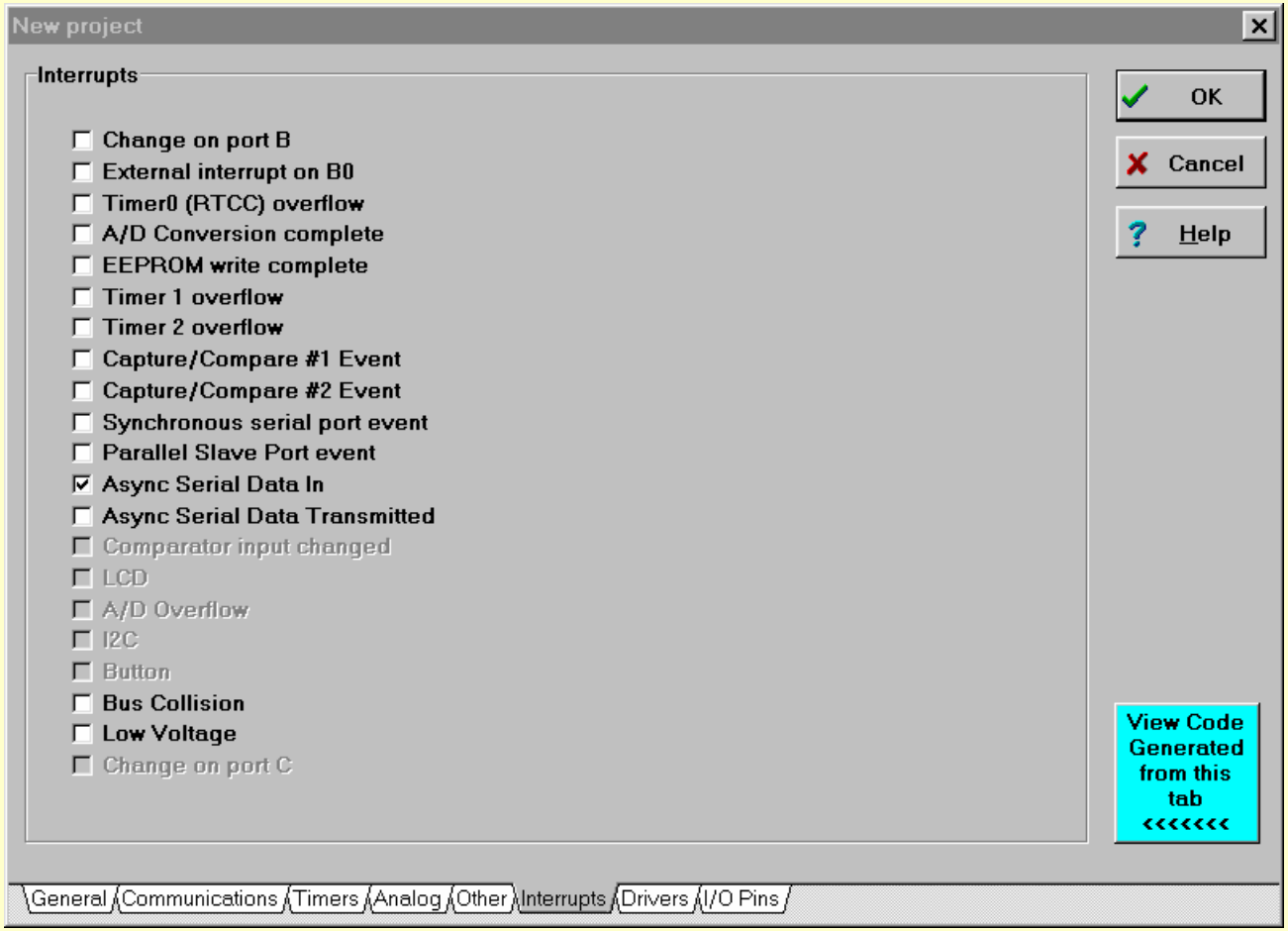
Il LED C0 viene acceso per 500 ms e quindi spento una volta ogni due pressioni del tasto B0. Si noti che nella routine main() non viene eseguito niente. Tutto viene eseguito dentrola routine di interrupt.

Esercizio (da consegnare):

1) Modificare il programma precedente in modo tale che il LED si accenda alla prima pressione del tasto e si spenga alla seconda pressione, come in un interruttore.

2) Modificare il programma in modo tale che il LED lampeggi alla prima pressione di RB0, e si spenga alla seconda pressione di RB0.

