Il mondo dei microcontroller

Dal 1977, quando Intel mise in commercio la sua prima famiglia di microcontroller, ad oggi, questi piccoli computer sono entrati nelle nostre vite fino a diventare praticamente indispensabili.
Nell'anno 2000 furono venduti circa 7.500 milioni di microcontroller, dato significativo se si tiene conto che la popolazione mondiale di quell'anno era stimata in 6.000 milioni di persone, però sappiamo cos'è un microcontroller?
La maggioranza delle persone non ne ha mai visto uno, né lo vedrà: ignorano la sua esistenza e non hanno la necessità di conoscerla.
Invece, loro sono lì, semplificano la nostra vita, facendoci risparmiare tempo, facilitandoci in molte operazioni, offrendoci delle comodità,
per questo e altri motivi stanno conquistando sempre più spazio tra di noi.

Che cos'è un microcontroller?

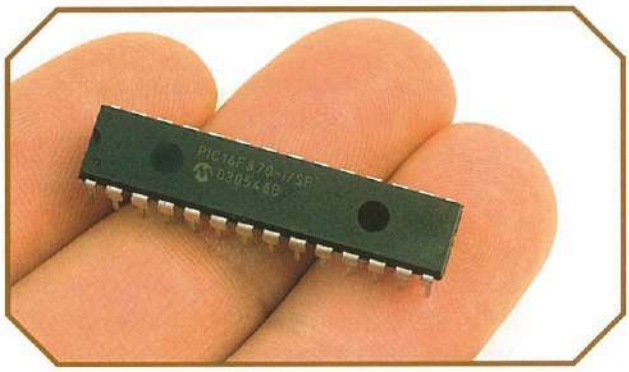
Un microcontroller è un circuito integrato che contiene gli elementi necessari per gestire uno o più processi. Si tratta di un computer ridotto alla minima espressione e integrato in un piccolo chip.

È importante stabilire la differenza tra microprocessore e microcontroller. Un microprocessore è un circuito integrato che contiene l'Unità Centrale di Processo (in inglese CPU) di un computer, mentre un microcontroller è un circuito integrato che contiene tutti i blocchi di un computer.

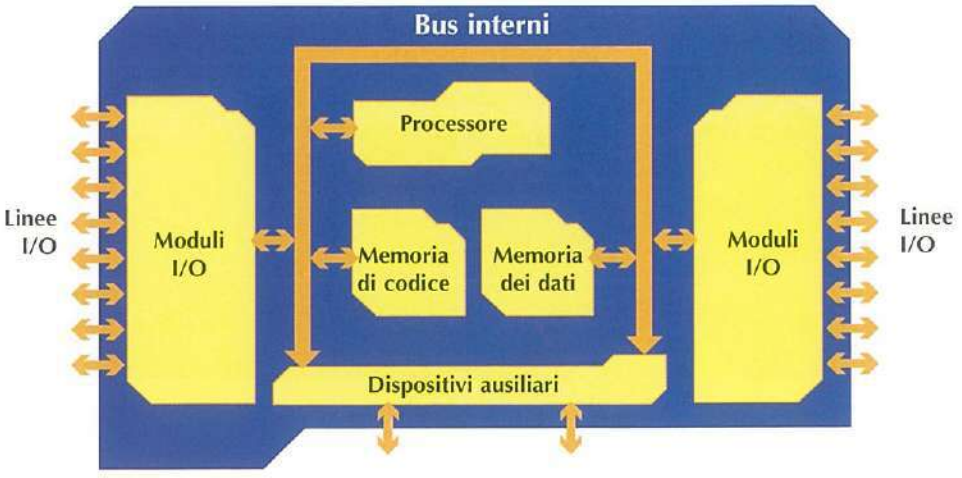
Anche se un microcontroller a prima vista può sembrare un dispositivo molto più complesso di un microprocessore, in realtà non è così, dato che, anche se il primo supporta un computer completo, è piuttosto semplice, mentre il microprocessore implementa con un maggior numero di transistor una CPU molto potente.

Dove sono i microcontroller?

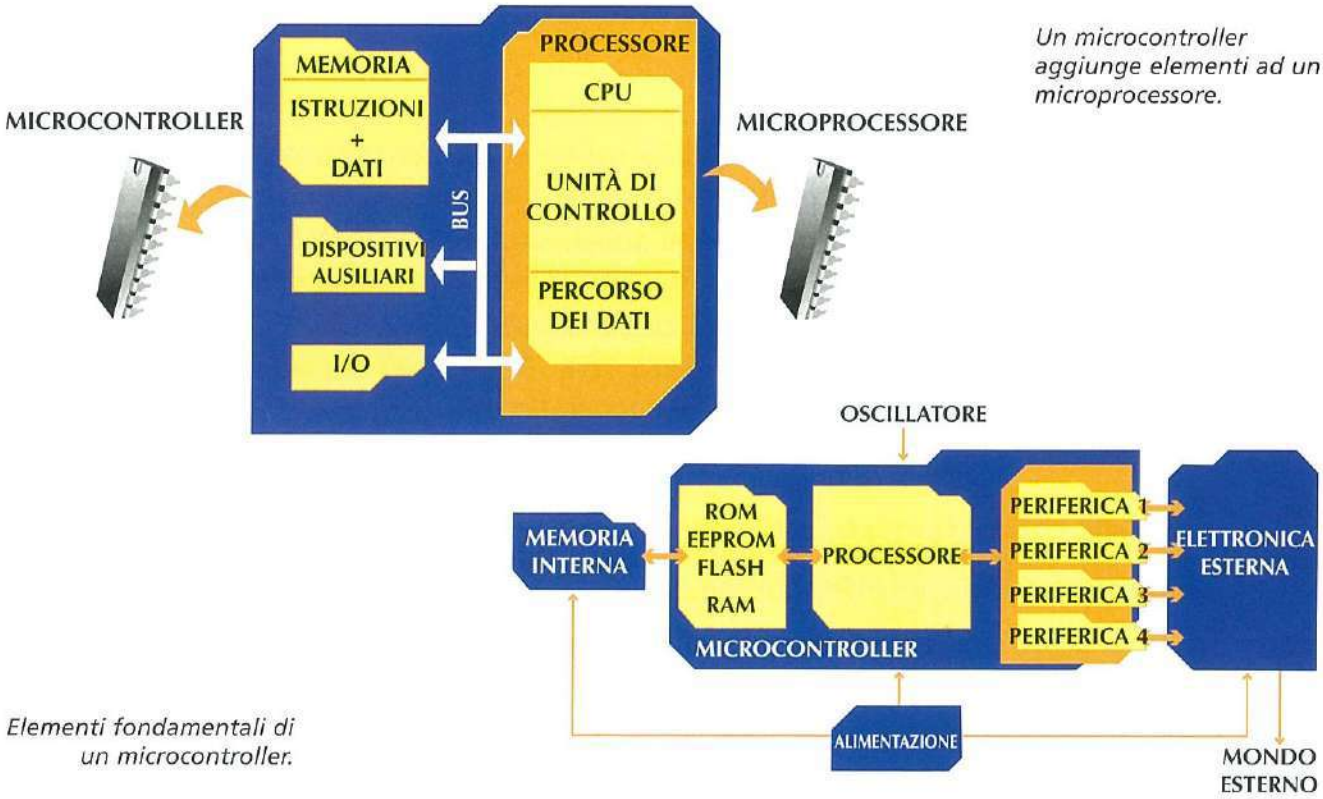
I microcontroller sono ovunque, sono dei chip onnipresenti capaci di risolvere compiti di diversa complessità. È tale la loro presenza che sarebbe molto difficile concepire la vita attuale senza di essi.



Microcontroller 16F870.



Nei microcontroller le memorie dei codici e dei dati sono separate.

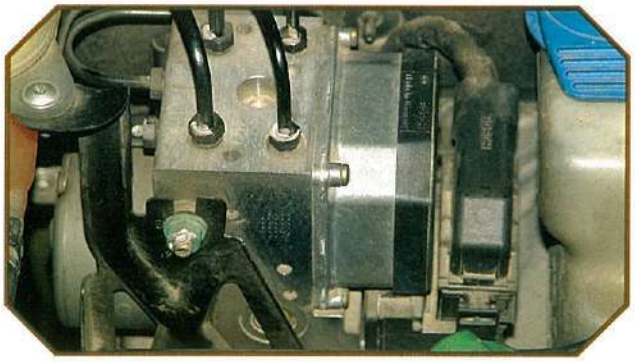


Un microcontroller aggiunge elementi ad un microprocessore.

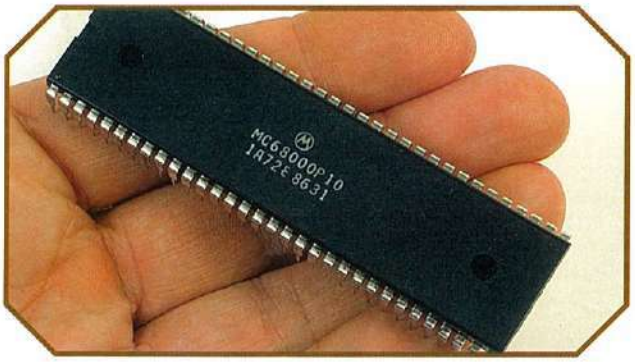
Elementi fondamentali di un microcontroller.

Da quando ci svegliamo fino a quando ci addormentiamo conviviamo con essi, e possono addirittura rendere più confortevole il nostro sonno. Sono nella sveglia, in cucina, nel telefono, nell'ascensore, nella macchina, nei semafori, al lavoro, nei centri di studio, nel televisore, nella catena musicale, oppure possono essere addirittura dentro di te impedendo che il tuo cuore smetta di battere.

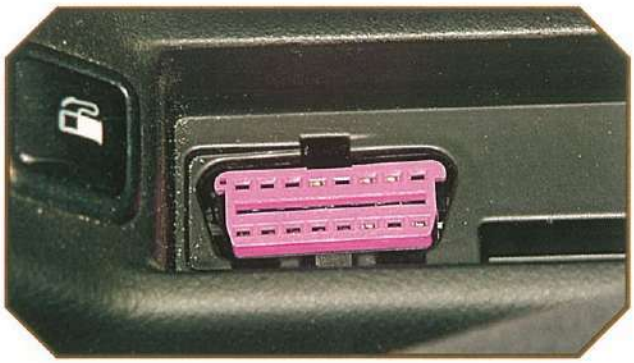
Questa intelligente invasione è in piena progressione. La vendita dei microcontroller ha un aumento esponenziale, però, fino a quando continuerà? Per rispondere a questa domanda dobbiamo guardarci indietro, dal momento che l'unica cosa di cui possiamo essere certi è che i microcontroller hanno conquistato il nostro pianeta, hanno viaggiato nelle profondità dei mari, sono stati i primi a raggiungere altri pianeti, però hanno sempre lavorato per realizzare compiti per i quali gli esseri umani li hanno programmati.



La centralina ABS di un'automobile, è controllata da un microprocessore.



Il microprocessore 68000 di Motorola a 10 Mhz, ha circa 15 anni.



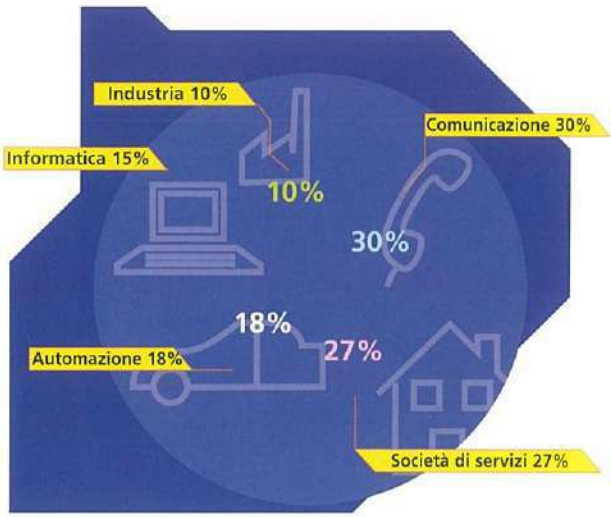
Connettore di un'automobile per diagnostica, si collega a un computer.

Campi di applicazione

Il numero di dispositivi elettronici che lavorano con microcontroller cresce ogni giorno. Si stima che la media dei microcontroller che possiamo trovare nella nostra casa è nell'ordine dei 300. Tenendo conto dello sviluppo e del momento favorevole che sta vivendo la domotica o della tendenza verso le case intelligenti, è possibile che questo numero aumenti considerevolmente. Un altro grande consumatore di microcontroller è l'industria automobilistica. Per mezzo dei microcontroller si gestiscono i freni, la sicurezza, gli allarmi, i controlli del motore e delle luci, computer di bordo e una grande varietà di funzioni che caratterizzano il comportamento di un'automobile.

L'industria delle comunicazioni e il boom della telefonia mobile hanno contribuito anch'esse ad incrementare la domanda. I microcontroller, inoltre, si utilizzano nei dispositivi quali agende elettroniche, apparecchi portatili, nelle carte di credito, ecc...

L'industria informatica li utilizza per costruire la maggior parte delle periferiche di un personal computer e l'industria in generale supera il 10% del consumo mondiale di microcontroller. Li possiamo trovare nei distributori automatici, nei giochi, nella strumentazione, nei controlli industriali, nella robotica, nell'elettromedicina, nei sistemi di navigazione spaziale, nella termoregolazione, e in moltissimi altri campi, il cui limite è solo l'immaginazione del progettista.

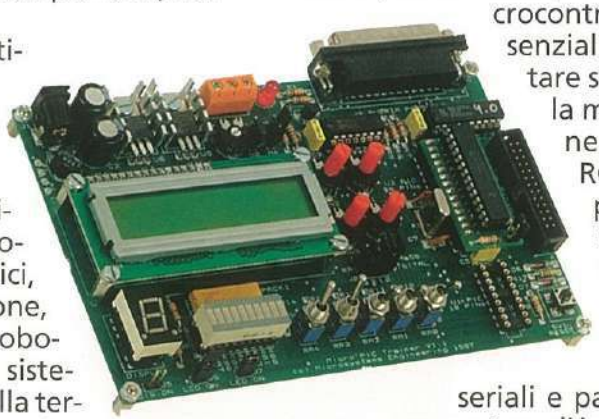


Utilizzo dei microcontroller.

Caratteristiche generali dei microcontroller

Quando si integra un microcontroller in un prodotto si dota quest'ultimo di maggiori prestazioni, si migliora la sua flessibilità e si diminuisce la sua dimensione, il suo consumo e il suo prezzo. Tutto questo grazie allo sviluppo tecnologico di cui hanno goduto questi circuiti integrati, che permette di disporre di un computer semplice ma di alta qualità in un minimo spazio e a un prezzo ragionevole.

L'offerta dei microcontroller è vasta e molto articolata, ne esiste una grandissima varietà di modelli che si adattano alle necessità specifiche delle applicazioni cui sono destinati. A parte queste grandi diversità, tutti devono avere alcune particolarità per potersi considerare microcontroller. I componenti essenziali su cui tutti devono contare sono il processore o CPU, la memoria RAM per contenere i dati, la memoria ROM/PROM/EPROM per il programma, le linee di ingresso/uscita (I/O) per comunicare con l'esterno, i moduli per il controllo delle periferiche (temporizzatori, porte seriali e parallele, CAD, ecc.) e un generatore di impulsi di clock per sincronizzare il funzionamento di tutto il sistema.



