I microcontrollori PIC – Parte II

Prof. Angelo Monfroglio

Introduzione

Come annunciato nella parte I, approfondiamo in questa seconda parte, la struttura del PIC (memorie, pila, timer, interrupt, gestione di A/D, ecc.) e forniamo esempi di programmi in Assembler e Basic per PIC e applicazioni, fra cui la gestione di un display LCD. Consideriamo il PIC16F876A, il PIC16F877A e il PIC16F887. Nella terza parte studieremo i timer 1 e 2, i comparatori, il PWM, l'interfaccia seriale e altre applicazioni, anche per i PIC18.

Come abbiamo detto nella prima parte, i PIC adottano l'architettura Harvard, cioè i bus di indirizzamento per la memoria dati (RAM) e per la memoria programmi (ad esempio Flash) sono distinti.

Una nota fondamentale

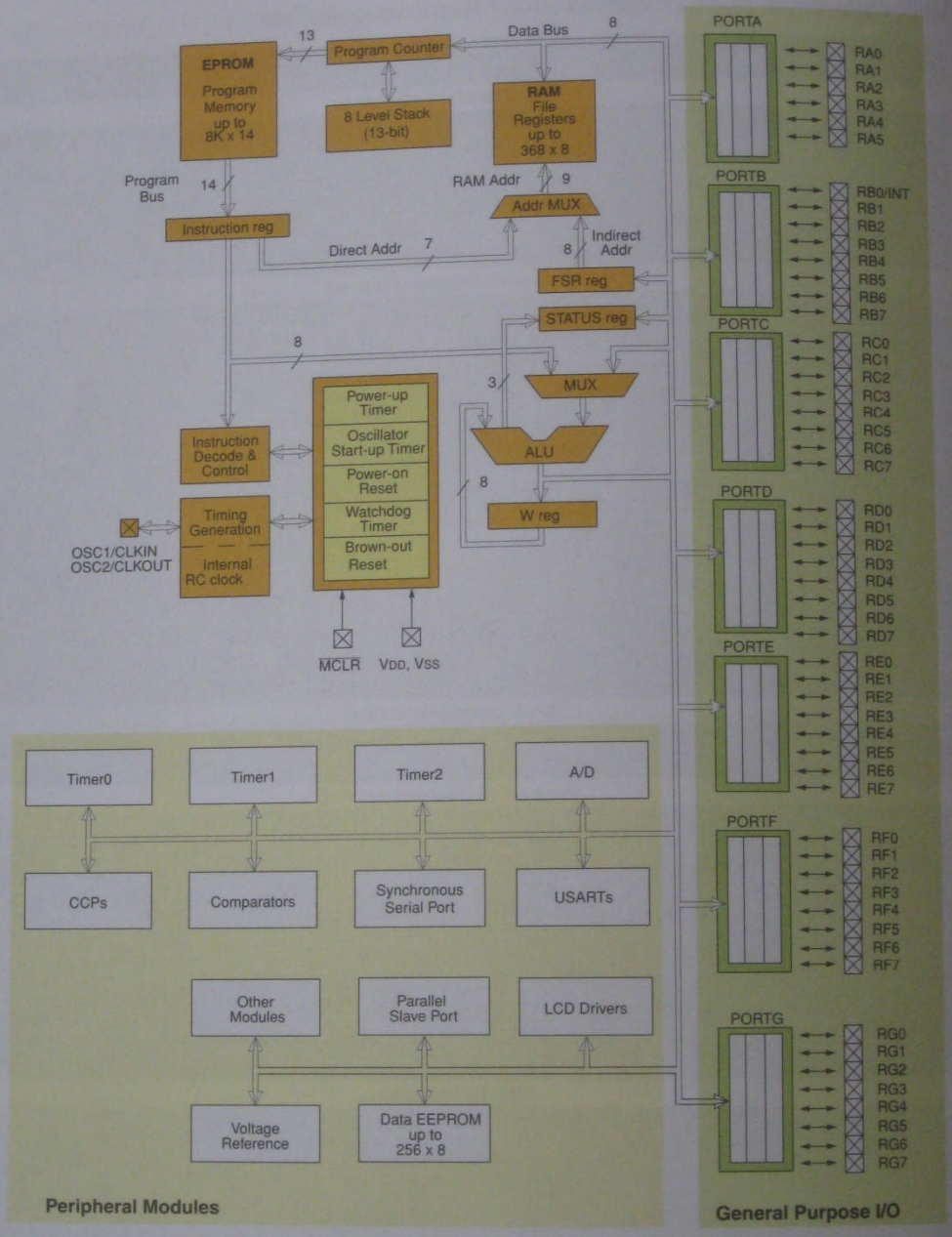
Anche se ci limitiamo a considerare PIC16F… (F sta per memoria Flash di programmi), ogni PIC è diverso dall'altro e richiede un file di configurazione appropriato. Questo viene gestito automaticamente con i sistemi di sviluppo Hardware e Software che abbiamo citato in questa dispensa. In caso contrario, occorre premettere la parola di configurazione e consultare attentamente i datasheet dei PIC adottati. Inoltre, l'assembler che usiamo è sempre lo stesso (quello dello Microchip), ma i Basic (e in parte anche i compilatori C) per PIC in circolazione sono tutti diversi. Pertanto, senza le opportune modifiche, i programmi non gireranno mai e saranno solo fonte continua di frustrazione!

In ogni caso occorre procedere in questo modo:

1. Scegliere se scrivere il programma in Assembler, o in Basic o in C; creare un progetto
2. Se si è scelto il Basic o il C, compilare generando anche l'assembler e il file HEX (eseguibile)
3. Se non ci sono errori di sintassi, simulare l'esecuzione per assicurarsi che il programma faccia davvero quello che si voleva
4. Provare il microcontrollore sulla piastra di sviluppo, eventualmente con il debugger hardware: non è detto che il programma simulato funzioni davvero.

Buona fortuna!

Qui sotto è riportata la struttura generale dei PIC a bus dati di 8 bit (PIC16xx, PIC18xx, ecc.), tratta dal libro edito da MikroElektronica citato in bibliografia.



The architecture of 8-bit PIC microcontrollers. Which of these modules are to be built into the microco
depends on the type thereof.

Interrupt

Tutti i microcontrollori (così come i microprocessori in un personal computer) permettono e gestiscono gli interrupt, cioè le interruzioni. Interrupt significa che il programma (principale) viene interrotto, a causa di un evento esterno o interno al microcontrollore, per eseguire una subroutine (di gestione dell'interrupt), dopo di che viene ripreso dal punto in cui è stato interrotto. E' un po' come un turista che segue in auto un percorso principale e, arrivato ad un certo punto, è attratto da un cartello e fa una deviazione, posteggiando l'automobile e badando bene a memorizzare il luogo dove ha posteggiato, per farvi ritorno. Ma perché gli interrupt? Il microcontrollore deve gestire molti eventi e parecchie periferiche: fra gli eventi interni un timeout di un timer, un segnale dalla porta seriale, ecc.; fra quelli esterni la conversione avvenuta di A/D, un impulso da un sensore, un tasto premuto su un tastierino, ecc. Per servire questi eventi ci sono due possibilità: il polling (appello) e, appunto, l'interrupt. L'esempio classico che viene bene in un'aula scolastica è l'interazione docente – studenti. Il docente periodicamente può rivolgersi ad ogni studente chiedendo se ha qualche domanda da fare(polling). E' evidente che con 30 allievi il tempo perso è tanto, visto che in molti casi lo studente non vuole intervenire. E' ovvio che la soluzione è chiedere agli studenti di interrompere, ad esempio alzando la mano, quanto vogliono interloquire. Il docente può accettare l'interruzione oppure no, badando a riprendere il discorso (senza perdere il filo) dopo la gestione dell'interruzione.