Interrupt da porta seriale (in input)

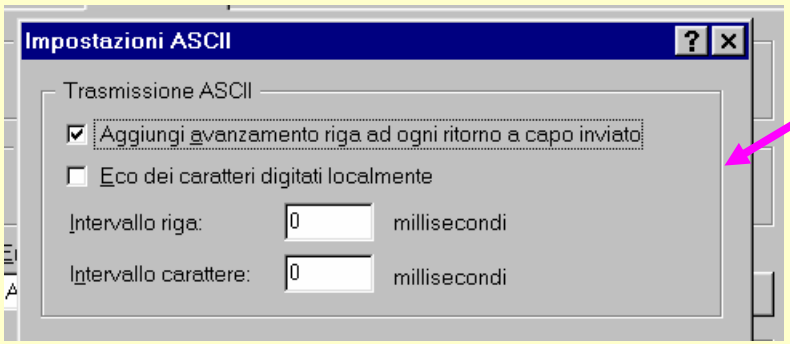


```
#include "C:\elettronica\pic\lab3\programmi\inter_rs232\inter232.h"
#int_rda
rda_isr() {
- . - .
}


void main() {

   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_CLOCK_DIV_2);
   setup_spi(FALSE);
   setup_psp(PSP_DISABLED);
   setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);
   enable_interrupts(INT_RDA);
   enable_interrupts(global);
```

routine di interrupt
generata dal wizard …

Esempio

```
#int_rda
rda_isr() {
buffer[next_in] = getc();

if (buffer[next_in] == '\n') {printf("ECCO LA STRINGA:  ",next_in);
                              i = 0;
                              while(i <= next_in) {printf("%c",buffer[i]);
                                                   i++;}
                                                   next_in =0;}
else if (next_in < (BUFFER_SIZE-1)) next_in ++;
else {printf("Comando troppo lungo ! \n\r"); next_in=0;}

}
```



Una stringa viene scritta da tastiera. Quando si riceve il carattere di invio \n , la stringa viene visualizzata. ATTENZIONE: per funzionare le proprieta' di hyperterminal (proprieta' -> impostazioni -> … devono essere come in figura

Esercizio

Generate un treno di impulsi, duty cycle 50 %, T = 1 ms ,
di numero variabile impostato da seriale
Puo` essere utile la funzione “myatoi” suggerita qui sotto per ottenere un intero da un stringa asci inserita da tastiera. A destra un esempio di come invocare “myatoi”:

```
unsigned long myatoi(char *s)
{
   unsigned long result = 0;
 int ptr;
 char c;

   ptr=0;
 result = 0;

 do
      c=s[ptr++];
   while (c<'0'||c>'9');

 while (c>='0' && c<='9') {
            result = 10*result + c - '0';
            c = s[ptr++];
                          }


   return(result);
}
```

```
….
#define MAX_BUF_SIZE 20
char buffer[MAX_BUF_SIZE];
int main() {
….
Printf(“digitare il nro di impulsi : \n”);
for (i=0;i<MAX_BUF_SIZE;i++) {
  c=getc();
  buffer[i]=c;
  if (buffer[i] == ‘\n’) i = MAX_BUF_SIZE;
}
num=myatoi(buffer);
….
}
```

Interrupt da timer0

Si puo` configurare il PIC per generare un interrupt ogni volta che il timer0 scatta da 255 a 0.
Tipicamente questa tecnica viene utilizzata per contare il tempo.
Al solito il PIC wizard puo` definire una funzione di interrupt che potete poi riempire.

Esempio: contatore di secondi

```
#include "C:\elettronica\pic\lab3\programmi\inter_tmr0\intertm0.h"
#define INTS_PER_SECOND 15     // (4000000/(4*256*256))
byte seconds;       // A running seconds counter
byte int_count;     // Number of interrupts left before a second has elapsed

#int_rtcc                      // This function is called every time
rtcc_isr() {                   // the RTCC (timer0) overflows (255->0).
                               // For this program this is apx 76 times
    if(--int_count==0) {       // per second.
      ++seconds;
      int_count=INTS_PER_SECOND;
    }
  }

void main() {

   byte start;

   setup_counters(RTCC_INTERNAL,RTCC_DIV_256);
   enable_interrupts(INT_RTCC);
   enable_interrupts(global);

   set_rtcc(0);
   int_count=INTS_PER_SECOND;

do {
      printf("Press any key to begin.\n\r");
      getc();
      start=seconds;
      printf("Press any key to stop.\n\r");
      getc();
      printf("%u seconds.\n\r",seconds-start);
   } while (TRUE);

}
```

Esercizio (da consegnare):
Scrivete un programma che conta anche
i centesimi di secondo.

Cronometro ?

Esercizio finale da consegnare separatamente:

Realizzare un capacimetro, uno strumento cioe` che misuri la capacita` di un condensatore.

Consigli iniziali: potete utilizzare ad esempio l'output digitale di una porta per caricare il condensatore attraverso una resistenza dell'ordine di 50 Ω e quindi mettere la porta in input, in modo da chiudere il pin su alta impedenza. A questo punto misurate la scarica su una resistenza nota utilizzando un canale dell'ADC. Vi consigliamo di utilizzare inizialmente un condensatore da 100 μF ed una resistenza da 1KΩ. Utilizzate la documentazione disponibile nei data-sheet di laboratorio per eventuali chiarimenti delle caratteristiche di output dei pin che utilizzate. Per utilizzare pin del PIC che sono collegati a qualcosa che non volete utilizzare togliete il ponticello.

Migliorate la realizzazione dello strumento a vostro piacimento e discutete poi: metodo, precisione, range dello strumento realizzato ed eventuali miglioramenti che si potrebbero apportare.