Sistema per lo studio e la sperimentazione con microcontroller.

FABBRICANTE	MODELLO DI MICROCONTROLLER
Arizona Microchip	PICs
Atmel	ATmegaXX, AT91FR40162
Dallas Semiconductor	Clone dell' 8051
Hitachi	HD64180
Intel	8048, 8051, 80C196, 80186, 80188, 80386EX
Motorola	6805, 68HC11, 68HC12, 68HC16, 683XXX
Nacional Semiconductor	COP400, COP800, COP8
Philips	Clone dell' 8051
SGS-Thomson	(ST) ST-62XX
Texas Instrument	TMS370
Toshiba	TLCS-870
Zilog	Z8, Z80, Z180, Z86XX

Macchina multifunzione contenente microprocessori.

Secondo il modello di microcontroller varierà la dimensione, il tipo di memoria, il numero di ingressi e uscite e i moduli che contiene per controllare le periferiche.

I costruttori

La grande domanda di questo tipo di chip ha generato un'alta competenza e ormai la maggior parte dei costruttori di circuiti integrati dispone di una propria gamma di microcontroller.

Intel è stata l'azienda che ha introdotto il concetto di microcontroller. Furono i pionieri e sono sempre un passo avanti a tutti gli altri costruttori. All'interno dell'ampia gamma di cui dispone ricordiamo l'8051, uno standard nell'industria. Aziende come Philips o Dallas Semiconductor commercializzarono microcontroller cloni di questi prodotti.

Motorola, che nel 1979 aveva lanciato il suo processore 68000, utilizzò il suo mercato per lanciare microcontroller quali il 68HC11, la cui potenza lo fece apprezzare da tutto il mercato mondiale. Con la sua gamma di prodotti, Motorola è stato il costruttore di maggior produzione e vendita mondiale negli ultimi dieci anni.

Arizona Microchip, costruttore dei PIC, ha raggiunto la quota di mercato più importante grazie a questi prodotti che riuniscono caratteristiche quali il basso costo, consumo ridotto, piccole dimensioni e facilità di programmazione. Altre caratteristiche cui Microchip e i suoi PIC devono il proprio successo sono la grande varietà ed economia degli strumenti di supporto e la vasta informazione tecnica dei propri prodotti.

Microprocessore Pentium.

Altri costruttori come Atmel, Hitachi, Nacional Semiconductor, SGS-Thomson, Texas Instruments, ecc, offrono modelli di microcontroller competitivi che occupano il loro settore in questo mercato.

Conclusioni

Il microcontroller nasce dalla necessità di un'elettronica economica, dalle ridotte dimensioni, intelligente e ri-programmabile. La riprogrammazione permette a questi dispositivi di essere riutilizzabili.

Inoltre, la programmazione di questi dispositivi permette di poterli meglio adattare alle nostre applicazioni e a modificarne le funzioni senza dover ricorrere all'hardware, cosa che implicherebbe maggiori costi. Questa qualità riduce anche il tempo di sviluppo, poiché un'implementazione di controllo via software è sempre meno complessa di un'implementazione hardware per lo stesso sistema.

Il successo dei microcontroller è dovuto a molti fattori che hanno contribuito alla loro capillare diffusione e, di conseguenza, allo sviluppo e al miglioramento delle condizioni di vita. Questi chip, per realizzare il proprio compito, dipendono da noi, però, fino a che punto noi dipendiamo da loro?

Caratteristiche

Ci siamo fatti una prima idea di cosa sono in grado di fare i microcontroller, ma, non conoscendo il loro funzionamento interno, non sappiamo ancora come possano realizzare molteplici funzioni. Come è possibile che un chip così piccolo risolva tanti problemi?

La tecnologia con cui si fabbricano i microcontroller, così come quella di molti circuiti integrati, è la CMOS (Complementary Metal Oxide Semiconductor). I dispositivi CMOS, oltre ad avere caratteristiche quali basso consumo, elevata immunità al rumore, ecc., hanno un'alta integrazione di transistor in uno spazio ridotto.

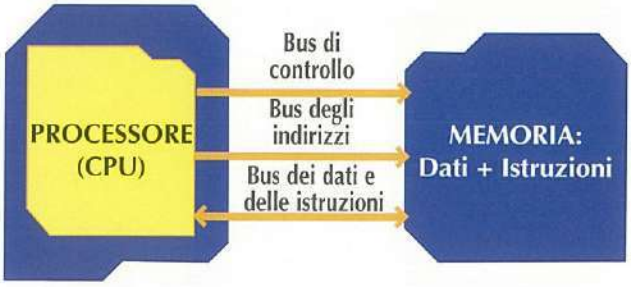
Questa caratteristica permette ad un piccolo chip di essere così funzionale, ma allo stesso tempo, è anche una limitazione che fa sì che non esistano molte differenze fra i diversi tipi di microcontroller. La struttura e le caratteristiche fondamentali di questi ultimi sono simili.

Tutti i microcontroller si assomigliano, anche se tutti sono diversi, per questo è necessario spiegare i dispositivi comuni a tutti, tenendo sempre presente le differenti alternative offerte in ogni modello.

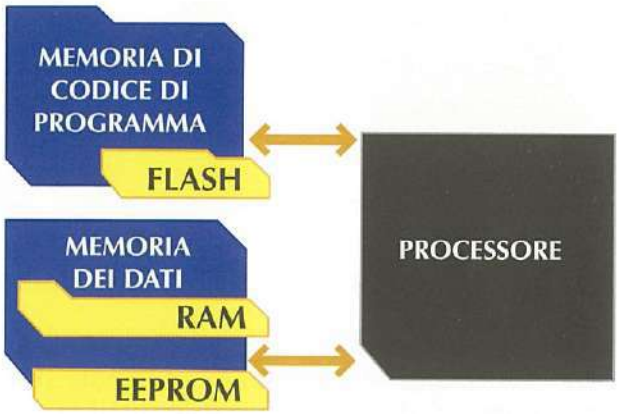
Tipi di architettura

Inizialmente i microcontroller presentavano un'architettura Von Neumann, però col tempo si impose l'architettura Harvard, adottata tuttora da tutti i microcontroller.

L'architettura Von Neumann è caratterizzata dal fatto che la CPU si collega ad una memoria unica dove coesistono dati e istruzioni tramite un unico sistema di bus (indirizzi, dati e controllo). Questo provoca un ritardo nel sistema, infatti quando la CPU si indirizza alla memoria, prima deve estrarre le istruzioni e dopo i dati per poter eseguire questa istruzione.



Architettura tipo Von Neumann.



Tipi di memoria utilizzati nei PIC 16F87X.

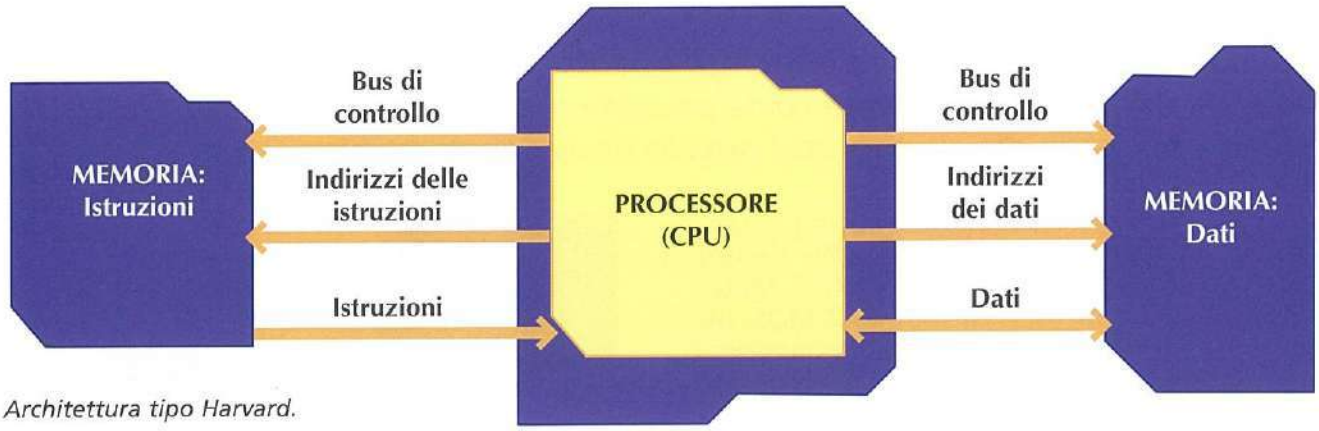
L'architettura Harvard, al contrario, dispone di una memoria per i dati e un'altra per le istruzioni. Le memorie sono indipendenti l'una dall'altra e dispongono di un proprio sistema di bus per l'accesso. Questa caratteristica permette il parallelismo, ovvero, si possono realizzare operazioni simultanee su entrambe le memorie, estraendo contemporaneamente istruzioni e dati, ne consegue, quindi, un elevato rendimento nell'elaborazione delle istruzioni.

Il processore

Il processore è l'elemento più importante del microcontroller e ne determina le sue principali caratteristiche, sia a livello hardware che software.

Ha il compito di indirizzare la memoria delle istruzioni, di ricevere il codice operativo (opcode) dell'istruzione in corso, della sua decodificazione e dell'esecuzione dell'operazione aritmetica o logica che implica l'istruzione. Realizza anche la ricerca degli operandi e la memorizzazione del risultato.

Facendo riferimento all'architettura e



Architettura tipo Harvard.

alla funzionalità dei processori, attualmente tre orientamenti:

– La filosofia CISC riguarda i microcontroller con un insieme di istruzioni complesse, normalmente più di 80, alcune delle quali possono richiedere molti cicli di clock per la loro esecuzione data la loro complessità e potenza.

– La filosofia RISC viene adottata da gran parte dei microcontroller e implica un insieme di istruzioni ridotte che si basano su operazioni semplici generalmente eseguite in un ciclo di clock, questo significa semplicità e velocità.

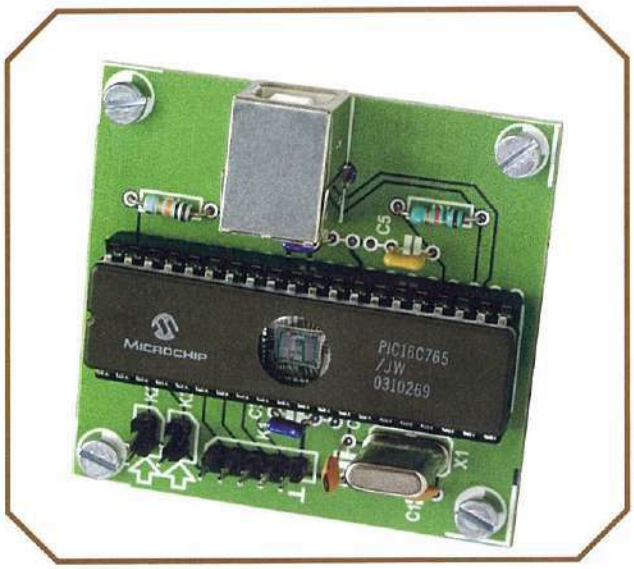
– In ultimo, la filosofia SISC si applica ai microcontroller destinati a compiti molto specifici, quindi adattano le loro istruzioni alle necessità dell'applicazione cui sono dedicati.

La memoria di programma

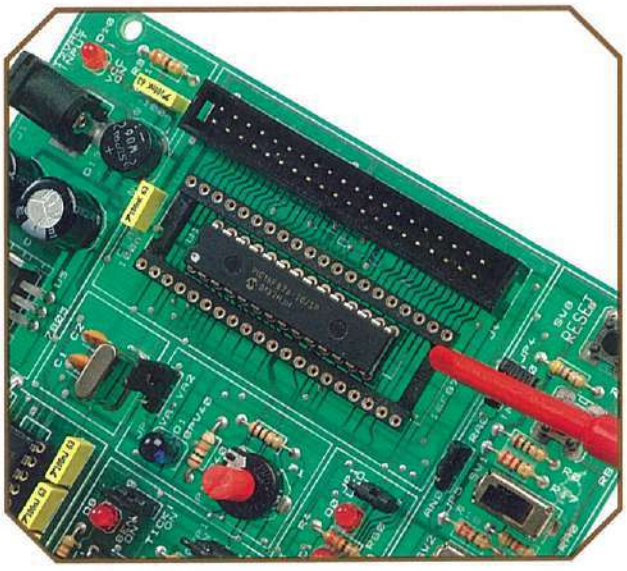
Nella memoria di programma del microcontroller sono contenute tutte le istruzioni del programma di controllo, e ovviamente sono memorizzate in modo permanente. Per fare ciò si possono utilizzare diversi tipi di memoria:

– ROM mascherata. Si tratta di una memoria non volatile di sola lettura il cui contenuto viene scritto durante il processo di fabbricazione del chip. Gli alti costi di progetto della maschera permettono di utilizzarla solo in grandi serie.

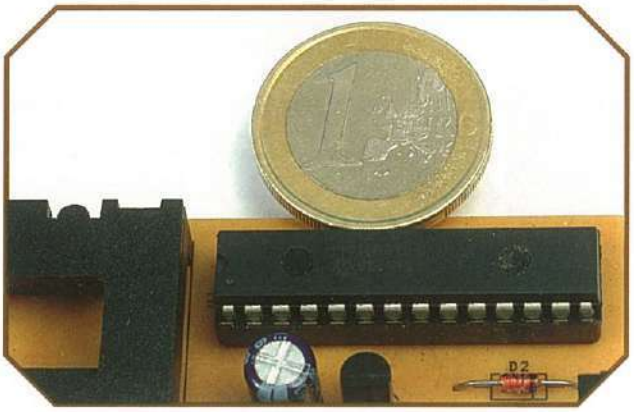
– EPROM. Il microcontroller ha, sulla superficie del suo contenitore, una finestra tramite la quale può essere sottoposto ai raggi ultravioletti.



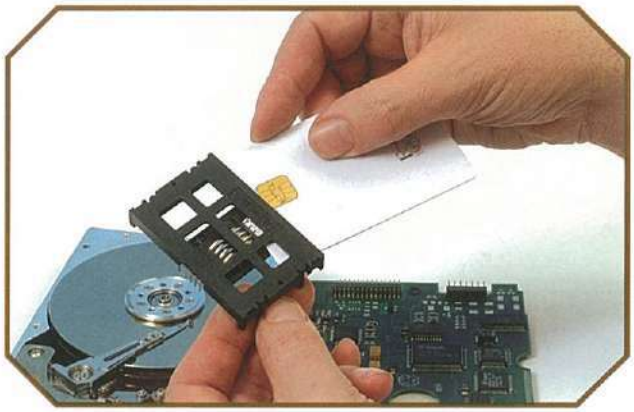
Finestra di cancellazione di una memoria EPROM.



Microcontroller 16F876 di Microchip.



Microcontroller 16F870 di Microchip.



Scheda di memoria.

Mediante questi raggi è possibile cancellare la memoria per il suo successivo riutilizzo. Questo particolare tipo di contenitore rende elevato il costo del microcontroller.

– OTP. Si possono scrivere solamente una volta, però può farlo lo stesso utente mediante un semplice scrittore collegato al PC. Il loro prezzo è ridotto.

– EEPROM. Così come le memorie EPROM e OTP, la loro scrittura si realizza mediante segnali elettrici, le EEPROM, però, possono anche essere cancellate mediante segnali elettrici tante volte quante si desidera.