Nel caso dei nostri microcontrollori abbiamo il Program Counter (contatore di programma) che è appunto un contatore che contiene l'indirizzo della memoria di programma dove c'è la prossima istruzione da eseguire. Il Program Counter (PC) ha una larghezza di bit tale da poter indirizzare tutta la memoria di programma: ad esempio 13 bit per avere 2 alla 13, cioè 8k parole. Il PC continua ad incrementarsi (arrivato in fondo ovviamente torna a zero) a meno che si verifichino 4 fatti:

-c'è un'istruzione di Stop (nel caso dei PIC di Sleep)

-c'è un'istruzione di salto (GOTO, Skip, Branch) o salto a subroutine (CALL). Se c'è salto alla posizione n, n è caricato nel PC e non si torna più indietro. Se c'è chiamata a subroutine, alla fine, con l'istruzione Return si torna al punto del PC di chiamata

-c'è un interrupt: avviene una chiamata alla subroutine che gestisce l'interrupt

-il programma va in stallo (il PC è bloccato o continua a ruotare in un ciclo senza fine): se è abilitato, interviene il watchdog e provvede ad un reset (il PC è rimandato a zero).

Ogni volta che viene lanciata una subroutine (semplice o di gestione di interrupt) l'indirizzo presente nel PC è salvato in un'apposita memoria chiamata Stack (pila) per poter essere recuperato. Sono possibili anche più subroutine annidate. In questo caso lo Stack è a più livelli e l'organizzazione si dice LIFO (Last In First Out), cioè vengono recuperati gli indirizzi prima della chiamata a partire dall'ultimo, proprio come in una pila di libri sovrapposti. Quando viene accettato un interrupt avviene un salto ad una predeterminata locazione di memoria. In alcuni microcontrollori, a seconda del dispositivo che ha richiesto l'interruzione, avviene un salto ad una differente posizione. Nel caso dei nostri PIC16 il salto è all'indirizzo fisso 0004 (esadecimale). Qui deve esserci una chiamata alla routine che gestisce l'interrupt.

Ci sono varie fonti di interrupt, di cui una esterna attraverso un apposito piedino. Alcuni registri governano gli interrupt e, prima di tutto devono essere abilitati, altrimenti sono ignorati (si dice anche mascherati). Vediamo in dettaglio.

Registro INTCON

Bit 7	6	5	4	3	2	1	0
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
R/W, 0	R/W, 0	R/W, 0	R/W, 0	R/W, 0	R/W, 0	R/W, 0	R/W, X

Dove R/W significa che il bit è leggibile e scrivibile; 0 significa che è azzerato al Reset, X che è indeterminato dopo il Reset.

GIE Global Interrupt Enable: 1 = tutti gli interrupt abilitati

PEIE Peripheral Interrupt Enable: tutti gli interrupt tranne che per il Timer0 e RB0/INT

T0IE TMR0 Overflow Interrupt Enable

INTE RB0/INT External Interrupt Enable

RBIE RB Change Interrupt Enable: quando la porta B è configurata come ingress, ogni cambiamento di livello logico provoca interrupt

T0IF TMR0 Overflow Interrupt Falg: segnala l'interrupt generato da overflow del Timer0

INTF RB0/INT External Interrupt Flag: segnala che c'è stato interrupt dal piedino INT

RBIF RB Port Change Interrupt Flag: segnala interrupt da cambio di livello dei pin della porta B

Registro PIE1

Bit 7	6	5	4	3	2	1	0
-	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

ADIE A/D Converter Interrupt Enable: 1 abilitato

RCIE EUSART Receive Interrupt Enable: 1 abilita interrupt ricezione seriale asincrona

TXIE Transmit Interrupt Enable: idem alla trasmissione

SSPIE Master Synchronous Serial Port (MSSP) Interrupt Enable: abilita interrupt ogni volt ache un dato è stato trasferito in modo sincrono sulla seriale (modo SPI o I^2C)

CCP1IE CCP1 Interrupt Enable: abilita la richiesta di interrupt dal modulo CCP1 (Compare/Capture)-PWM

TMR2IE abilita interrupt confronto Timer2 e PR2

TMR1IE abilita interrupt per overflow Timer1

Esempio di Interrupt scritto in mikroBasic (ditta MikroElektronika)

‘Ogni overflow del Timer1 (registri TMR1H e TMR1L) causa un interrupt, incrementando la variabile ‘contatore cnt

```
dim unsigned  short cnt

sub procedure interrupt

        cnt = cnt + 1

        PIR1.TMR1IF = 0         ‘Reset bit TMR1IF

        TMR1H = 0x80            ‘return TMR1 registers

        TMR1L = 0x80

end sub


main:

        ANSEL, ANSELH = 0       ‘tutti I pedini sono configurati come digitali

        T1CON = 1               ‘Fa partire il Timer1

        PIR1.TMR1IF = 0         ‘Reset TMR1IF

        TMR1H  =  0x80          ‘Imposta il valore iniziale del Timer1

        TMR1L  = 0x00

        PIE1.TMR1IE 0 1         ‘Abilita interrupt per overflow


        Cnt  = 0                ‘reset contatore

        INTCON  =  0xC0         ‘abilita interrupt (GIE e PEIE)

…
```

Si noti la parola chiave interrupt per qualificare la subroutine.

Registro PIE2

Il registro PIE2 contiene altri bit di abilitazione per gli interrupt.

Bit 7	6	5	4	3	2	1	0
OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	-	CCP2IE

OSFIE Oscillator Fail Interrupt Enable: abilita interruzione per caduta oscillatore

C2IE Comparator C2 Interrupt Enable: abilita interrupt comparatore numero 2

C1IE Idem per il numero 1

EEIE EEPROM Write Operation Interrupt Enable: abilita interrupt per operazione di scrittura su EEPROM

BCLIE Bus Collision Interrupt Enable: abilita interruzione per collisione sul Bus

ULPWUIE Ultra Low Power Wake-up Interrupt Enable: abilita interruzione risveglio da basso consumo

CCP2IE CCP2 Interrupt Enable: abilita interrupt da modulo CCP2

Esempio di programma in mikroBasic (dal libro citato in bibliografia)

```
'Il comparatore C2 è configurato per usare i piedini RA0 e RA2 come ingresso
'Ogni cambiamento dell'uscita del C2 causa il cambiamento sul pin PORTB.1 nella routine di interrupt
Sub procedure interrupt
        PORTB.F1 = not PORTB.1          'complementa il pin
        PIR2.C2IF = 0                   'azzera il flag di C2IF
end sub
main:
        TRISB = 0                       'tutta la porta B come uscita
        PORTB.1 = 1
        ANSEL = %00000101               'fissa RA0/C12IN0 e RA2/C2IN come ingressi analogici
        ANSELH = 0                      'fissa come digitali
        CM2CON0.C2CH0 = 0               'piedino Ra0 diventa il complemento di C2
        CM2CON0.C2CH1 = 0
        PIE2.C2IE = 1                   'abilita l'interrupt per il comparatore C2
        INTCON.GIE = 1                  'abilitazione globale interrupt
        CM2CON0.C2ON = 1                'il comparatore C2 è abilitato
```

Occore notare che ANSEL e ANSELH sono SFR (Special Function REgister) collocati nel banco 3 agli indirizzi 188h e 189h che non avevamo riportato nella parte I.

Il registro PIR1

Contiene flag di interrupt. Non lo si confonda con il simile PIE1 che abilita i corrispondenti interrupt.

Bit 7	6	5	4	3	2	1	0
-	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

ADIF A/D Converter Interrupt Flag: 1 significa che la conversione è stata ultimata

RCIF Receive Interrupt Flag: 1 significa che il buffer di ricezione della seriale (EUSART) è pieno

TXIF Transmit Interrupt Flag: 1 significa che il buffer di trasmissione è vuoto

SSPIF Master Synchronous Serial Port Interrupt Flag: 1 significa che le condizioni di ricezione e trasmissione seriale sincrona sono verificate

CCP1If CCP1 interrupt: 1 significa che il confronto del modulo Compare-Capture ha dato esito positivo

TMR2IF Timer2 Interrupt Flag: 1 significa che il registro di 8 bit è stato confrontato con PR2; questo Flag deve essere azzerato dalla routine di interrupt prima del ritorno

TMR1IF Timer1 Overflow Interrupt: 1 significa che il registro del Timer1 ha raggiunto il fondo

Il registro PIR2

Altri Flag.

Bit 7	6	5	4	3	2	1	0
OSFIF	C2If	C1If	EEIF	BCLIF	ULPWUIF	-	CCP2If

OSFIF Oscillator Fail Interrupt Flag: 1 significa che l'oscillatore al quarzo non funziona e si commuta all'oscillatore di riserva. Deve essere poi resettato

C2IF Comparator C2 Interrupt Flag: 1 significa che C2OUT è cambiato; va resettato

C1IF idem per C1

EEIF EEPROM Write Operation Interrupt Flag: 1 significa che la scrittura su EEPROM è stata completata

BCLIF Bus Collision Interrupt Flag: 1 significa che c'è stata collision di bus su MSSP

ULPWUIF Ultra Low Power Wake-up Interrupt Flag: 1 significa che c'è stato risveglio da condizione di basso consumo

CCP2IF CCP2 interrupt condition met: unità Capture-Comparing-PWM

Registro PCON

Varie condizioni di funzionamento: abilitazioni e stato

Bit 7	6	5	4	3	2	1	0
--	-	ULPWUE	SBOREN	-	-	POR	BOR

ULPWUE Ultra Low Power Wake-up Enable: 1 significa abilitato

SBOREN Software BOR Enable: 1 significa abilitazione Brown-out reset (reset per caduta di tensione)

POR Power-on Reset Status: 1 significa che non c'è stato reset per accensione

BOR Brown-out Reset Status: 1 se non c'è stato reset per caduta di tensione

Esempio in mikroBasic di Wake Up

```
'Sequenza attivazione ULPWU (wake-up)
main:
    PORTA.0 = 1
    ANSEL,ANSELH = 0      'tutti i pin di Porta A digitali
    TRISA =  0            'Porta A come uscita
    Delay_ms(1)           'ritardo di 1 millisecondo
    PIR2.ULPWU = 1        'azzera
    PCON.ULPWUE = 1       'abilita
    TRISA.0 = 1           'piedino 0 di Porta A come ingresso
    PIE2.ULPWUIE = 1      'abilita interrupt
    INTCON.GIE = 1        'abilita tutti gli interrupt (non mascherati)
    SLEEP                 'Stop (sleep mode)
    …
```

Riassunto delle fonti di interrupt

EEPROM, Oscillatore, A/D Converter, USART in ricezione, USART in trasmissione, MSSP (errore di trasmissione) CCP1, CCP2, Timer1, Timer2, Comparatore analogico 1, Comparatore analogico2, RA0; Timer0, RB0/INT (interrupt esterno).

I Timer

Il timer 0

Il PIC16F887 ha 3 timer con caratteristiche e uso diversi. Il Timer0 è un timer/counter a 8 bit che può essere usato come:

-timer o contatore a 8 bit

-prescaler (divisore di frequenza) a 8 bit (usato anche per il Watchdog)

-sorgente di clock

-interrupt

-selettore per il fronte di salita o discesa degli impulsi esterni.

Il registro OPTION

Bit 7	6	5	4	3	2	1	0
RBPU	INTEDG	T0Cs	T0SE	PSA	PS2	PS1	PS0

RBPU PORTB Pull-up enable: 1 significa che i resistori di pull-up sono disabilitati

INTEDG Interrupt Edge Select: 1 significa che l'interrupt sul pin INT è attivato dal fronte di salita

T0CS TMR0 Clock Select: 1 significa che gli impulsi al Timer0/Counter sono presi dal pin RA4; 0 che il Timer0 usa il clock interno (Frequenza dell'oscillatore / 4)

T0SE TMR0 Source Edge Select: 1 significa che l'incremento avviene per un passaggio da alto a basso

PSA Prescaler Assignment: 1 significa che il prescaler è assegnato al Watchdog,0 al Timer0

PS2, PS1, PS0 sono bit di prescaler (divisore) secondo la seguente tabella che indica di quanto viene divisa la frequenza

PS2	PS1	PS0	TMR0	Watchdog
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32

1	1	0	1:128	1:64
1	1	1	1:256	1:128

Esempio di uso del Timer 0 come timer e come contatore, e del Prescaler assegnat al timer, in mikroBasic attraverso il registro Option

```
'Timer0 come timer e prescaler assegnato al timer
Insigne cnt
Sub procedure interrupt
        cnt = cnt + 1
        TMR0 = 155
        INTCON = 0x20          'T0IE settato, T0IF zero
end sub
main:
        OPTION_REG = 0x04      'Prescaler (1:32) al timer
        TMR0 = 155             'Timer0 da 155 a 255
        INTCON = 0xA0          'abilitazione interrupt Timer0/Counter0
…
'Timer0 come contatore con prescaler
        OPTION_REG = 0x20      'Prescaler 1:2
        TMR0 = 155             'Contatore da 155 a 255
        INTCON = 0xA0          'abilitazione interrupt
…
'Esempio di assegnazione del prescaler (1:64) al timer del Watchdog
main:
        OPTION_REG = 0x0E
        asm                    'inserisco istruzioni in assembler per agire sul Watchdog
        CLRWDT                 'resetto il Watchdog timer
        end asm                'ritorno in ambiente Basic
```

….