– FLASH. Si tratta di una memoria non volatile di basso consumo che si può cancellare e scrivere mediante segnali elettrici e che presenta, di solito, una capacità elevata e una maggiore velocità di accesso.

La memoria dei dati

Quando si gestiscono dati in un programma, questi variano continuamente, quindi la memoria che li contiene deve necessariamente essere di lettura e scrittura. Normalmente i dati si memorizzano nella memoria RAM, che è una memoria volatile.

Esistono microcontroller che possiedono una memoria di tipo EEPROM per contenere i dati, quindi questi non vanno persi in caso di mancanza di tensione elettrica, anche se si tratta di casi rari.

Le porte di ingresso e uscita

Il principale utilizzo delle linee di ingresso e uscita consiste nel mettere in comunicazione il computer interno con le periferiche esterne, anche se possono servire per alimentare il microcontroller, fornire una frequenza di lavoro e resettarlo.

A seconda del modello, il microcontroller supporta diversi controller di periferica e, in base ad essi, le linee di ingresso e uscita (I/O) sono destinate a supportare i segnali di ingresso, uscita e controllo.

Tra i dispositivi di questo tipo su cui può contare un microcontroller troviamo la UART (adattatore di comunicazione seriale asincrona), la USART (adattatore di comunicazione seriale sincrona e asincrona), la porta parallela slave (per collegare i bus degli altri microprocessori), il bus I2C, ecc.

Il clock principale

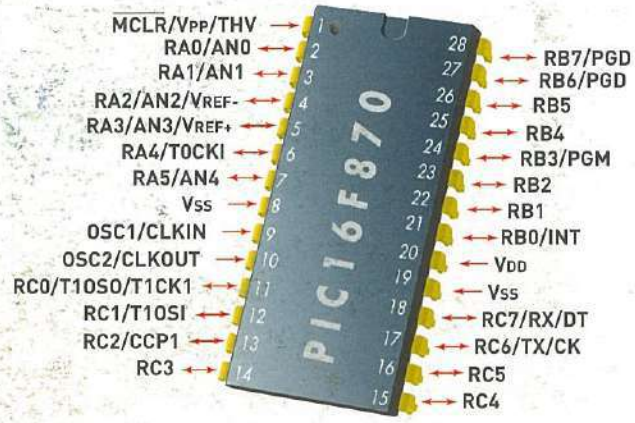
Tutti i microcontroller dispongono di un circuito oscillatore che genera una frequenza. Mediante questa frequenza configuriamo gli impulsi di clock che sincronizzano tutte le operazioni del sistema.

Generalmente, il circuito di clock è integrato nel microcontroller e sono necessari solamente pochi componenti esterni per selezionare e stabilizzare la frequenza di lavoro.

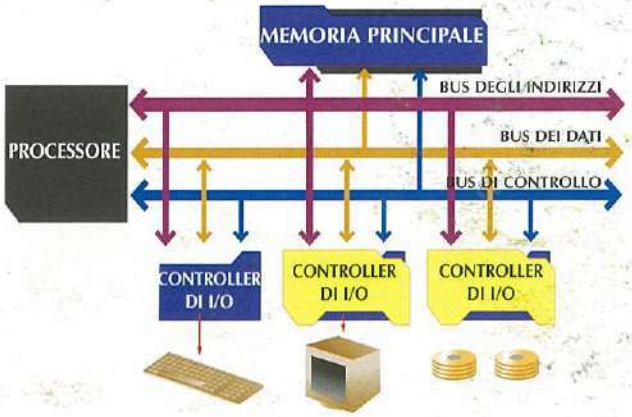
Una maggiore frequenza di clock implicherà una diminuzione del tempo di esecuzione delle istruzioni, ma anche un incremento nel consumo di energia.

Dispositivi speciali

Secondo le applicazioni, ogni modello di microcontroller integra una varietà di



Distribuzione dei terminali del 16F870.



Le porte scambiano informazioni con l'esterno.

complementi che aumentano la potenza del dispositivo. Possono essere dotati di grande capacità di memoria, avere una gamma completa di dispositivi o, semplicemente, essere ridotti ad una architettura di base con prestazioni minime per applicazioni molto semplici.

I principali dispositivi specifici che integrano i microcontroller sono:

– Temporizzatori o timer: si utilizzano per controllare periodi di tempo (temporizzatori) o per eseguire il conteggio di eventi esterni (contatori).

– Watchdog: è un temporizzatore che quando va in overflow e passa per lo 0 provoca un reset del sistema. Il programma dovrà reinizializzare il conteggio del watchdog prima che questo provochi il reset. In questo modo, in caso di errori o blocchi del programma, il temporizzatore non viene reinizializzato e provoca il reset.

– Protezione da calo di tensione o Brownout: si tratta di un circuito che resetta il microcontroller quando la tensione di alimentazione risulta essere inferiore a quella minima (brownout).

– Stato di riposo o basso consumo: stato di risparmio di energia in cui entra il microprocessore quando non è operativo e da cui esce quando si produce un interrupt che lo avvisa del fatto che si è verificato l'evento previsto.

– Convertitore A/D: permette al microcontroller di elaborare segnali analogici.

– Convertitore D/A: trasforma i dati digitali ottenuti dall'elaborazione al loro corrispondente valore analogico.

– Comparatore analogico: amplificatore operazionale interno che funziona come comparatore fra un segnale fisso di riferimento e un altro variabile. L'uscita del comparatore fornisce un livello logico 1 o 0, a seconda se un segnale sia maggiore o minore di un altro.

– Modulatore dell'ampiezza degli impulsi o PWM: sono circuiti che, alla loro uscita, forniscono impulsi ad ampiezza variabile.

Trattamento dei dati analogici e/o digitali.

Che microcontroller usare?

Una volta definito il progetto per una determinata applicazione con microcontroller, occorre scegliere il più adeguato.

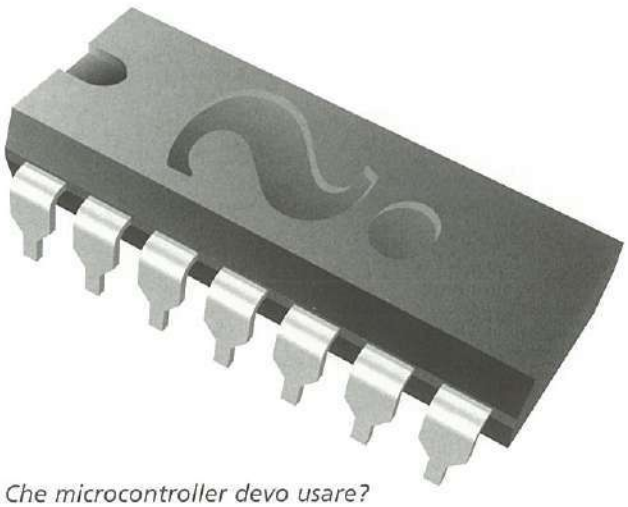
I costruttori di microcontroller sono costantemente impegnati nello sviluppo di questi chip, nel tentativo di renderli sicuri e affidabili, questo però complica il lavoro del progettista, dato che deve scegliere il microcontroller che meglio soddisfa i requisiti del suo progetto, in questo caso il Laboratorio di Elettronica Digitale.

Fattori da considerare

Al momento di scegliere un microcontroller, i fattori di cui tener conto sono molteplici, e dipendono dalla specifica applicazione, alcuni avranno più importanza di altri.

Il costo, la documentazione e gli strumenti di sviluppo disponibili, il prezzo di questi ultimi, la quantità dei costruttori che lo producono e le caratteristiche proprie del microcontroller (memoria, porte di I/O, temporizzatori, ecc.) determineranno la scelta.

Per scegliere correttamente, la prima cosa da fare è un'analisi dei diversi requisiti dell'applicazione a cui sarà destinato il microcontroller:



Che microcontroller devo usare?

Costruttore	Microcontroller	Frequenza (MHz)
Motorola	M68HC05 Family	4, 2.2, 2.1, 2, 1.05
	M68HC08 Family	8.4, 8.2, 8, 4.1, 4
	M68HC11 Family	5, 4, 3, 2, 1
Atmel	Atmel AVR 8 bits	16
	Atmel AT91 8 bits	82, 47, 41,40,38,29
	Atmel 8051 8 bits	66, 60, 40, 33, 26,25
STMicroelectronics	ST5 Family	20
	ST6 Family	8
	ST7 Family	8
	ST9 Family	25
Microchip	PIC12 Family	20, 10
	PIC16 Family	20
	PIC17 Family	33
	PIC18 Family	40

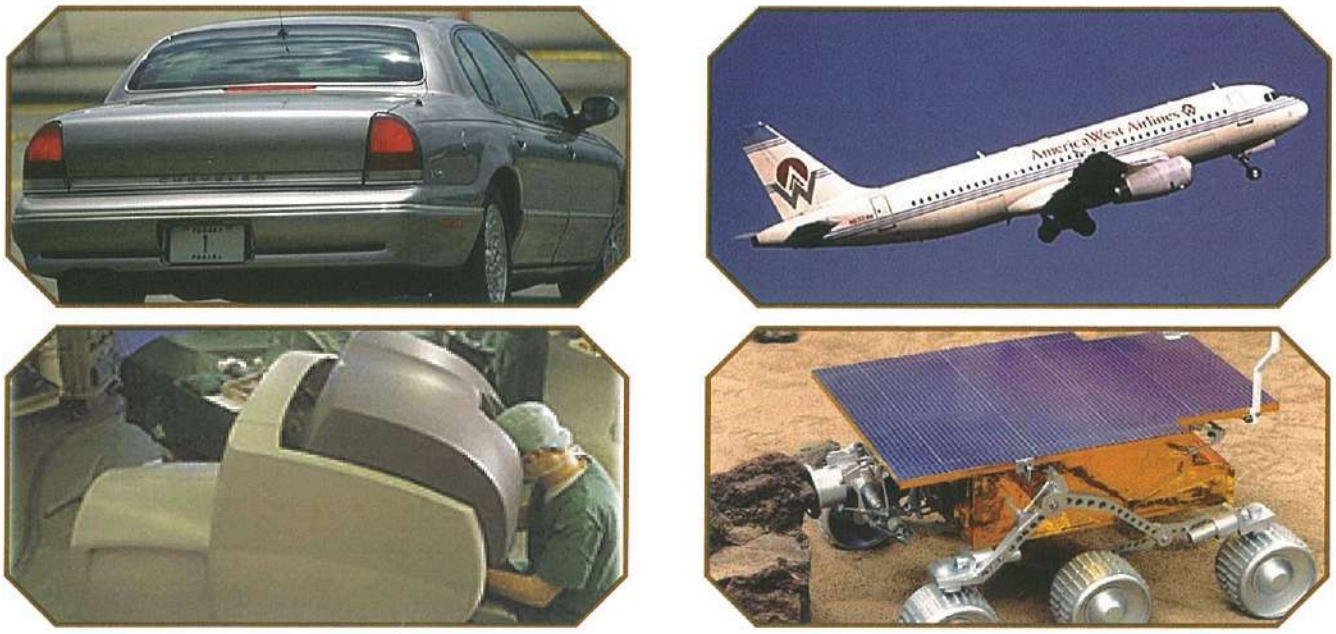
Tabella comparativa delle velocità per le principali famiglie di microcontroller.

Costi

I microcontroller sono dei dispositivi che vengono integrati all'interno di un sistema per sviluppare un'applicazione specifica. Il loro prezzo sarà in funzione del prodotto finale cui sono destinati.

Così, se si utilizza un controller per soddisfare le necessità di un sistema specifico, all'interno di un progetto unico, si valorizzeranno altri aspetti prima del prezzo; per contro, se fa parte di un prodotto che deve arrivare sul mercato con un prezzo competitivo, come ad esempio un riproduttore di DVD o un semplice distributore, il prezzo sarà uno dei principali fattori da prendere in considerazione. All'interno del fattore costi si considerano anche gli strumenti di sviluppo.

Infatti, il costruttore inserisce all'interno del prezzo del chip l'ammortizzamento di tutto ciò che è relativo ad esso (documentazione, strumenti, ecc.). Per questo, solo i grandi costruttori con elevate capacità di vendita possono offrire un prodotto potente ad un prezzo accessibile e con gli strumenti per facilitarne l'utilizzo.



Applicazioni in cui dei valori inadeguati di velocità e precisione possono avere conseguenze fatali.

Elaborazione dei dati

Cosa succederebbe se il chip che controlla il sistema ABS di un'automobile impiegasse uno o due secondi per elaborare l'informazione e dare la risposta all'attuatore? Esistono applicazioni per le quali la velocità di elaborazione è molto importante, dato che devono realizzare calcoli critici in un tempo determinato. Il progettista si deve assicurare di scegliere un microcontroller sufficientemente veloce per rispondere ai requisiti del sistema.

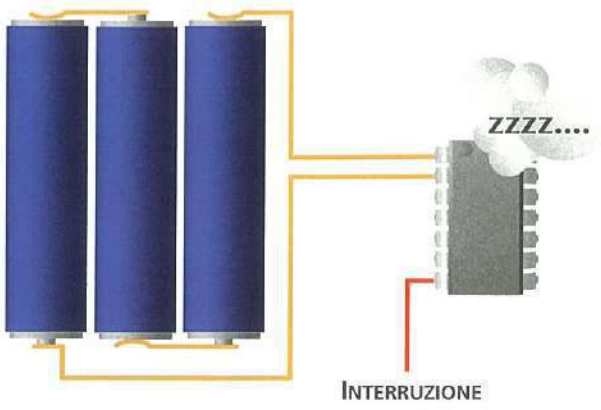
Supponiamo che la nostra applicazione sia il controllo di un robot chirurgico in cui il posizionamento dell'elemento terminale deve essere fatto con una precisione al centesimo di millimetro. In base alla precisione dei dati da gestire dovremo ricorrere a microcontroller da 8, 16 o 32 bit, compresi quelli con hardware a virgola mobile. Questa dimensione della word è direttamente proporzionale ai costi, dato che ad una maggiore dimensione di word corrisponde un microcontroller più caro.

Ingressi e uscite

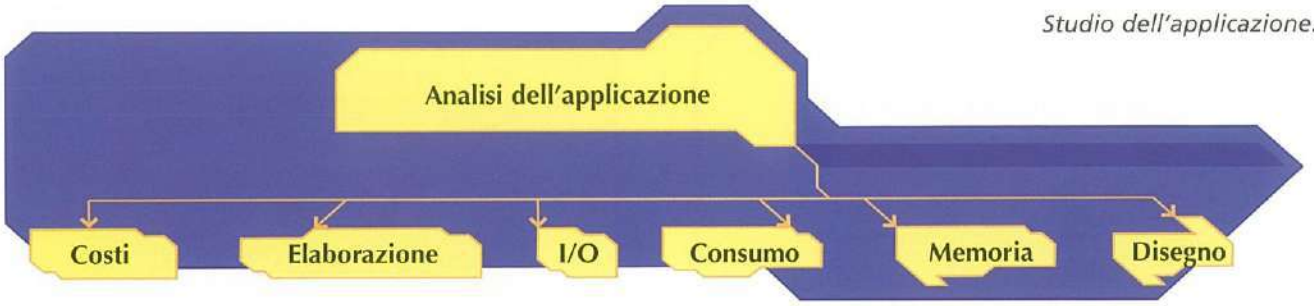
In tutte le applicazioni avremo dei parametri di ingresso in base ai quali attiveremo le uscite. Dobbiamo definire sia il numero dei segnali di ingresso e di uscita necessari per la nostra applicazione e la natura di essi: analogica o digitale. Spesso è conveniente scegliere un microcontroller che abbia capacità relative ai segnali di ingresso e di uscita superiori a quelle richieste, per poter far fronte ad eventuali future espansioni.