Riassunto delle operazioni per gestire il Timer0

-Selezione del modo timer o counter con T0CS nel registro OPTION

-Se si usa un prescaler, va assegnato ponendo a zero PSA e scegliendo il valore di divisione con i bit PS2,PS1,PS0

-Se si vuole attivare l'interrupt, occorre mettere a 1 GIE e TMR0IE nel registro INTCON

Nel modo timer (conta tempi): azzerare TMR0 oppure caricare il valore desiderato di partenza. Ogni volta che si raggiunge l'overflow (fine corsa) il Flag TMR0IF è posto a 1 automaticamente e, se abilitato, interviene l'interrupt

Nel modo contatore (conta eventi): vengono contati gli impulsi sul piedino RA4. T0SE seleziona il fronte di salita o discesa che attiva il conteggio. Il registro TMR0 contiene il conteggio. Prescaler e interrupt sono gestiti come nel modo temporizzatore.

Occorre anche ricordare che:

-ogni scrittura su TMR0 azzera il prescaler se è assegnato al timer

-se il prescaler è assegnato al watch-dog, l'istruzione CLRWDT azzera entrambi

-la scrittura su TMR0 non fa partire subito il timer, ma dopo 2 cicli

-se il microcontrollore è posto in modo SLEEP, il clock si ferma e il TMR0 non può risvegliarlo

-se usato come contatore senza prescaler, la durata minima dell'impulso è 2 Tosc + 20 nano secondi

-se con prescaler vale 10 nano secondi

-il prescaler non può essere usato dall'utente in scrittura o lettura diretta.

Il convertitore analogico – digitale

Il PIC16F887 dispone di 14 canali di A/D a 10 bit. Per gestire gli A/D si usano 4 registri:

ADRESH indirizzo 1Eh banco 0: contiene la parte alta del risultato della conversione

ADRESL indirizzo 9Eh banco 1: contiene la parte bassa

ADCON0 indirizzo 1Fh banco 0: registro di controllo 0

Bit 7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

ADCS1	ADCS0	Clock
0	0	Freq. Oscillatore / 2
0	1	/8
1	0	/32
1	1	Oscillatore interno RC

Selezione del canale

CHS3	CHS2	CHS1	CHS0	Canale	Pin
0	0	0	0	0	RA0/AN0
0	0	0	1	1	RA1/AN1
0	0	1	0	2	RA2/AN2
0	0	1	1	3	RA3/AN3
0	1	0	0	4	RA5/AN4
0	1	0	1	5	RE0/AN5
0	1	1	0	6	RE1/AN6
0	1	1	1	7	RE2/AN7
1	0	0	0	8	RB2/AN8
1	0	0	1	9	RB3/AN9
1	0	1	0	10	RB1/AN10
1	0	1	1	11	RB4/AN11
1	1	0	0	12	RB0/AN12
1	1	0	1	13	RB5/AN13

1	1	1	0	CVref	
1	1	1	1	Vref = 0.6 V	

ADCON1 indirizzo 9Fh banco 1: registro di controllo 1

ADFM	-	VCFG1	VCFG0	-	-	-	-

ADFM: 1 = il risultato è allineato a destra, i 6 bit più a sinistra di ADRESH non sono usati;

ADRESH ADRESL

						Ris9	Ris8	7	6	5	4	3	2	1	0
Bit 7	6	5	4	3	2	1	0	Bit 7	6	5	4	3	2	1	0

0 = il risultato è allineato a sinistra, i 6 bit meno significativi di ADRESL non sono usati:

ADRESH ADRESL

Ris9	Ris8	7	6	5	4	3	2	1	0						

VCFG1: 1 = il piedino Vref- alimentato con tensione negativa; 0 significa che Vss è il valore inferiore per il convertitore A/D.

VCFG0: 1 = Vref+ è il riferimento superiore; 0 = Vdd è il riferimento superiore di tensione.

ANSEL, ANSELH devono essere configurati per selezionale i piedini come analogici, ai rispettivi indirizzi 188h e 189h del banco 3 degli SFR. Un 1 significa che l'ingresso è analogico. Vanno anche selezionati gli ingressi (valore 1) oppure uscite (0) con i TRISA, TRISB, ecc.

I convertitori sono ad approssimazione successiva, come più della metà di quelli oggi usati. Ogni A/D richiede di fatto 3 segnali: partenza della conversione (genericamente SOC = start of conversion), segnale di fine (EOC = end of conversion) e lettura dell'uscita. Inoltre, deve essere impostata la tensione di riferimento che determina il fondo scala e il clock che regola il tempo di conversione.

Qui GO/DONE = 1 significa che la conversione è in corso 0 significa EOC. ADON = 1 significa convertitore abilitato: parte la conversione.

Nel caso peggiore, in condizioni normali, il tempo totale di conversione per i 10 bit non supera 20 micro secondi, un tempo usuale per la maggior parte dei convertitori A/D ad approssimazioni successive.

Quando la conversione è finita ADRESH e ADRESL vengono caricati col risultato e ADIF (flag di interrupt dell'A/D) è posto a 1.

Riassunto delle operazioni per il convertitore A/D

1. Porre un 1 sul bit corrispondente del registro TRIS (TRIA, ecc.) per configurarlo come ingresso
2. Porre un 1 nel registro ANSEL per configurarlo come analogico
3. Configurare il riferimento di tensione in ADCON1
4. Scegliere un clock in ADCON0
5. Selezionare il canale desiderato in ADCON0
6. Selezionare il formato in uscita in ADFM del registro ADCON0
7. Se si usa l'A/D in modalità interrupt azzerare ADIF
8. Mettere a 1 ADIE, PEIE e GIE (se si usa l'interrupt)
9. Abilitare l'A/D con ADON in ADCON0
10. Dopo 20 micro secondi la conversione è pronta
11. Se si sceglie il polling si può testare GO/DONE
12. Se si usa l'interrupt attendere l'interruzione
13. Il risultato si trova nei registri ADRESH e ADRESL.

Un semplice esempio in MikroBasic

'Lettura continua di un segnale analogico sul canale A/D 2 e scrittura del risultato sulle porte B e C

```
dim adc_rd as word

main:

    ANSEL = 0x04    'configura il piedino come analogico canale 2

    TRISA = 0xFF    'PORTA come ingresso

    ANSELH = 0      'Configura gli altri ingressi come digitali

    TRISC = 0x3F    'i pin RC6 e RC7 sono configurati come uscite

    TRISB = 0       'PORTB come uscita

    while 1         'Ciclo perpetuo

        temp_res = ADC_Read(2)      'subroutine che legge dall'A/D

        PORTB = tem_res             'parte meno significativa in PORTB

        PORTC = temp_res >> 2       'parte più significativa (2 bit) in PORTC

    wend

end.
```

Esempio in assembler (commenti in C)

```
_main:
;example7.c,21 ::           void main() {
;example7.c,22 ::           ANSEL  = 0x0C;                // Pins AN2 and
AN3 are configured as analog
      MOVLW      12
      MOVWF      ANSEL+0
;example7.c,23 ::           TRISA  = 0xFF;                // All port A pins
are configured as inputs
      MOVLW      255
      MOVWF      TRISA+0
;example7.c,24 ::           ANSELH = 0;                 // Rest of pins is
configured as digital
      CLRF       ANSELH+0
;example7.c,25 ::           TRISB  = 0x3F;                  // Port B pins
RB7 and RB6 are configured as
      MOVLW      63
      MOVWF      TRISB+0
;example7.c,27 ::           TRISD  = 0;                   // All port D
pins are configured as outputs
      CLRF       TRISD+0
;example7.c,28 ::           ADCON1.F4 = 1 ;                 // Voltage
reference is brought to the RA3 pin.
      BSF        ADCON1+0, 4
;example7.c,30 ::           do {
```

```
L_main0:
;example7.c,31 ::           temp_res = ADC_Read(2);          // Result of
A/D conversion is copied to temp_res
      MOVLW      2
      MOVWF      FARG_ADC_Read_channel+0
      CALL       _ADC_Read+0
      MOVF       R0+0, 0
      MOVWF      _temp_res+0
      MOVF       R0+1, 0
      MOVWF      _temp_res+1
;example7.c,32 ::           PORTD = temp_res;               // 8 LSBs are
moved to port D
      MOVF       R0+0, 0
      MOVWF      PORTD+0
;example7.c,33 ::           PORTB = temp_res >> 2;         // 2 MSBs are
moved to bits RB6 and RB7
      MOVF       R0+0, 0
      MOVWF      R2+0
      MOVF       R0+1, 0
      MOVWF      R2+1
      RRF        R2+1, 1
      RRF        R2+0, 1
      BCF        R2+1, 7
      RRF        R2+1, 1
      RRF        R2+0, 1
      BCF        R2+1, 7
      MOVF       R2+0, 0
      MOVWF      PORTB+0
;example7.c,34 ::           }  while(1);                     // Endless
loop
      GOTO       L_main0
;example7.c,35 ::           }
      GOTO       $+0
; end of _main
```

Esempi applicativi in mikroBasic, mikroC e Assembler

Un semplice esempio

```
' Program name:
'    Example_1
' Copyright:
'   (c) MikroElektronika, 2005-2010
' Description:
'            This is a simple program used to demonstrate the operation of
the microcontroller.
'            Every second LED on PORTB is turned on.
' Configuration:
'    MCU:          PIC16F887
'
http://ww1.microchip.com/downloads/en/DeviceDoc/41291F.pdf
'    Device:       EasyPIC6
'                  http://www.mikroe.com/eng/products/view/297/easypic6-
development-system/
'    Oscillator:   HS, 08.0000 MHz
'    SW:           mikroBasic PRO v3.8
```

```
'                  http://www.mikroe.com/en/compilers/mikrobasic/pro/pic/
'
' * NOTES:
'     - Make sure you turn ON the PORTB LEDs at SW9.
'                           ac:SCHEMATIC
'*************************************************************************
**

program example_1

main:
  ANSEL  = 0                  ' All I/O are configured as digital
  ANSELH = 0
  PORTB = %01010101           ' Binary combination on PORTB 'ac:LED
  TRISB = 0                   ' PORTB pins are configured as outputs

end.
```

Un esempio di accensione di LED sulle porte A, B, C e D, in linguaggio C

```
/*
 * Project name:
     LED_Blinking (Simple 'Hello World' project)
 * Copyright:
     (c) Mikroelektronika, 2008.
 * Revision History:
     20080930:
       - initial release;
 * Description:
     This is a simple 'Hello World' project. It turns on/off LEDs connected
to
     PORTA, PORTB, PORTC and PORTD.
 * Test configuration:
     MCU:             PIC16F887

http://ww1.microchip.com/downloads/en/DeviceDoc/41291F.pdf
     Dev.Board:       EasyPIC5
                      http://www.mikroe.com/en/tools/easypic5/
     Oscillator:      HS, 08.0000 MHz
     Ext. Modules:    -
     SW:              mikroC PRO for PIC
                      http://www.mikroe.com/en/compilers/mikroc/pro/pic/
 * NOTES:
     - Turn ON the PORT LEDs at SW6. (board specific)
*/

void main() {

  ANSEL  = 0;            // Configure AN pins as digital
  ANSELH = 0;
  C1ON_bit = 0;          // Disable comparators
  C2ON_bit = 0;

  TRISA = 0x00;          // set direction to be output
  TRISB = 0x00;          // set direction to be output
  TRISC = 0x00;          // set direction to be output
  TRISD = 0x00;          // set direction to be output

  do {
```

```
    PORTA = 0x00;        // Turn OFF LEDs on PORTA
    PORTB = 0x00;        // Turn OFF LEDs on PORTB
    PORTC = 0x00;        // Turn OFF LEDs on PORTC
    PORTD = 0x00;        // Turn OFF LEDs on PORTD
    Delay_ms(1000);      // 1 second delay

    PORTA = 0xFF;        // Turn ON LEDs on PORTA
    PORTB = 0xFF;        // Turn ON LEDs on PORTB
    PORTC = 0xFF;        // Turn ON LEDs on PORTC
    PORTD = 0xFF;        // Turn ON LEDs on PORTD
    Delay_ms(1000);      // 1 second delay
  } while(1);            // Endless loop
}
```

Assembler

```
;Example1.mbas,24 ::        main:
;Example1.mbas,25 ::        ANSEL  = 0                  ' All I/O are
configured as digital
      CLRF       ANSEL+0
;Example1.mbas,26 ::        ANSELH = 0
      CLRF       ANSELH+0
;Example1.mbas,27 ::
      PORTB = %01010101           ' Binary combination on PORTB 'ac:LED
      MOVLW      85

      MOVWF      PORTB+0
;Example1.mbas,28 ::        TRISB = 0                   ' PORTB pins are
configured as outputs
      CLRF       TRISB+0
      GOTO       $+0
; end of _main
```

Esempio di uso di un display LCD (2 righe di 16 caratteri)

Generalità sul display LCD (Hitachi con controller HD44780)

E' fatto apposta per essere usato con un micro controller e costa circa 10 euro. Può visualizzare i normali caratteri alfanumerici, lettere greche, segni di interpunzione, simboli matematici e caratteri creati dall'utente, attraverso 3 memorie di cui dispone:

-DDRAM (RAM per i dati)

-CGRAM (RAM per generare caratteri personalizzati)

-CGROM (ROM che contiene il generatore di caratteri interno).

Il messaggio può essere spostato automaticamente a destra e sinistra, si può controllare la forma del cursore, e, come opzione, è disponibile la retroilluminazione. I piedini sono14, più 2 per la retroilluminazione a LED.

Funzione	Pin	Nome	Valore	Descrizione
Massa	1	VSS	-	0V

Alimentazione	2	VDD	-	+5V
Contrasto regolabile	3	VEE	-	Da 0 a VDD
Controllo	4	RS	0 o 1	0 = comando, 1 dato (pin D0 – D7)
Controllo	5	R/W	0 o 1	0 scrive, 1 legge
Controllo	6	E	0 o 1 o fronte di discesa	0 accesso disabilitato, 1 operativo, fronte = trasferimento a LCD
Dato o comando	Da 7 a 14	D0..D7	0 o 1	Dato o comando

Lo schermo visualizza 2 righe di 16 caratteri con matrice di 5x8 pixel (oppure 5x11). E' anche possibile regolare il contrasto: solitamente va regolato al massimo.