Consumo

Sono molte le applicazioni in cui il microcontroller è alimentato con batterie. In questi casi bisogna offrire la possibilità dello stato di basso consumo. Per capire questo immagineremo



Stato di riposo del microcontroller.



Studio dell'applicazione.

un microcontroller utilizzato in un controllo remoto da una porta di garage; normalmente il chip è in stato di riposo – basso consumo – fino a quando non si preme il pulsante e si produce un interrupt: a quel punto si sveglia, realizza il compito per il quale è stato programmato, per poi tornare in stato di riposo fino ad un nuovo interrupt.

Se il chip non presentasse questa caratteristica dovremmo cambiare le batterie molto frequentemente.

Memoria

Per l'analisi della memoria necessaria, la prima cosa che dobbiamo conoscere è se il nostro microcontroller è riutilizzabile o riprogrammabile. In seguito, dovremo studiare la quantità di dati da immagazzinare all'interno del chip, la lunghezza del programma, i dati con cui lavoreremo, ecc. Infine, una volta conosciuta la quantità, dobbiamo classificarla, per sapere se abbiamo bisogno di una memoria volatile, non volatile o modificabile.

Progetto del circuito stampato

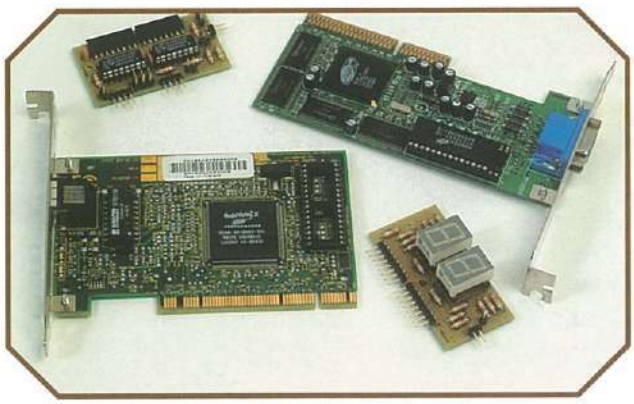
In funzione delle richieste del volume disponibile, della dimensione del circuito elettronico o della tecnologia di fabbricazione di quest'ultimo, ricorreremo a diversi tipi di contenitore per il microcontroller.

Contenitori DIP o DIL

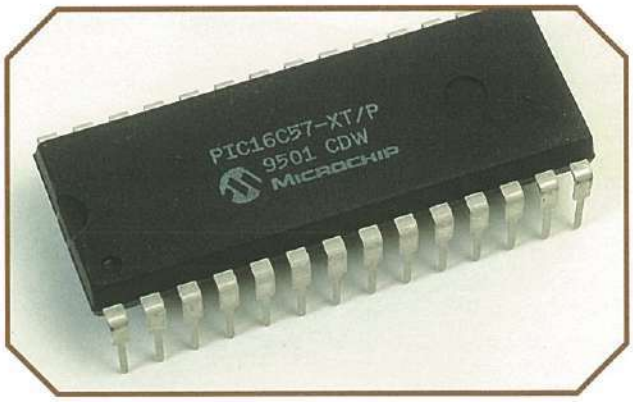
Questo contenitore è il più comune. È stato pensato per circuiti stampati convenzionali a fori passanti, e può essere ceramico o plastico. La distanza standard tra i terminali, o "pin", è di 2,54 mm. Come ultima caratteristica principale, i chip con questi contenitori presentano lo stesso formato di identificazione dei pin: il pin numero uno si trova su uno degli estremi, indicato da un punto o una tacca di riferimento, e partendo da questo, si procede alla numerazione in senso antiorario (senso contrario a quello delle lancette dell'orologio), guardando l'integrato dall'alto.

Contenitore FLAT-PACK

La sua forma facilita la saldatura in macchine automatiche o semiautomatiche, dato che, per la disposizione dei suoi terminali, è possibile saldare per punti. È costruito in materiale ceramico. La distanza tra i terminali è di

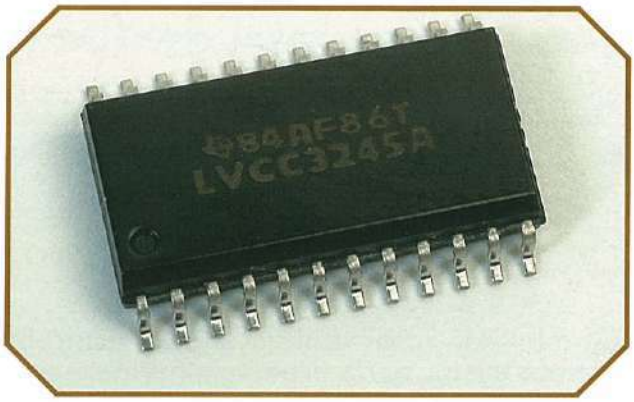


Tecnologia usata nei PCB.



Contenitore DIP.

Contenitore FLAT-PACK.



Contenitore SOIC.

1,27 mm, la metà rispetto ai DIP, anche se la numerazione dei pin è simile alla precedente.

Contenitore SOIC

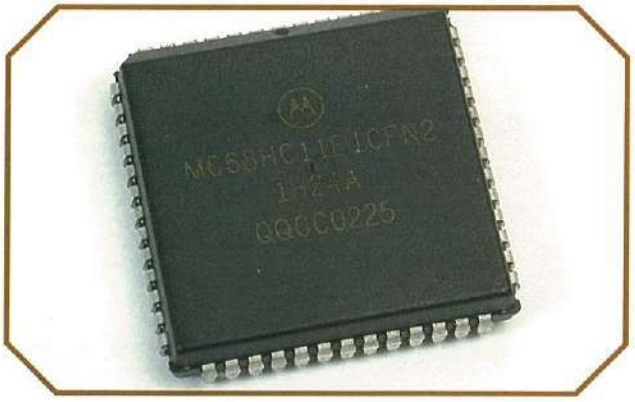
I circuiti integrati con questo tipo di contenitore sono caratterizzati da piccole dimensioni, sono infatti molto stretti. Questi circuiti sono i più comuni nella logica combinazionale, e si saldano direttamente sulle piste del circuito stampato. La distanza tra i terminali è di 1,27 mm e la numerazione si realizza come nei casi precedenti. Si saldano con tecniche SMD, cioè, a montaggio superficiale.

Contenitore LPCC

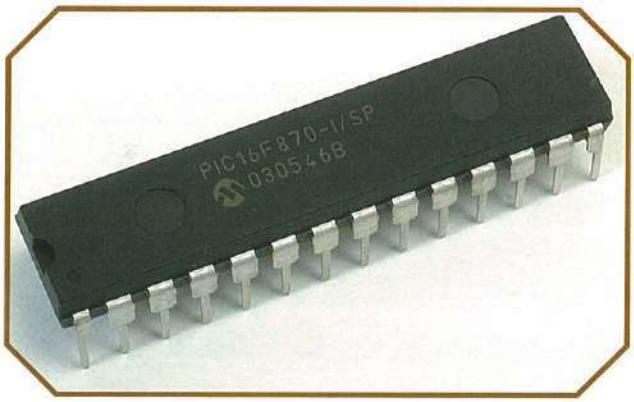
Nelle grandi serie di schede vengono solitamente utilizzate le tecniche di montaggio superficiale. Il contenitore LPCC è il più adatto a questa tecnologia, anche se viene montato comunemente anche su zoccoli. Sono costruiti in materiale plastico e la distanza tra i terminali è di 1,27 mm. La numerazione dei terminali è diversa dalla precedente. Il punto di inizio si trova in uno dei lati del contenitore, quello con l'angolo più smussato, e il senso è quello antiorario. La sua versione in ceramica prende il nome di LCCC.

La nostra scelta

La prima cosa da fare è analizzare l'applicazione, che in questo caso è molto particolare. Si tratta di imparare a gestire un microcontroller e conoscere la sua programmazione. Per questo, avremo bisogno di un microcontroller di uso comune nei diversi campi o settori di applicazione, del costruttore con il maggior numero di vendite, che disponga di documentazione e strumenti di sviluppo, che sia potente e ci permetta di lavorare con una grande varietà di dispositivi. La scelta che soddisfa questi requisiti è un PIC di Arizona Microchip, più precisamente il modello PIC 16F870.



Contenitore LPCC.



PIC16F870.

Programmazione del PIC

Il microcontroller è un dispositivo molto potente capace di realizzare svariate funzioni, però per qualsiasi tipo di operazione deve essere programmato, quindi è necessario il lavoro di un progettista che realizzi, oltre all'hardware o al circuito di utilizzo, il programma. Queste due fasi sono inseparabili.

Il microcontroller è un dispositivo hardware che necessita di elementi software per il suo funzionamento. Nel nostro CD troviamo i programmi necessari per poter gestire il nostro chip a tutti i livelli, però prima di affrontare il contenuto del CD e per conoscere ciò che dobbiamo fare, vediamo una breve introduzione generale al software che normalmente si associa a un'applicazione con microcontroller.

CD del laboratorio di Elettronica Digitale.

Ambiente di sviluppo

L'espressione "ambiente di sviluppo" comprende l'insieme di strumenti necessari per progettare, programmare e provare l'applicazione.

Linguaggio assembler

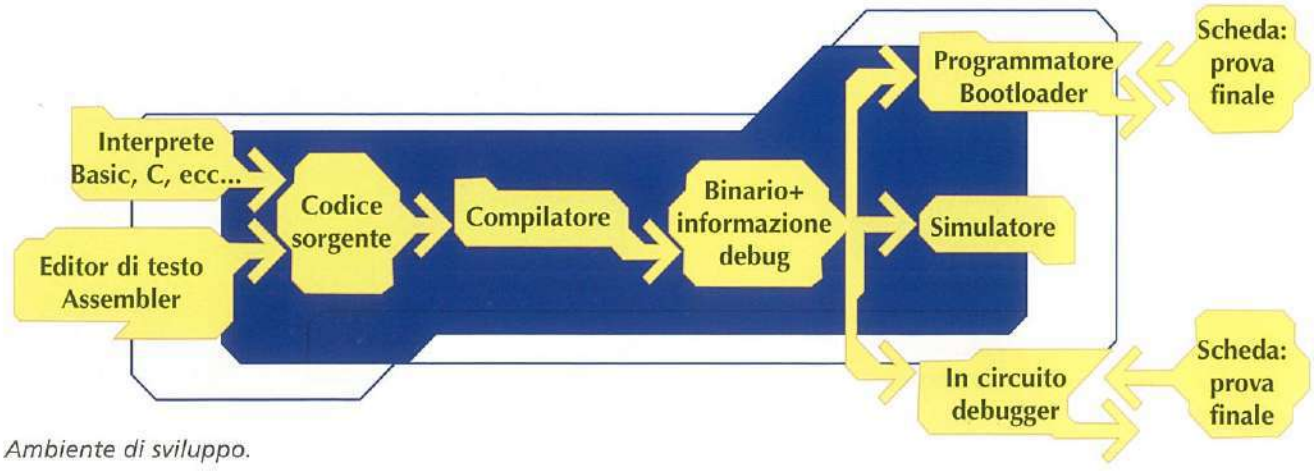
Il linguaggio macchina è la rappresentazione del programma così come lo intende il microcontroller. L'assembler è il linguaggio più vicino alla macchina, si tratta della sua rappresentazione alfanumerica in cui a ogni istruzione corrisponde un'istruzione in codice macchina (senza tener conto delle macro e delle direttive).

In assembler, il programmatore genera il miglior codice possibile, adattandosi al microcontroller. Solo mediante la programmazione in assembler è possibile capire bene l'architettura e la struttura del chip.

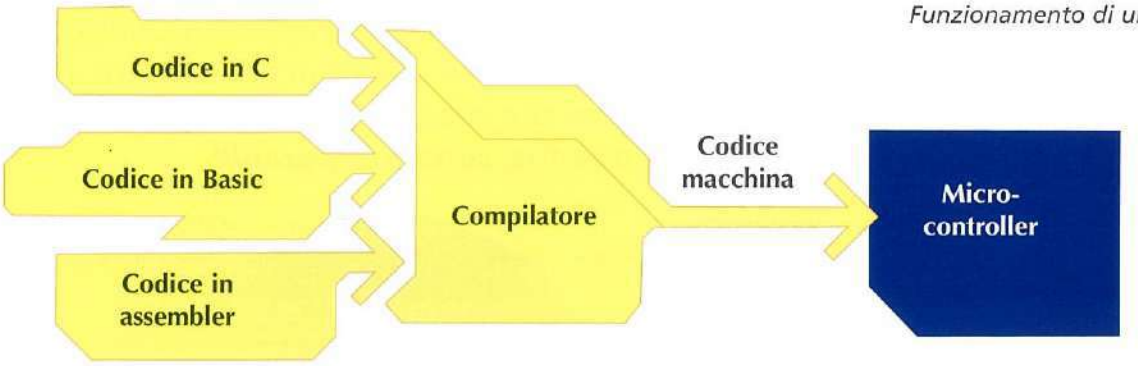
Al momento di affrontare un progetto con molte linee di codice, il lavoro del programmatore si fa più complicato, dal punto di vista sia della manutenzione che dell'aggiornamento del software. In questi casi è consigliabile utilizzare un interprete o compilatore di linguaggio ad alto livello.

Interpreti

Un interprete è un linguaggio traduttore ad alto livello (vicino al linguaggio naturale) a codice macchina. Risiede sul microcontroller ed esegue il programma leggendo ogni istruzione in alto livello una a una, traducendole ed eseguendole (traduce ed esegue allo stesso tempo).



Ambiente di sviluppo.



Funzionamento di un compilatore.

L'interprete più diffuso è il Basic, ma ne esistono anche altri come, ad esempio, il Forth.

Compilatori

Il compilatore è il programma che traduce il codice sviluppato per l'applicazione in linguaggio macchina per la successiva scrittura sul microcontroller.

Anche se l'assembler è il linguaggio per eccellenza in questi dispositivi, attualmente si tende a utilizzare compilatori di linguaggio ad alto livello. Ancora non esiste un programma interprete che risiede sul chip, ma per ora, viene sviluppato il programma in un linguaggio vicino a quello naturale e il compilatore si incarica di tradurlo in linguaggio macchina.

I linguaggi più utilizzati sono il C e il Basic.

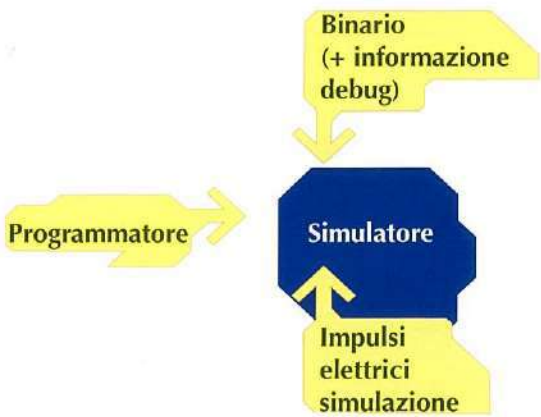
Scrittore

Abbiamo visto le diverse opzioni per lo sviluppo del programma e mediante un compilatore, lo possiamo tradurre in codice macchina, però manca ancora una cosa molto importante: portare questo codice al microcontroller.