La memoria DDRAM è usata per memorizzare i caratteri da visualizzare: 80. Di questi 32 sono visibili direttamente sul display. Si può configurare la DDRAM in modo che avvenga uno scorrimento automatico a destra e deve essere fissato il primo indirizzo, ad esempio 00h. Ecco la mappa.

00h																																			27h
40h							4F																												67h
Visibile								Non vis.																											

La memoria CGROM, non volatile, contiene il set di base dei caratteri visualizzabili, sotto forma di matrici di puntini (nel nostro caso 5 x 8). Gli indirizzi corrispondono in pratica ai caratteri ASCII. Occorre controllare l'indirizzo effettivo di ogni simbolo sulla mappa.

Oltre ai simboli standard, l'utente può crearne di propri con la memoria CGRAM di 64 bytes. La matrice di punti è 5 x 8 che significa 8 righe di 5 colonne:

*				
		*		
	*			

* significa acceso

Gli indirizzi vanno da 00 a 3F e ovviamente ciascuna locazione usa solo 5 bit degli 8 perché ogni riga ha 5 colonne.

00

…

3K.

Il modulo LCD può essere interfacciato con il microcontrollore usando 8 bit di dati o 4 per volta. Il vantaggio del primo modo è la facilità di gestione e la velocità, del secondo modo il risparmio di piedini usati per le porte del microcontrollore. Se si usa un PIC da 28 piedini quest'ultima soluzione può essere preferibile. Si può anche risparmiare un piedino mettendo a massa R/W e rinunciando a leggere da LCD, ma è sconsigliabile. Ogni comando dura circa 2 milli secondi per essere eseguito. Quando il circuito è alimentato LCD è azzerato., dopo 15 milli secondi, e posto nella condizione di default:

DL = 1 comunicazione a 8 bit

N = 0 il dato su una linea

F = 0 il formato della griglia è 5 x 8

D = 0 display off

U = 0 cursor off

B = 0 lampeggio off

I/D = 0 gli indirizzi sono incrementati ogni volta automaticamente di 1

S = 0 display shift off.

I comandi

Il valore sui pin D0-D7 può essere un comando o un dato. Se RS = 0 viene considerato un comando.

Comando	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Tempo
Clear	0	0	0	0	0	0	0	0	0	1	1.64 mS
Home	0	0	0	0	0	0	0	0	1	X	1.64
Mode	0	0	0	0	0	0	0	1	I/D	S	40
Display ON/OFF	0	0	0	0	0	0	1	D	U	B	40
Shift	0	0	0	0	0	1	D/C	R/L	X	X	40
Function	0	0	0	0	1	DL	N	F	X	X	40
Indirizzo CGRAM	0	0	0	1	Addr	Addr	Addr	Addr	Addr	Addr	40

Indirizzo DDRAM	0	0	1	ADDr	Addr	Addr	Addr	Addr	Addr	Addr	40
Legge flag busy	0	1	BF	Addr	Addr	Addr	Addr	Addr	Addr	Addr	40
Scrive in CGRAM o in DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	40
Legge	1	1	D7	D6	D5	D4	D3	D2	D1	D0	40

Con i significati:

I/D = 1 incremento di 1, = 0 decremento

S = 1 shift del display on, 0 off

D = 1 display on, 0 off

U = 1 cursor on, 0 cursor off

B = 1 lampeggio, 0 non lampeggio

R/L = 1 sposta a destra, 0 a sinistra

DL = 1 interfaccia a 8 bit, 0 a 4 bit

N = 1 display su 2 linee, 0 su una linea

F = 1 matrice dei caratteri 5 x 10, = 0 5 x 7

D/C = 1 spostamento sul display, = 0 spostamento del cursore.

A questo punto occorre inizializzare LCD. Cominciamo con il modo a 8 bit alla volta.

Power ON

Aspetta 15 mS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	X	X	X	X

Display in modo 8 bit

Aspetta 4.1 mS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	X	X	X	X

Aspetta 100 mS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	1	X	X	X	X	
0	0	0	0	1	1	N	F	X	X	
0	0	0	0	0	0	1	0	0	0	Off
0	0	0	0	0	0	0	0	0	1	Clear
0	0	0	0	0	0	0	1	I/D	S	Mode

N = numero linee, F = font, I/D S = Mode Set

Per il modo a 4 bit l'inizializzazione avviene così:

Power ON

Aspetta 15 milli secondi

RS	R/W	D7	D6	D5	D4
0	0	0	0	1	1

Aspetta 4.1 milli secondi

RS	R/W	D7	D6	D5	D4
0	0	0	0	1	1

Aspetta 100 milli secondi

RS	R/W	D7	D6	D5	D4
0	0	0	0	1	1

RS	R/W	D7	D6	D5	D4
0	0	0	0	1	0
0	0	0	0	1	0
0	0	N	F	X	x
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0

0	0	0	0	0	1
0	0	0	0	0	0
0	0	0	1	I/D	S

Il seguente esempio applicativo visualizza una tensione convertita dall'A/D (piedino RA2) e una scritta alternativamente. E' tratto dai testi in bibliografia.

Linguaggio C

```
/*
 * Program name:
     Example 10
 * Copyright:
     (c) MikroElektronika, 2005-2009
 * Description:
             This is a simple program used to demonstrate the operation of
the microcontroller.
 * Configuration:
     Microcontroller: PIC16F887
     Device:               EasyPIC6
     Oscillator:           HS, 08.0000 MHz
     SW:                    mikroC PRO v2.5
 * Notes: -  In order to make this example work properly, it is necessary to
tick off the following
             libraries in the Library Manager prior to compiling:
             -  ADC
             -  LCD
*/

// LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections

unsigned char ch;                          //
unsigned int adc_rd;                       // Declare variables
char *text;                                //
long tlong;                                //

void main() {
  INTCON = 0;                              // All interrupts disabled
```

```
  ANSEL  = 0x04;                          // Pin RA2 is configured as an
analog input
  TRISA  = 0x04;
  ANSELH = 0;                             // Rest of pins are configured as
digital
  Lcd_Init();                             // LCD display initialization
  Lcd_Cmd(_LCD_CURSOR_OFF);               // LCD command (cursor off)
  Lcd_Cmd(_LCD_CLEAR);                    // LCD command (clear LCD)

  text = "mikroElektronika";              // Define the first message
  Lcd_Out(1,1,text);                      // Write the first message in the
first line
  text = "LCD example";                   // Define the second message
  Lcd_Out(2,1,text);                      // Define the first message

  ADCON1     = 0x82;                      // A/D voltage reference is VCC
  TRISA      = 0xFF;                      // All port A pins are configured
as inputs
  Delay_ms(2000);
  text  = "voltage:";                     // Define the third message
  while (1) {
    adc_rd  = ADC_Read(2);                // A/D conversion. Pin RA2 is an
input.
    Lcd_Out(2,1,text);                    // Write result in the second
line

    tlong = (long)adc_rd * 5000;          // Convert the result in
millivolts
    tlong = tlong / 1023;                 // 0..1023 -> 0-5000mV

    ch     = tlong / 1000;                // Extract volts (thousands of
millivolts)
                                          // from result
    Lcd_Chr(2,9,48+ch);                   // Write result in ASCII format
    Lcd_Chr_CP('.');

    ch    = (tlong / 100) % 10;           // Extract hundreds of millivolts
    Lcd_Chr_CP(48+ch);                    // Write result in ASCII format

    ch    = (tlong / 10) % 10;            // Extract tens of millivolts
    Lcd_Chr_CP(48+ch);                    // Write result in ASCII format
    ch    = tlong % 10;                   // Extract digits for millivolts
    Lcd_Chr_CP(48+ch);                    // Write result in ASCII format
    Lcd_Chr_CP('V');

    Delay_ms(1);
  }
}
```

Assembler

```
_main:

;Example10.mbas,43 ::       main:                          ' Start of
program
;Example10.mbas,44 ::       TRISB = 0                     ' All port PORTB
pins are configured as outputs
```

```
      CLRF       TRISB+0
;Example10.mbas,45 ::       PORTB = 0xFF
      MOVLW      255
      MOVWF      PORTB+0
;Example10.mbas,46 ::       INTCON = 0                      ' All interrupts
disabled
      CLRF       INTCON+0
;Example10.mbas,47 ::       ANSEL  = 0x04                   ' Pin RA2 is
configured as an analog input
      MOVLW      4
      MOVWF      ANSEL+0
;Example10.mbas,48 ::       TRISA  = 0x04
      MOVLW      4
      MOVWF      TRISA+0
;Example10.mbas,49 ::       ANSELH = 0                      ' Rest of pins
is configured as digital
      CLRF       ANSELH+0
;Example10.mbas,50 ::       Lcd_Init()                      ' LCD display
initialization
      CALL       _Lcd_Init+0
;Example10.mbas,51 ::       Lcd_Cmd(_LCD_CURSOR_OFF)        ' LCD command
(cursor off)
      MOVLW      12
      MOVWF      FARG_Lcd_Cmd_out_char+0
      CALL       _Lcd_Cmd+0
;Example10.mbas,52 ::       Lcd_Cmd(_LCD_CLEAR)             ' LCD command
(clear LCD)
      MOVLW      1
      MOVWF      FARG_Lcd_Cmd_out_char+0
      CALL       _Lcd_Cmd+0
;Example10.mbas,54 ::       text = "mikroElektronika"       ' Define the
first message
      MOVLW      _text+0
      MOVWF      FSR
      MOVLW      109
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      105
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      107
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      114
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      111
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      69
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      108
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      101
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      107
      MOVWF      INDF+0
```

```
      INCF       FSR, 1
      MOVLW      116
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      114
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      111
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      110
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      105
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      107
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      97
      MOVWF      INDF+0
      INCF       FSR, 1
      CLRF       INDF+0
      INCF       FSR, 1
;Example10.mbas,55 ::       Lcd_Out(1,1,text)               ' Write the
first message in the first line
      MOVLW      1
      MOVWF      FARG_Lcd_Out_row+0
      MOVLW      1
      MOVWF      FARG_Lcd_Out_column+0
      MOVLW      _text+0
      MOVWF      FARG_Lcd_Out_text+0
      CALL       _Lcd_Out+0
;Example10.mbas,56 ::       text = "LCD example"            ' Define the
second message
      MOVLW      _text+0
      MOVWF      FSR
      MOVLW      76
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      67
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      68
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      32
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      101
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      120
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      97
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      109
      MOVWF      INDF+0
```

```
      INCF       FSR, 1
      MOVLW      112
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      108
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      101
      MOVWF      INDF+0
      INCF       FSR, 1
      CLRF       INDF+0
      INCF       FSR, 1
;Example10.mbas,57 ::       Lcd_Out(2,1,text)               ' Write the
second message in the second line
      MOVLW      2
      MOVWF      FARG_Lcd_Out_row+0
      MOVLW      1
      MOVWF      FARG_Lcd_Out_column+0
      MOVLW      _text+0
      MOVWF      FARG_Lcd_Out_text+0
      CALL       _Lcd_Out+0
;Example10.mbas,59 ::       ADCON1     = 0x80               ' A/D voltage
reference is VCC
      MOVLW      128
      MOVWF      ADCON1+0
;Example10.mbas,60 ::       TRISA      = 0xFF               ' All PORTA pins
are configured as inputs
      MOVLW      255
      MOVWF      TRISA+0
;Example10.mbas,61 ::       Delay_ms(2000)
      MOVLW      21
      MOVWF      R11+0
      MOVLW      75
      MOVWF      R12+0
      MOVLW      190
      MOVWF      R13+0
L__main1:
      DECFSZ     R13+0, 1
      GOTO       L__main1
      DECFSZ     R12+0, 1
      GOTO       L__main1
      DECFSZ     R11+0, 1
      GOTO       L__main1
      NOP
;Example10.mbas,62 ::       text  = "Voltage="              ' Define the
third message
      MOVLW      _text+0
      MOVWF      FSR
      MOVLW      86
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      111
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      108
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      116
      MOVWF      INDF+0
      INCF       FSR, 1
```

```
      MOVLW      97
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      103
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      101
      MOVWF      INDF+0
      INCF       FSR, 1
      MOVLW      61
      MOVWF      INDF+0
      INCF       FSR, 1
      CLRF       INDF+0
      INCF       FSR, 1
;Example10.mbas,64 ::       while 1                          ' Endless loop
L__main3:
;Example10.mbas,65 ::       adc_rd  = ADC_Read(2)         ' A/D conversion.
Pin RA2 is an input.
      MOVLW      2
      MOVWF      FARG_ADC_Read_channel+0
      CALL       _ADC_Read+0
      MOVF       R0+0, 0
      MOVWF      _adc_rd+0
      MOVF       R0+1, 0
      MOVWF      _adc_rd+1
;Example10.mbas,66 ::       Lcd_Out(2,1,text)             ' Write result in
the second line
      MOVLW      2
      MOVWF      FARG_Lcd_Out_row+0
      MOVLW      1
      MOVWF      FARG_Lcd_Out_column+0
      MOVLW      _text+0
      MOVWF      FARG_Lcd_Out_text+0
      CALL       _Lcd_Out+0
;Example10.mbas,68 ::       tlong = adc_rd * 5000         ' Convert the
result in millivolts
      MOVF       _adc_rd+0, 0
      MOVWF      R0+0
      MOVF       _adc_rd+1, 0
      MOVWF      R0+1
      CLRF       R0+2
      CLRF       R0+3
      MOVLW      136