Esistono due modi di scrivere il programma su un chip, mediante un software specifico di scrittura con l'hardware appropriato, oppure tramite un programma residente nel microcontroller, sempre che quest'ultimo lo permetta, cioè che supporti la scrittura del codice dell'applicazione sul proprio hardware di funzionamento. Questo programma è conosciuto come "boot loader".

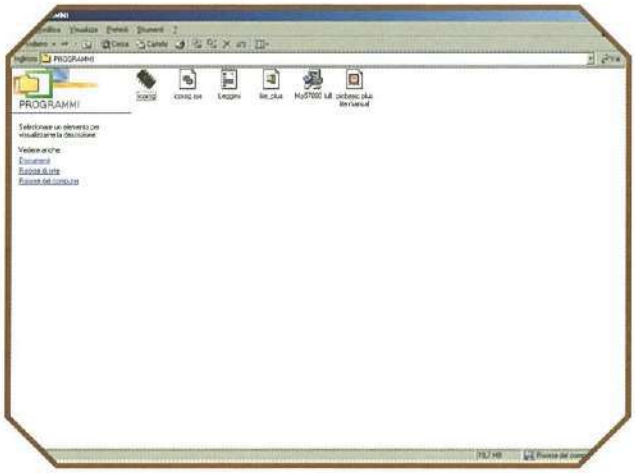
Simulatori

Un simulatore è un programma che esegue il codice del microcontroller in un computer. Permette di eseguire il codice passo a passo, in modo che si possa osservare ciò che avviene a ogni istruzione. È possibile vedere e verificare il contenuto dei registri, della memoria e delle variabili, oltre a verificare come risponde il programma. Questa caratteristica lo rende uno strumento molto utile, specialmente quando si sta imparando a programmare, perché permette di sperimentare con i programmi e studiare le conseguenze senza la necessità di scrivere il programma sul chip.



Possiamo simulare gli ingressi elettrici esterni.

Videata iniziale del CD del Laboratorio di Elettronica Digitale.



Contenuto della directory Programmi.

Strumenti di messa a punto o debuggers

Un debugger residente gira su un programma interno al microcontroller stesso, contemporaneamente mostra il progresso della messa a punto su un PC.

Ha le medesime caratteristiche di un simulatore, con l'ulteriore vantaggio che il programma gira su un microcontroller reale.

Logicamente questo strumento occupa risorse del microcontroller in quanto richiede memoria, comunicazioni e interrupt.

Emulatori

L'emulatore è una soluzione utilizzata raramente in quanto il suo prezzo e la possibilità di impiego ne rendono difficoltoso l'uso. Si tratta di un sofisticato dispositivo che sostituisce il microcontroller e contemporaneamente è in grado di acquisire informazioni.

Per utilizzare un emulatore dobbiamo togliere il microcontroller dalla scheda e mettere quest'ultimo al suo posto. Fornisce un'informazione totale su ciò che sta avvenendo nella realtà e non sottrae alcuna risorsa alla scheda che si sta analizzando.

Contenuto del CD

Quando si inserisce il CD nel lettore del computer apparirà una videata di navigazione in cui, oltre all'opzione di uscita dall'applicazione, è possibile scegliere fra tre opzioni: Programmi, Esercizi e applicazioni e PIC16F870.

Vediamo nel dettaglio il contenuto di queste directory.

Programmi

Nella directory "Programmi" sono compresi tutti i software che dobbiamo installare sul nostro PC per gestire il PIC. I programmi da installare sono:

ICPROG.

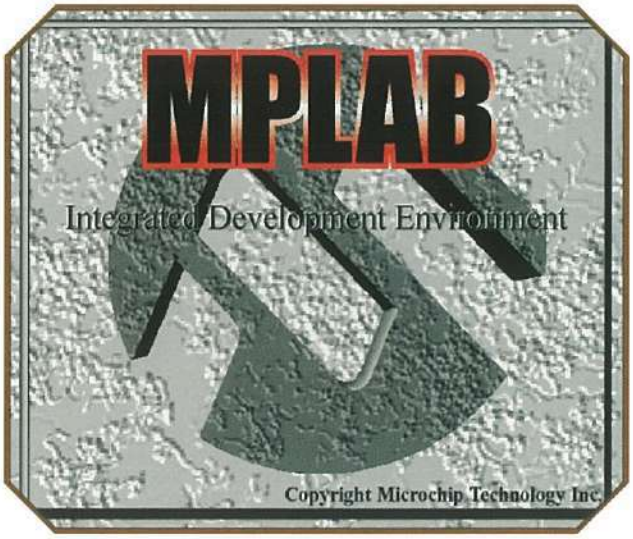
– ICPROG.EXE: Questo programma servirà per scrivere un PIC o le memorie.

Esso ci permette di selezionare il dispositivo che vogliamo scrivere mediante una delle porte di comunicazione del PC. Ha bisogno di un hardware di scrittura che vi verrà fornito.

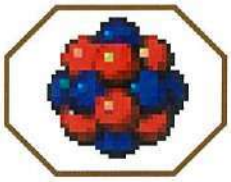
– Mp57000 full.exe: è il programma d'installazione di MPLAB, che è un ambiente di sviluppo per diversi PIC. Si tratta di un potente simulatore con cui lavoreremo molto allo scopo di capire meglio i programmi di applicazione.

Installando questa applicazione, oltre al simulatore MPLAB.EXE, avremo a disposizione il compilatore MPASMWIN.EXE, in grado di convertire il nostro codice da assembler a codice macchina, che verrà caricato sul PIC.

– Lite_plus.exe: è l'installatore del programma PICBASIC Plus che permette di sviluppare e compilare applicazioni utilizzando il Basic come linguaggio di programmazione con la famiglia di PIC 16F87X.



MPLAB.



PIC Basic Plus Lite.

Il resto del contenuto di questa directory è spiegato nel file "leggimi.txt" della directory principale.

Esercizi e applicazioni

In questa sezione sono compresi diversi esercizi in linguaggio assembler che dovranno essere compilati e convertiti per la loro scrittura e prova sul PIC. Questi esercizi ci serviranno per imparare a muoverci con il PIC 16F870 e familiarizzare con la sua programmazione e il suo funzionamento.

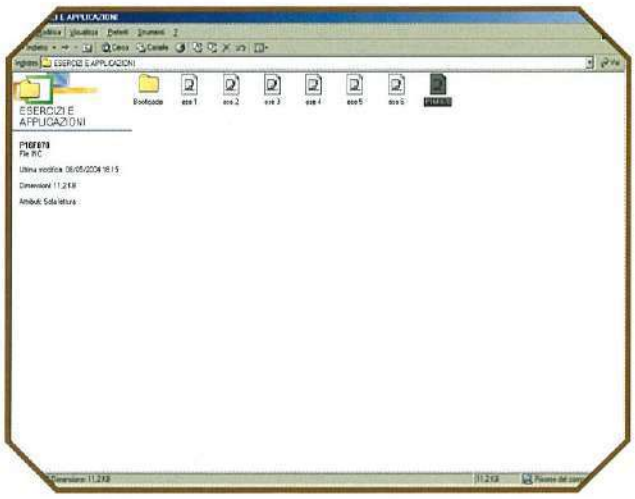
Anche se hanno l'estensione ".asm" si possono vedere con il programma "blocco note" o con qualsiasi altro editor di testo.

Troviamo anche un'applicazione per sfruttare i dispositivi del PIC come il Bootloader (vedi il file leggimi della directory Bootloader) e il file "P16F870.inc", una libreria dove vengono definiti tutti i registri del PIC che dovrà essere richiamata da qualsiasi programma che realizzeremo.

PIC 16F870

In questa directory si trova la documentazione tecnica fornita dal costruttore Microchip che fa riferimento al modello PIC 16F870. Spesso i progettisti hanno bisogno di consultare questi documenti per ricavare informazioni e chiarire dei concetti.

Come potrete verificare, Microchip fornisce molte informazioni e strumenti per gestire i suoi microcontroller. Questo va a beneficio dell'azienda e dei suoi utilizzatori.

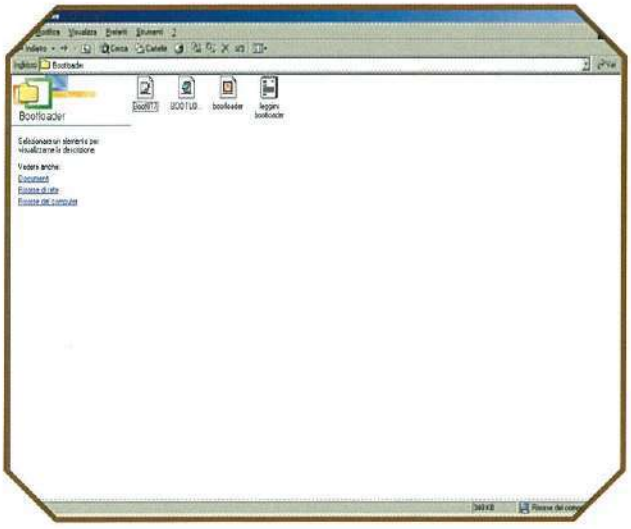


Contenuto della directory "Esercizi" e applicazioni.

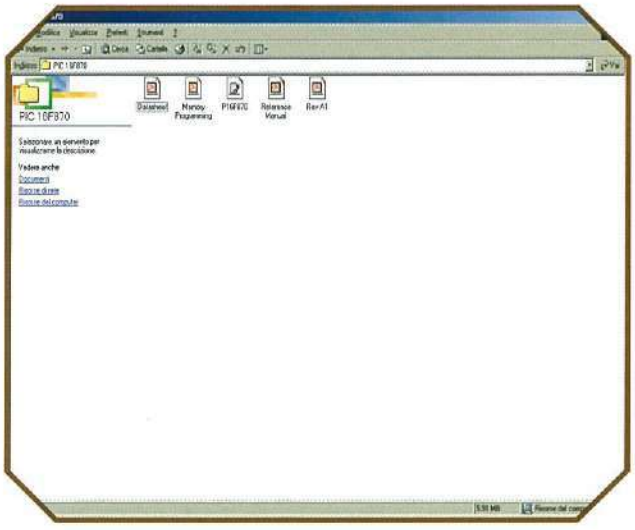
Troviamo in questa directory anche il file citato in precedenza: "P16F870.inc".

Nella directory principale del nostro CD, oltre ai file dell'applicazione di avvio, troveremo un file "leggimi.txt" che è indispensabile leggere se vogliamo installare correttamente i programmi e analizzare nel dettaglio il contenuto del CD.

Il lettore può esercitarsi attraverso l'installazione di questi programmi, entrando nei menù di ognuno di essi per vedere le opzioni possibili e fare pratica con alcuni esempi.



Contenuto della directory Bootloader.



Contenuto della directory PIC16F870.

Avvicinandosi al PIC 16F870

Fino a questo momento abbiamo visto i possibili microcontroller da utilizzare ma, a partire da questo momento, ci concentreremo su quello scelto e sulla sua programmazione. Analizzeremo le schede del nostro chip, per conoscere la funzione di ognuna delle sue parti, al fine di ottenere il massimo rendimento da questo dispositivo. In questo modo approfondiremo le conoscenze sul PIC16F870, acquisendo concetti applicabili a quasi tutti i controller.

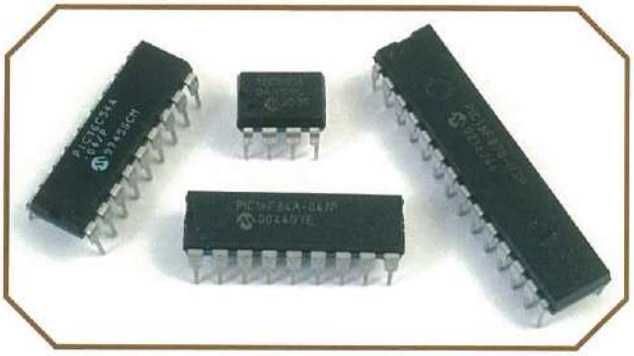
Le quattro gamme dei PIC

Microchip ha sul mercato più di 100 modelli di microcontroller PIC che si dividono in quattro categorie o gamme, in base alle prestazioni che forniscono. Esiste una compatibilità in discesa del software, cioè, un programma che si può eseguire su un PIC a bassa potenza si può eseguire anche su uno a potenza superiore. Questa caratteristica unita alla flessibilità hardware fornita da questo microcontroller, permette ad un progettista con delle conoscenze su un modello specifico di implementare un sistema su qualsiasi altro PIC.

La gamma media

La gamma media è, per così dire, il capostipite di Microchip, è la più popolare ed è quella che dispone di un maggior ventaglio di possibilità. Tra le caratteristiche principali ricordiamo la semplicità, il basso consumo e il prezzo dei microcontroller.

All'interno dell'ampia diversità di microcontroller esistenti nella gamma media, passiamo dai semplici PIC12C6XX a 8 pin, fino ai più potenti e sofisticati PIC16F87X. In ogni caso, tutti i modelli di questa gamma hanno un insieme di 35 istruzioni con una lunghezza di 14 bit ciascuno e tutti presentano la stessa architettura di base. Le differenze si trovano nel tipo e nelle capacità della memoria, oltre che nella disponibilità di dispositivi e periferiche.



Alcuni PIC della gamma media.

Gestione di interrupt, programmazione "on-board" , stack da 8 livelli, sono alcune delle caratteristiche più rilevanti che insieme al numero elevato di modelli della gamma media dispongono di memoria FLASH, la quale si può scrivere e cancellare elettricamente, fanno di questa gamma una delle principali scommesse del futuro di Microchip.

La famiglia PIC16F87X

Questa famiglia nacque in seguito al grande successo di mercato che ebbe il PIC16F84. Tutte le previsioni di Microchip furono superate e il PIC16F84 diventò lo standard dei

Gamma migliorata	PIC18CXXX	**Repertorio di 77 istruzioni da 16 bit**
Gamma alta	PIC17CXXX	**Repertorio di 58 istruzioni da 16 bit**
Gamma media	PIC12C6XX, PIC16CXX	**Repertorio di 35 istruzioni da 14 bit**
Gamma di base	PIC12CXXX, PIC16C5X	**Repertorio di 33 istruzioni da 12 bit**

Gamma dei PIC.

Prodotto	Memoria del programma		EEPROM memoria dei dati bytes	RAM bytes	Terminali di I/O	Contenitore	Analogico	Digitale			Velocità Max. MHz	CCP/ ECCP
	Bytes FLASH	Words FLASH					Canali AD	PWM 10-bit	Temporizzatori /WDT	I/O Serie		
PIC16F84A	1792	1024x14	64	68	13	18P, 18SO, 20SS	NO	NO	1-8 bit, 1 WDT	NO	20	NO
PIC16F870	3584	2048x14	64	128	22	28SP, 28SO, 28SS	5 (10-bit)	1	2-8 bit,1-16 bit, 1 WDT	AUSART	20	1
PIC16F871	3584	2048x14	64	128	33	40P, 44L, 44PT	8 (10-bit)	1	2-8 bit,1-16 bit, 1 WDT	AUSART	20	1
PIC16F872	3584	2048x14	64	128	22	28SP, 28SO, 28SS	5 (10-bit)	1	2-8 bit,1-16 bit, 1 WDT	MI_C/SPI	20	1
PIC16F873	7168	4096x14	128	192	22	28SP, 28SO 28SS	5 (10-bit)	2	2-8 bit,1-16 bit, 1 WDT	AUSART/ MI_C/SPI	20	2
PIC16F874	7168	4096x14	128	192	33	40P, 44L, 44PQ, 44PT	8 (10-bit)	2	2-8 bit,1-16 bit, 1 WDT	AUSART/ MI_C/SPI	20	2
PIC16F876	14336	8192x14	256	368	22	28SP, 28SO	5 (10-bit)	2	2-8 bit,1-16 bit, 1 WDT	AUSART/ MI_C/SPI	20	2
PIC16F877	14336	8192x14	256	368	33	40P, 44L, 44PQ, 44PT	8 (10-bit)	2	2-8 bit,1-16 bit, 1 WDT	AUSART/ MI_C/SPI	20	2

Tabella comparativa PIC16F84 e PIC16F87X.

microcontroller. Microchip era riuscita a portare sul mercato un buon prodotto: semplice, affidabile, piccolo, economico e con memoria FLASH; ma era sempre possibile migliorarlo! Questo microcontroller aveva una memoria dei dati piuttosto scarsa e una memoria di programma da 1 K, solamente 13 linee di I/O e mancava di alcuni dispositivi facilmente compensabili mediante hardware.

Mantenendo la linea del PIC16F84 nacque la famiglia PIC16F87X, dotata di maggiori dispositivi, con una memoria FLASH potenziata e più capace.

Nella tabella comparativa si possono vedere le poche differenze che ci sono tra i microcontroller presentati, quindi, conoscendone uno risulta facile anche lavorare con gli altri.

Il PIC16F870

È arrivato il momento di analizzare il nostro microcontroller, di conoscerlo, di sapere cosa si nasconde in questo piccolo chip e cosa si può arrivare a fare con esso.

Di seguito sono riportate le caratteristiche di questo potente microcontroller e la sua architettura interna.

Caratteristiche generali di tutti i PIC

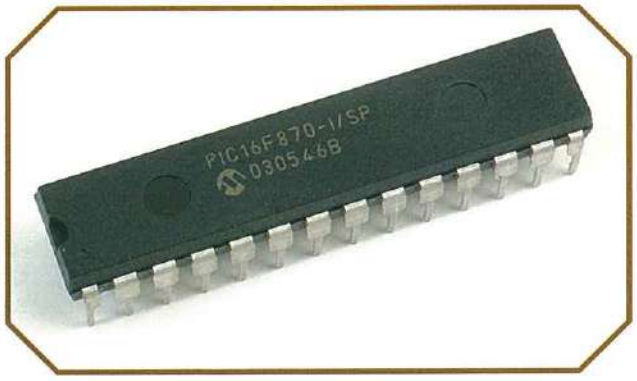
Il PIC16F870, analogamente al resto dei PIC, risponde a una architettura Harvard in cui la CPU si collega, in modo indipendente e con bus diversi, con la memoria delle istruzioni e con la memoria dei dati.

Viene applicata la tecnica della segmentazione nell'esecuzione delle istruzioni, quindi il processore può realizzare, nello stesso tempo, un'istruzione e la ricerca del codice di quella successiva.

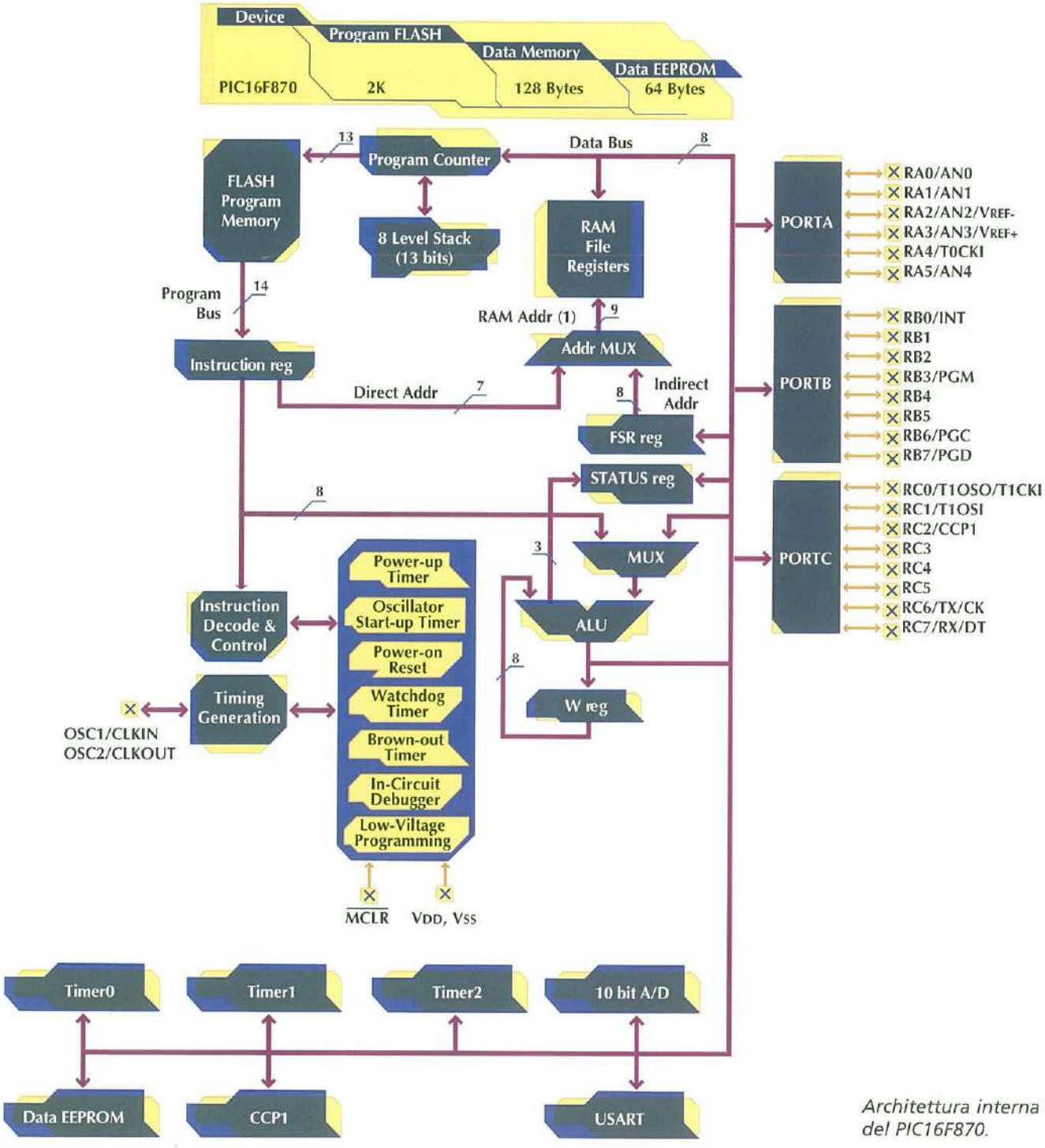
Il formato di tutte le istruzioni ha la stessa lunghezza (14 bit nella gamma media, 12 bit nella gamma di base e 16 bit nelle altre gamme) e sono ortogonali, quindi possono utilizzare qualsiasi dispositivo come sorgente o come destinazione.

Processore RISC: computer con un insieme di istruzioni ridotto (35 nella gamma media).

Architettura basata su un banco di registri, per cui tutti i dispositivi del sistema (porte di I/O, temporizzatori, ecc.) sono implementati fisicamente come registri.



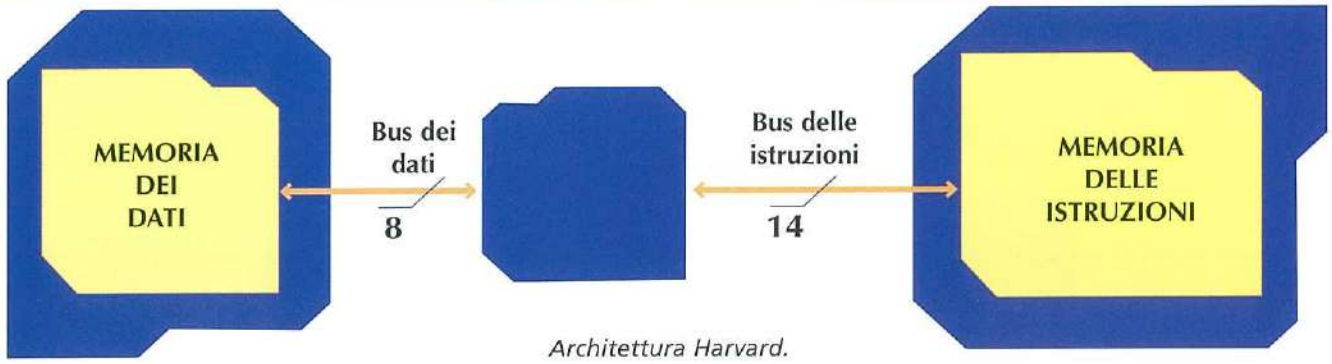
Il PIC16F870.



Architettura interna del PIC16F870.

Caratteristiche del PIC16F870

Nella tabella sono indicate le principali caratteristiche del microcontroller 16F870, però non sappiamo ancora come funziona, come possono interagire i diversi moduli per fare in modo che nel suo insieme, il chip possa dare la soluzione per i nostri progetti. È necessario uno studio della sua architettura interna, vedere e capire ciò che vi è all'interno del chip; a questo scopo, Microchip fornisce lo schema della sua architettura interna.



Architettura Harvard.

Architettura interna del PIC16F870

La memoria di programma (FLASH) è indirizzata dal contatore di programma (PC) da 13 bit che contiene l'istruzione successiva che deve essere eseguita ed è legato ad uno stack a 8 livelli, dove viene salvato l'indirizzo di ritorno in caso di chiamata a subroutine o interrupt.

La memoria RAM dei dati può indirizzarsi in forma diretta o indiretta tramite un multiplexer. Un multiplexer è un dispositivo cui si collegano degli ingressi e che mediante un segnale di controllo sceglie di aprire la via verso un'uscita oppure un'altra.

L'indirizzamento è il modo di accedere a una determinata cella della memoria. Bisogna prendere l'informazione richiesta da una cella specifica all'interno della memoria, a questo scopo è necessario indicare il suo indirizzo.

Il decodificatore di istruzioni interpreta queste ultime e ha il compito di fare in modo che ALU o Unità Aritmetico Logica realizzi le operazioni corrispondenti. Tramite il registro di lavoro W si riceve un operando oppure un altro tramite un multiplexer, che discriminerà tra il bus dei dati o la propria istruzione. Il risultato dell'operazione passerà al registro di lavoro o al bus dei dati.

Tramite i terminali VDD e VSS si applica la tensione di alimentazione da +5 V e massa. Per eseguire un reset si imposta a livello basso il terminale MCLR. Sui terminali OSC1/CLKIN e OSC2/CLKOUT si monterà il quarzo che determina la frequenza di lavoro del microprocessore.

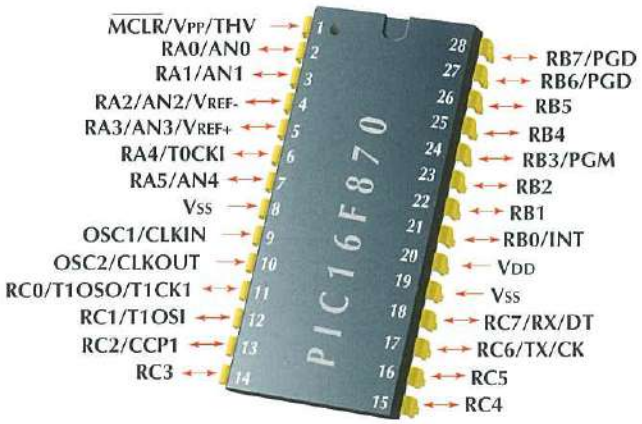
Il resto dei pin corrisponde a ingressi e uscite, anche se molte di esse hanno altre funzioni multiplexate.

Il chip si completa con i dispositivi speciali di cui dispone: temporizzatori, moduli di comunicazione, convertitori, ecc.

Dopo questa introduzione al PIC16F870 siamo già pronti per iniziare un'analisi più approfondita. Vogliamo che conosciate nel dettaglio il microcontroller, perché in questo modo vi risulterà molto più semplice capire la sua programmazione.

Caratteristiche	PIC16F870
Frequenza di funzionamento	DC – 20 MHz
Reset (e ritardi)	POR, BOR (PWRT, OST)
Memoria di programma FLASH (14-bits words)	2 K
Memoria dei dati (bytes)	128
Memoria dei dati EEPROM	64
Interrupt	10
Porte di I/O	Porte A, B, C
Temporizzatore	3
Modulo Capture/Compare/PWM	1
Comunicazione seriale	USART
Comunicazione parallela	---
Modulo Analogico/Digitale 10 –bit	5 canali di ingresso
Set di istruzioni	35 istruzioni

Caratteristiche del PIC16F870.



Piedinatura dei 28 pin del PIC16F870.

Collegamenti del PIC16F870

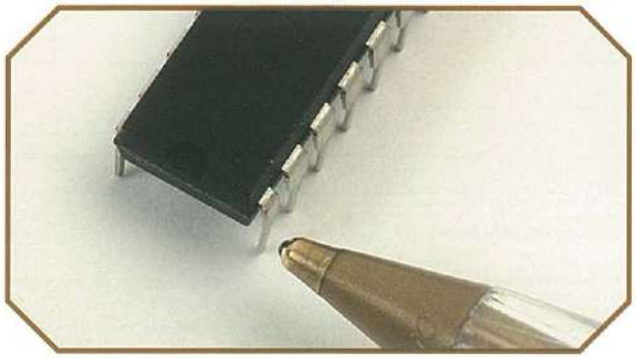
Il microcontroller interagisce con il mondo esterno tramite i suoi terminali di collegamento. Il PIC scelto, il 16F870, ha 28 linee di collegamento o terminali, e ognuno di essi ha una o più funzioni specifiche. Di seguito spiegheremo gli utilizzi di ognuno di essi e alcuni dei circuiti ausiliari che sono necessari per il corretto funzionamento del PIC.

I PIC, come la maggioranza dei circuiti integrati, seguono uno standard di numerazione. Se posizioniamo la tacca di riferimento del chip verso l'alto, il pin numero 1 sarà quello che si trova a sinistra della tacca. La numerazione aumenterà mano a mano che scenderemo lungo la fila di terminali che contiene anche il numero 1. Al termine della fila continueremo a numerare i terminali della fila del lato opposto dal basso verso l'alto, in modo che il terminale che si trova a destra della tacca di riferimento sia l'ultimo.

Dato che ci sono molti terminali, abbiamo scelto di realizzare una tabella riassuntiva che ci sarà di grande aiuto. Negli schemi si rappresenta il PIC con i terminali numerati e la denominazione abbreviata di ciascun terminale per facilitare la comprensione del circuito.