      MOVWF      R4+0
      MOVLW      19
      MOVWF      R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Mul_32x32_U+0
      MOVF       R0+0, 0
      MOVWF      _tlong+0
      MOVF       R0+1, 0
      MOVWF      _tlong+1
      MOVF       R0+2, 0
      MOVWF      _tlong+2
      MOVF       R0+3, 0
      MOVWF      _tlong+3
;Example10.mbas,69 ::       tlong = tlong / 1023          ' 0..1023 -> 0-
5000mV
      MOVLW      255
```

```
      MOVWF      R4+0
      MOVLW      3
      MOVWF      R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Div_32x32_U+0
      MOVF       R0+0, 0
      MOVWF      _tlong+0
      MOVF       R0+1, 0
      MOVWF      _tlong+1
      MOVF       R0+2, 0
      MOVWF      _tlong+2
      MOVF       R0+3, 0
      MOVWF      _tlong+3
;Example10.mbas,71 ::       ch = (tlong / 1000) mod 10    ' Extract volts
(thousands of millivolts)
      MOVLW      232
      MOVWF      R4+0
      MOVLW      3
      MOVWF      R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Div_32x32_U+0
      MOVLW      10
      MOVWF      R4+0
      CLRF       R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Div_32x32_U+0
      MOVF       R8+0, 0
      MOVWF      R0+0
      MOVF       R8+1, 0
      MOVWF      R0+1
      MOVF       R8+2, 0
      MOVWF      R0+2
      MOVF       R8+3, 0
      MOVWF      R0+3
      MOVF       R0+0, 0
      MOVWF      _ch+0
      MOVF       R0+1, 0
      MOVWF      _ch+1
;Example10.mbas,73 ::       Lcd_Chr(2,9,48+ch)            ' Write result in
ASCII format
      MOVLW      2
      MOVWF      FARG_Lcd_Chr_Row+0
      MOVLW      9
      MOVWF      FARG_Lcd_Chr_Column+0
      MOVF       R0+0, 0
      ADDLW      48
      MOVWF      FARG_Lcd_Chr_Out_Char+0
      CALL       _Lcd_Chr+0
;Example10.mbas,75 ::       Lcd_Chr_CP(".")               ' Write the
decimal pint
      MOVLW      46
      MOVWF      FARG_Lcd_Chr_CP_Out_Char+0
      CALL       _Lcd_Chr_CP+0
;Example10.mbas,77 ::       ch = (tlong / 100) mod 10     ' Extract hundreds
of millivolts
      MOVLW      100
      MOVWF      R4+0
```

```
      CLRF       R4+1
      CLRF       R4+2
      CLRF       R4+3
      MOVF       _tlong+0, 0
      MOVWF      R0+0
      MOVF       _tlong+1, 0
      MOVWF      R0+1
      MOVF       _tlong+2, 0
      MOVWF      R0+2
      MOVF       _tlong+3, 0
      MOVWF      R0+3
      CALL       _Div_32x32_U+0
      MOVLW      10
      MOVWF      R4+0
      CLRF       R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Div_32x32_U+0
      MOVF       R8+0, 0
      MOVWF      R0+0
      MOVF       R8+1, 0
      MOVWF      R0+1
      MOVF       R8+2, 0
      MOVWF      R0+2
      MOVF       R8+3, 0
      MOVWF      R0+3
      MOVF       R0+0, 0
      MOVWF      _ch+0
      MOVF       R0+1, 0
      MOVWF      _ch+1
;Example10.mbas,78 ::       Lcd_Chr_CP(48+ch)             ' Write result in
ASCII format
      MOVF       R0+0, 0
      ADDLW      48
      MOVWF      FARG_Lcd_Chr_CP_Out_Char+0
      CALL       _Lcd_Chr_CP+0
;Example10.mbas,80 ::       ch = (tlong / 10) mod 10      ' Extract tens of
millivolts
      MOVLW      10
      MOVWF      R4+0
      CLRF       R4+1
      CLRF       R4+2
      CLRF       R4+3
      MOVF       _tlong+0, 0
      MOVWF      R0+0
      MOVF       _tlong+1, 0
      MOVWF      R0+1
      MOVF       _tlong+2, 0
      MOVWF      R0+2
      MOVF       _tlong+3, 0
      MOVWF      R0+3
      CALL       _Div_32x32_U+0
      MOVLW      10
      MOVWF      R4+0
      CLRF       R4+1
      CLRF       R4+2
      CLRF       R4+3
      CALL       _Div_32x32_U+0
      MOVF       R8+0, 0
      MOVWF      R0+0
```

```
      MOVF        R8+1, 0
      MOVWF       R0+1
      MOVF        R8+2, 0
      MOVWF       R0+2
      MOVF        R8+3, 0
      MOVWF       R0+3
      MOVF        R0+0, 0
      MOVWF       _ch+0
      MOVF        R0+1, 0
      MOVWF       _ch+1
;Example10.mbas,81 ::       Lcd_Chr_CP(48+ch)            ' Write result in
ASCII format
      MOVF        R0+0, 0
      ADDLW       48
      MOVWF       FARG_Lcd_Chr_CP_Out_Char+0
      CALL        _Lcd_Chr_CP+0
;Example10.mbas,83 ::       ch = tlong mod 10            ' Extract digits
for millivolts
      MOVLW       10
      MOVWF       R4+0
      CLRF        R4+1
      CLRF        R4+2
      CLRF        R4+3
      MOVF        _tlong+0, 0
      MOVWF       R0+0
      MOVF        _tlong+1, 0
      MOVWF       R0+1
      MOVF        _tlong+2, 0
      MOVWF       R0+2
      MOVF        _tlong+3, 0
      MOVWF       R0+3
      CALL        _Div_32x32_U+0
      MOVF        R8+0, 0
      MOVWF       R0+0
      MOVF        R8+1, 0
      MOVWF       R0+1
      MOVF        R8+2, 0
      MOVWF       R0+2
      MOVF        R8+3, 0
      MOVWF       R0+3
      MOVF        R0+0, 0
      MOVWF       _ch+0
      MOVF        R0+1, 0
      MOVWF       _ch+1
;Example10.mbas,84 ::       Lcd_Chr_CP(48+ch)            ' Write result in
ASCII format
      MOVF        R0+0, 0
      ADDLW       48
      MOVWF       FARG_Lcd_Chr_CP_Out_Char+0
      CALL        _Lcd_Chr_CP+0
;Example10.mbas,86 ::       Lcd_Chr_CP("V")              ' Write a mark for
voltage "V"
      MOVLW       86
      MOVWF       FARG_Lcd_Chr_CP_Out_Char+0
      CALL        _Lcd_Chr_CP+0
;Example10.mbas,88 ::       Delay_ms(1)                  ' 1mS delay
      MOVLW       3
      MOVWF       R12+0
      MOVLW       151
      MOVWF       R13+0
```

```
L__main7:
      DECFSZ     R13+0, 1
      GOTO       L__main7
      DECFSZ     R12+0, 1
      GOTO       L__main7
      NOP
      NOP
;Example10.mbas,89 ::       wend
      GOTO       L__main3
      GOTO       $+0
; end of _main
```

Strumenti di sviluppo

Software

Oltre a quelli citati nella Parte I:

-MicroBasic di Mikroelektronica (società di consulenza per la Microchip, in Serbia), versione free limitata a programmi non superiori a 2k bytes.

-MicroC di Mikroelektronica, come sopra.

Hardware

-piastra di sviluppo per PIC a 8, 14, 18, 28, 40 pin di MikroElektronika.

Bibliografia

-Fernando E. Valdes-Perez, Ramon Pallas-Areny, Microcontrollers, CRC Press, Boca Raton, Florida, 2009

-Milan Verde, PIC Microcontrollers, Programming in Basic, MikroElektronika, Belgrado, 2009

-Milan Verde, PIC Microcontrollers, Programming in C, MikroElektronika, Belgrado, 2009