L'alimentazione

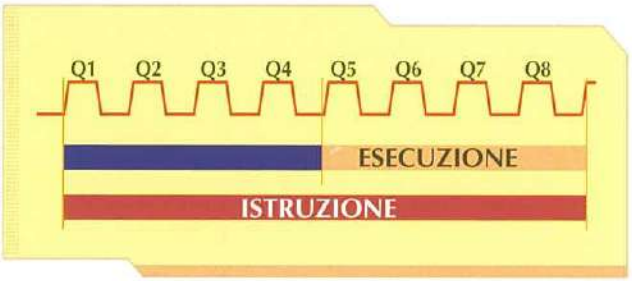
Qualsiasi dispositivo elettronico, per funzionare, necessita di un'alimentazione. Il 16F870 si può alimentare con una tensione continua compresa tra 2 e 5,5 V, anche se normalmente lo alimenteremo a 4,5 V (batterie) o 5 V, infatti, ad eccezione di qualche applicazione molto specifica, i progetti elettronici utilizzano normalmente questa tensione.



Il terminale 1 è quello più vicino alla tacca di orientamento.

Quando la tensione di alimentazione è di 5 V e sta funzionando a 4 MHz in condizioni normali, il consumo tipico è inferiore a 1,5 mA. Se riduciamo la frequenza di lavoro a 32 KHz e lo alimentiamo a 3 V il consumo tipico sarà di 20 µA e in stato di riposo (stand by) la corrente tipica è inferiore a 1 µA. Questi consumi così ridotti hanno dato a Microchip un vantaggio importante nei confronti di alcuni dei suoi concorrenti.

Per fare in modo che l'applicazione, ovvero il programma caricato sul microcontroller, funzioni, bisogna collegare i terminali 8 e 19 (Vss) al negativo dell'alimentazione (0 V), mentre il positivo dell'alimentazione è collegato al terminale 20 (Vdd).



Fasi di un'istruzione: ricerca ed esecuzione.

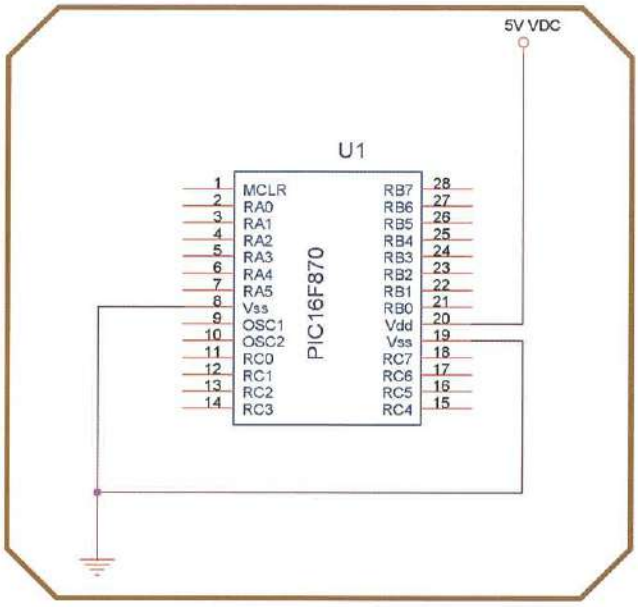
Il reset

Quando si produce un reset si reinizializza il processore. Il programma in esecuzione viene abbandonato e il contatore di programma (PC) è caricato con l'indirizzo 0000H, che corrisponde all'indirizzo riservato per il Vector di Reset nella memoria di codice. In questo indirizzo inizia il programma dell'applicazione. I due modi più comuni per provocare un reset sono:

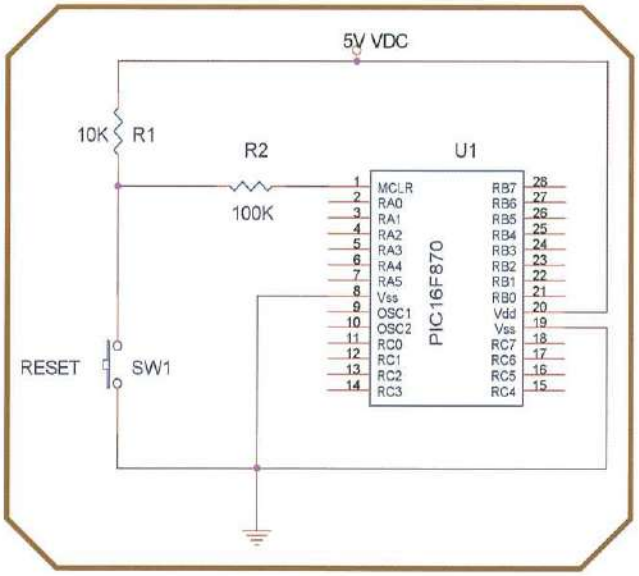
– Togliere e poi ristabilire l'alimentazione del microcontroller. Quando si collega l'alimentazione si produce automaticamente un reset. Questo tipo di reset è conosciuto come "POR" (Power-On Reset).

Denominazione terminale	Numerazione	Tipo I/O/P	Descrizione
OSC1/CLKI	9	I	Ingresso quarzo o clock esterno
OSC2/CLKO	10	O	Uscita quarzo. Nel modo RC esce da questo terminale 1/4 della frequenza che entra su OSC1
MCLR/VPP/THV	1	I/P	Ingresso di reset (il reset si produce con un livello basso di tensione) o ingresso della tensione di programmazione o ingresso di alta tensione per il modo test
RA0/AN0	2	I/O	Linea di I/O digitale o ingresso analogico 1
RA1/AN1	3	I/O	Linea di I/O digitale o ingresso analogico 2
RA2/AN2/VREF-	4	I/O	Linea di I/O digitale o ingresso analogico 3 o ingresso analogico di tensione negativa di riferimento
RA3/AN3/VREF+	5	I/O	Linea di I/O digitale o ingresso analogico 4 o ingresso analogico di tensione positiva di riferimento
RA4/T0CKI	6	I/O	Linea di I/O digitale o ingresso di clock del Timer 0
RA5/AN4	7	I/O	Linea di I/O digitale o ingresso analogico 5
RB0/INT	21	I/O	Linea di I/O digitale o ingresso di interrupt esterno
RB1	22	I/O	Linea di I/O digitale
RB2	23	I/O	Linea di I/O digitale
RB3/PGM	24	I/O	Linea di I/O digitale o ingresso di programmazione a bassa tensione
RB4	25	I/O	Linea di I/O digitale o ingresso di clock del Timer 0
RB5	26	I/O	Linea di I/O digitale
RB6/PGC	27	I/O	Linea di I/O digitale o ingresso del clock in programmazione seriale
RB7/PGD	28	I/O	Linea di I/O digitale o ingresso dei dati in programmazione seriale
RC0/T1OSO/T1CKI	11	I/O	Linea di I/O digitale o uscita dell'oscillatore del Timer 1 o ingresso del clock del Timer 1
RC1/T1OSI	12	I/O	Linea di I/O digitale o ingresso dell'oscillatore del Timer 1
RC2/CCP1	13	I/O	Linea di I/O digitale o ingresso di capture 1 o uscita del comparatore 1 o uscita del PWM 1
RC3	14	I/O	Linea di I/O digitale
RC4	15	I/O	Linea di I/O digitale
RC5	16	I/O	Linea di I/O digitale
RC6/TX/CK	17	I/O	Linea di I/O digitale o trasmissione asincrona della USART o sincrona
RC7/RX/DT	18	I/O	Linea di I/O digitale o ricezione asincrona della USART o dati clock sincroni
Vss	8, 19	P	Riferimento a massa
Vdd	20	P	Alimentazione positiva

Descrizione dei terminali del microcontroller.



Alimentazione del PIC16F870.



Tipico circuito di reset.

–- Applicare un livello logico basso sul terminale MCLR (Master Clear Reset). Attraverso questo terminale potremo provocare dall'esterno un reset al momento desiderato, senza togliere l'alimentazione al circuito. Un modo classico per far arrivare il livello basso di tensione (0 logico) per provocare la reinizializzazione è il seguente: nel momento in cui si attiva il pulsante si applica uno 0 al terminale 1 provocando il reset.

La frequenza di lavoro

Per eseguire un'istruzione sono necessarie due fasi:

– Fase di ricerca. A partire dall'indirizzo indicato dal PC, il processore deve localizzare nella memoria di codice l'istruzione successiva da eseguire.

– Fase di esecuzione. Interpretazione ed esecuzione dell'istruzione dopo aver ricevuto il codice binario di quest'ultima da parte del processore.

Ognuna di queste fasi si realizza con quattro operazioni elementari e ogni operazione elementare dura un ciclo di clock. Lavorando a una frequenza di 4 MHz, ogni ciclo di clock dura 250 ns, quindi il ciclo di un'istruzione durerebbe:

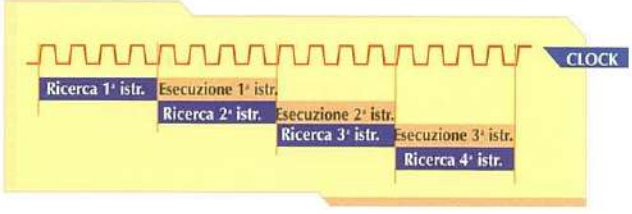
2 fasi = 8 cicli di clock = 8x250 ns = 2 µs

I PIC di Microchip utilizzano la segmentazione del processore per accelerare l'esecuzione delle istruzioni. Le due fasi vengono quindi eseguite in parallelo, in modo che, mentre si esegue un'istruzione, contemporaneamente si sta cercando quella successiva. In questo modo il ciclo di un'istruzione si realizza in quattro cicli di clock, quindi in 1 µs se lavoriamo a 4 MHz.

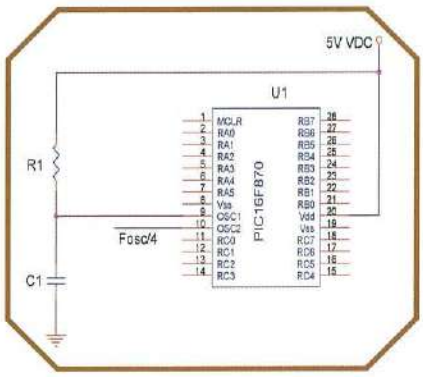
Nelle istruzioni di salto non si conosce l'istruzione successiva da eseguire fino a quando non si completa la sua esecuzione, quindi non è possibile applicare la segmentazione ed è necessario il doppio del tempo rispetto al resto delle istruzioni.

Tipi di oscillatore

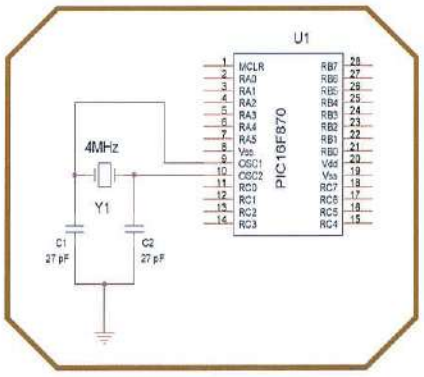
Il PIC16F870 accetta quattro modi di funzionamento dell'oscillatore. Quando scriviamo il programma dell'applicazione sul microcontroller dobbiamo configurare il modo di lavoro dell'oscillatore. Il software di scrittura ci permetterà di scegliere i seguenti modi:



Un'istruzione di salto richiede otto cicli di clock.



Oscillatore RC.



Oscillatore con quarzo.

Sul programma Icprog bisogna selezionare il tipo di oscillatore.

- RC: Resistenza-condensatore.
- LP: Quarzo per bassa frequenza.
- XT: Quarzo o risonatore per frequenze intermedie.
- HS: Quarzo o risonatore per alte frequenze.

Il PIC memorizzerà il modo di lavoro su due bit (FOSC1 e FOSC2) della Word di Configurazione, che è un registro specifico di configurazione del microcontroller.

Oscillatore RC

Nelle applicazioni in cui il tempo non è un fattore critico, possiamo utilizzare una resistenza in serie con un condensatore per implementare un oscillatore. È molto semplice ed economico, però non fornisce una frequenza stabile.

La frequenza di questo tipo di oscillatore è in funzione dei valori della resistenza e del condensatore, benché anche altri fattori, come la tensione di alimentazione e la temperatura di funzionamento, influiscano sul risultato. I valori raccomandati vanno da 3 a 100 K per le resistenze e condensatori superiori a 20 pF. Sul terminale 10 (OSC2/CLKOUT) otterremo un segnale che è la quarta parte della frequenza dell'oscillatore che avremo collegato al terminale 9 (OSC1/CLKIN).

Tipo	Frequenza	C1 (pF)	C2 (pF)
LP	32 kHz	33	33
	200 kHz	15	15
XT	200 kHz	47-68	47-68
	1 MHz	15	15
	4 MHz	15	15
HS	4 MHz	15	15
	8 MHz	15-33	15-33
	20 MHz	15-33	15-33

Range di frequenze degli oscillatori e dei condensatori di disaccoppiamento consigliati.

Oscillatori LP, XT e HS

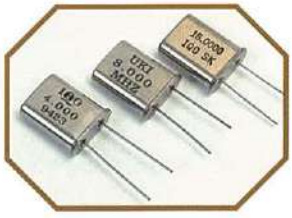
La frequenza di lavoro in questi modi di funzionamento si ottiene da un quarzo o da un risonatore ceramico montato fra i terminali 9 e 10.

Con il quarzo e i condensatori di disaccoppiamento, che sono indispensabili, la frequenza ottenuta è molto stabile. Questo metodo è utilizzato nella maggioranza delle applicazioni.

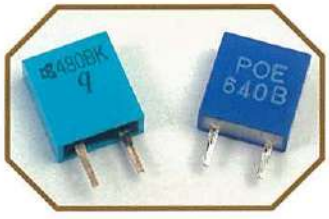
Conclusioni

Abbiamo visto come collegare il microcontroller e l'utilizzo di ognuno dei suoi terminali. Più avanti entreremo nel dettaglio per quanto riguarda il lavoro delle porte di ingresso e di uscita, però ora dobbiamo approfondire l'interno del PIC, conoscere la memoria e saperla gestire; in questo modo, quando arriveremo alla fase pratica, sarà più semplice capire il funzionamento dei circuiti reali proposti.

Quarzi.



Risonatori ceramici.



La memoria del PIC16F870

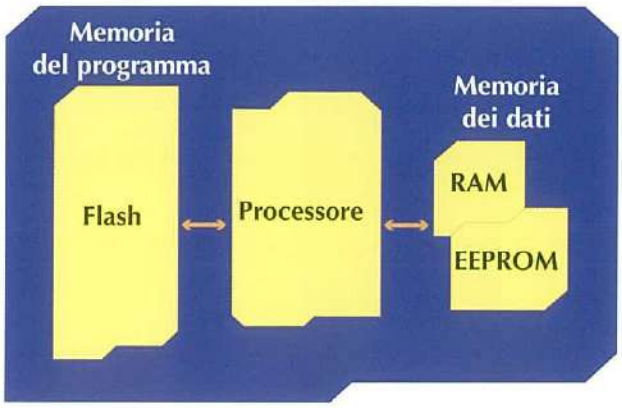
Il progettista deve conoscere e gestire la memoria del suo microcontroller. La programmazione dipende dalla memoria e dalla sua organizzazione. Per la maggioranza delle persone che iniziano a utilizzare un linguaggio di programmazione l'utilizzo corretto della memoria è la cosa più gravosa, forse perché è apparentemente poco interessante. In questo capitolo parleremo della memoria di cui è dotato il PIC16F870, della sua gestione e dei suoi registri specifici.

Tre tipi di memoria

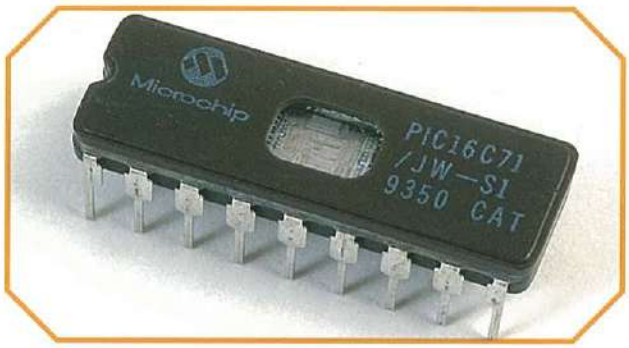
L'architettura Harvard permette alla memoria di programma e alla memoria dei dati di essere separate, dato che dispongono di bus indipendenti, è possibile accedere contemporaneamente a entrambe. Queste memorie, comunque, hanno funzionalità molto diverse e per questo la loro tecnologia di fabbricazione è differente.

Memoria di programma

La memoria di programma contiene le istruzioni dell'applicazione. Per un utilizzo didattico o per applicazioni in cui sia richiesta una certa flessibilità, il PIC deve permettere la sua scrittura quando necessario, in quanto i compiti a cui viene dedicato possono variare. Dopo aver scritto il programma dobbiamo leggere sulla memoria il codice per poterlo eseguire. La memoria deve accettare lettura, cancellazione e scrittura e deve mantenere le informazioni registrate anche quando si scollega l'alimentazione, in altre parole non deve essere volatile.



Architettura Harvard e i tre tipi di memoria.



Il PIC 16C71 ha una finestra per cancellare la sua memoria EPROM.

Esistono due tecnologie di fabbricazione che soddisfano questi requisiti. Le memorie EPROM possono essere cancellate sottoponendo il chip a raggi ultravioletti, dispongono infatti di una finestra di cancellazione. Questo è un inconveniente perché bisogna disporre dell'hardware necessario per questa operazione e il chip deve essere tolto dal circuito. L'altro tipo di memoria è la memoria FLASH. Ultimamente le memorie FLASH stanno sostituendo e praticamente facendo sparire, perlomeno nei nuovi progetti, le EPROM in quanto si possono cancellare elettricamente tramite alcuni pin. Non è necessario estrarre il chip dalla scheda sulla quale è montato per realizzare qualsiasi operazione sulla memoria, sempre che questa possibilità sia stata prevista.

PIC16F870		
	DIMENSIONE	TIPO
MEMORIA DEL PROGRAMA	2K x 14 words	FLASH
MEMORIA DEI DATI	128 x 8 bytes	RAM
	64 x 8 bytes	EEPROM

Memorie del PIC16F870.

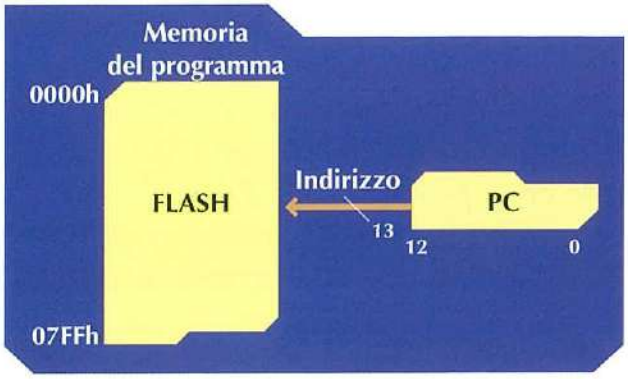
Memoria dei dati

La memoria dei dati deve disporre di celle che possano essere lette o scritte in qualsiasi momento. I dati gestiti da un programma possono essere modificati durante l'esecuzione dello stesso. I PIC16F87X utilizzano memoria RAM. Questa memoria è molto veloce, condizione necessaria per accedere ai dati, ed è volatile, ovvero i dati si perdono togliendo l'alimentazione.

Dato che esistono applicazioni in cui alcuni dati devono essere permanenti, ad esempio una chiave di sicurezza o valori limite di temperatura nel controllo di un forno, si aggiunge una memoria dei dati di tipo EEPROM. Questa memoria ha un funzionamento abbastanza simile alla FLASH, ma con una capacità assai più ridotta e un tempo di accesso molto più lento.

Organizzazione della memoria di programma

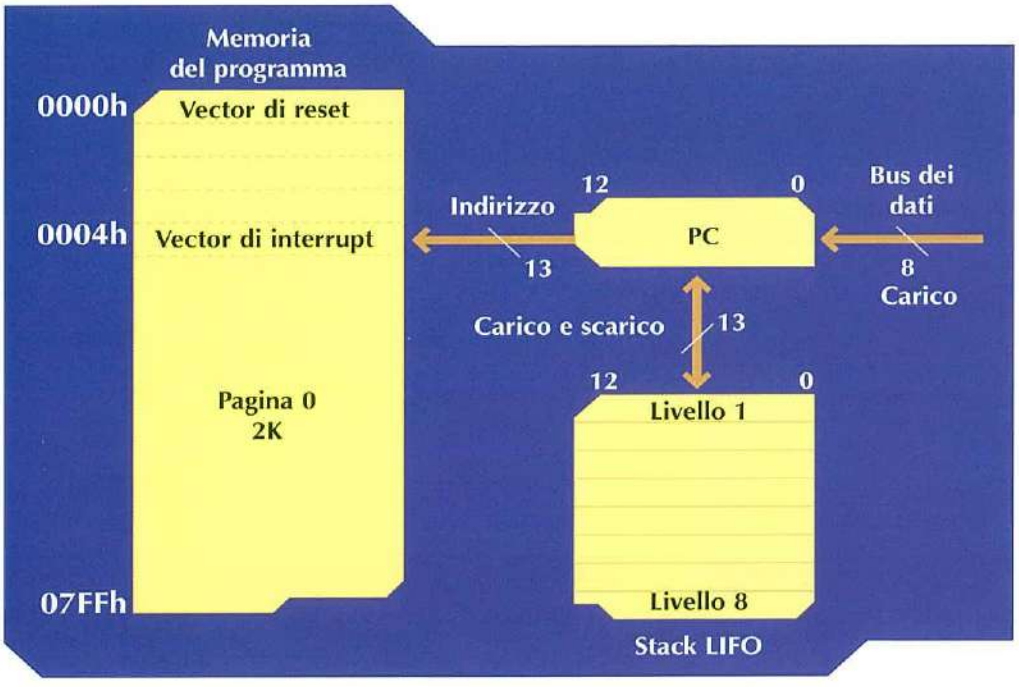
Le istruzioni dei PIC16F87X hanno una lunghezza di 14 bit, quindi le celle della memoria di programma devono avere la stessa lunghezza. Il PIC16F870 ha 2.048 celle di questa memoria (2 K), quindi per accedere a uno degli indirizzi abbiamo bisogno di un bus da 11 bit (211=2 K). Dato che il modello superiore della famiglia PIC16F87X ha una capacità da 8 K, questo implica un bus da 13 bit, Microchip utilizza per tutta la gamma dei PIC di questa famiglia un bus da 13 bit, pertanto nel nostro microcontroller sono utilizzati solo gli 11 bit meno significativi.



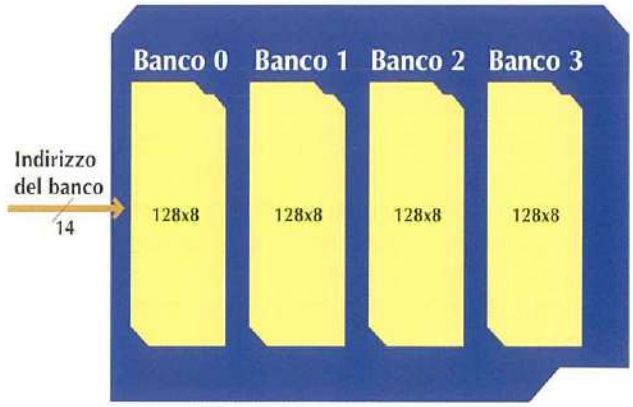
L'indirizzo del PC punta all'istruzione successiva da eseguire.

L'indirizzo a cui punta il microprocessore lungo il corso del programma corrisponde all'istruzione successiva a quella in esecuzione. Questo indirizzo si trova sul registro Contatore di Programma (PC) ed è generato automaticamente.

La memoria FLASH è divisa in pagine di



Funzionamento del Contatore di Programma. Interno della figura:



Banco della memoria RAM.

uguale dimensione. Nei PIC16F87X le pagine sono da 2 K. Pertanto il PIC con cui stiamo lavorando avrà solamente una pagina, mentre i PIC16F876/7 gestiranno quattro pagine.

La pagina 0, comune in tutti i modelli, ha due posizioni che sono riservate:

– Vector di reset (Indirizzo 0000h): quando si collega l'alimentazione o si resetta il processore, la prima istruzione che viene eseguita è quella che si trova a questo indirizzo. Tutti i nostri programmi inizieranno nel vector di reset.

– Vector di interrupt (Indirizzo 0004h): quando si genera un interrupt il Contatore di Programma punta sempre a questo indirizzo, quindi l'inizio del programma che fa riferimento agli interrupt dovrà essere posizionato a questo indirizzo.

Il Contatore di Programma

Questo registro incrementa di una unità il suo valore ogni ciclo di istruzione, fatta eccezione per le istruzioni di salto o per gli interrupt, in questi ultimi casi può essere caricato con qualsiasi altro valore e il valore che arriva prima del salto deve essere memorizzato per essere richiamato in seguito.

A questo scopo utilizza uno stack LIFO (Last Input First Output) da 8 livelli, in modo da scrivere l'ultimo indirizzo prima del salto sul livello 1 dello stack, e poterlo richiamare nel momento del bisogno. Lo stack si carica e scarica tramite il livello 1, quindi quando si carica un indirizzo su questo livello, il contenuto di ogni livello passa al successivo. Quando si scarica il livello 1 sul PC, il contenuto di ogni livello passa al precedente.

Le istruzioni di salto e l'istruzione CALL scrivono il contenuto del PC sullo stack e le istruzioni RETURN, RETFIE e RETLW caricano il contenuto del livello 1 dello stack sul PC.

L'analisi della memoria dei dati è fondamentale nella simulazione. Osserviamo se i registri GPR e quelli SFR evolvono nel modo desiderato.

c:\progra~1\mplab\prova-1.asm

```
                List    p=16F870        ;Processore
                include "P16f870.INC"   ;Definizione dei registri interni

Numero          equ     0x20            ;Numero casuale
Delay_Cont      equ     0x21            ;Contatore di intervalli
Temporale       equ     0x23            ;Variabiale temporale

                org     0x00            ;Vector di Reset
                goto    Inzio
                org     0x05            ;Salva vector di interrupt

;-------------------------------------------------------------------------------
;Tabella: Routine che converte il codice binario presente sui 4 bit meno significativi
;del registro W nel loro equivalente a 7 segmenti. il codice a 7 segmenti viene scritto anche
;sul registro W

Tabella         addwf   PCL,F           ;Spostamento sulla tabella
                retlw   b'00111111'     ;Numero 0
                retlw   b'00000110'     ;Numero 1
                retlw   b'01011011'     ;Numero 2
                retlw   b'01001111'     ;Numero 3
                retlw   b'01100110'     ;Numero 4
                retlw   b'01101101'     ;Numero 5
                retlw   b'01111101'     ;Numero 6
                retlw   b'00000111'     ;Numero 7
                retlw   b'01111111'     ;Numero 8
                retlw   b'01100111'     ;Numero 9
```

Microcontroller	RAM Registri di utilizzo generale	EEPROM
PIC16F870/1/2	128x8	64x8
PIC16F873/4	192x8	128x8
PIC16F876/7	368x8	256x8

Capacità della memoria RAM ed EEPROM della famiglia del PIC16F87X.

La memoria dei dati

Abbiamo visto che la memoria dei dati è formata da una memoria RAM e da una memoria EPROM. Nella memoria RAM sono contenuti i dati di utilizzo generale che vengono utilizzati nel programma e i registri specifici i cui bit controllano il funzionamento del processore e dei dispositivi interni. Nella memoria EPROM (non volatile) si scrivono i dati il cui valore deve essere mantenuto anche quando si toglie l'alimentazione.

Organizzazione della memoria RAM

La memoria RAM si divide in registri da 8 bit. I Registri di Utilizzo Generale (GPR) conterranno i dati che si utilizzano nel programma e i Registri di Utilizzo Specifico (SFR) conterranno i bit di controllo del processore e dei registri complementari.

La zona RAM si configura in quattro banchi da 128 indirizzi o registri da 8 bit ciascuno. I registri specifici sono situati nelle prime posizioni di ogni banco, lasciando quelle successive ai registri di utilizzo generale. In alcuni modelli esistono indirizzi che non sono implementati fisicamente.

Conclusioni

Al momento di progettare un programma dobbiamo considerare la capacità che ha il microcontroller per contenere il codice (memoria FLASH), per gestire registri di utilizzo generale, perché su essi scriveremo i dati con cui dovremo lavorare nell'applicazione (memoria RAM) e la capacità relativa alla memorizzazione di dati che vogliamo conservare nel momento in cui togliamo l'alimentazione (memoria EEPROM).

Questi fattori permettono di ottimizzare l'utilizzo della memoria e rivestono una notevole importanza al momento di scegliere il microcontroller da utilizzare nell'applicazione, se non vogliamo utilizzare memorie esterne.

c:\progra~1\mplab\prova-1.asm

```
                List    p=16F870        ;Processore
                include "P16f870.INC"   ;Definizione dei registri i

Numero          equ     0x20            ;Numero casuale
Delay_Cont      equ     0x21            ;Contatore di intervalli
Temporale       equ     0x23            ;Variabile temporale

                org     0x00            ;Vector di Reset
                goto    Inzio
                org     0x05            ;Salva vector di interrupt

;-------------------------------------------------------------------
;Tabella: Routine che converte il codice binario presente sui 4 bit
;del registro W nel loro equivalente a 7 segmenti. il codice a 7 se
;sul registro W

Tabella         addwf   PCL,F           ;Spostamento sulla tabella
                retlw   b'00111111'     ;Numero 0
                retlw   b'00000110'     ;Numero 1
                retlw   b'01011011'     ;Numero 2
                retlw   b'01001111'     ;Numero 3
                retlw   b'01100110'     ;Numero 4
```

Durante la simulazione di un programma potremo osservare l'evoluzione dei registri della memoria di programma, della memoria dei dati EEPROM e della memoria dei dati RAM.

Memoria RAM

Sappiamo a cosa serve la memoria RAM e com'è organizzata, dobbiamo però imparare a muoverci all'interno di essa e conoscere quali registri contiene.

Selezione del banco della RAM

La memoria dei dati RAM si divide in quattro banchi della stessa dimensione però con differenti contenuti. Quando vogliamo accedere a un registro dobbiamo specificare su quale banco si trova e questa operazione possiamo realizzarla utilizzando i bit RP1 e RP0 del Registro di Stato (STATUS). Nella tabella successiva possiamo vedere come, in funzione dei valori che assegniamo a questi bit, sarà possibile accedere a un banco oppure a un altro.

Distribuzione dei registri nella memoria RAM del PIC16F870

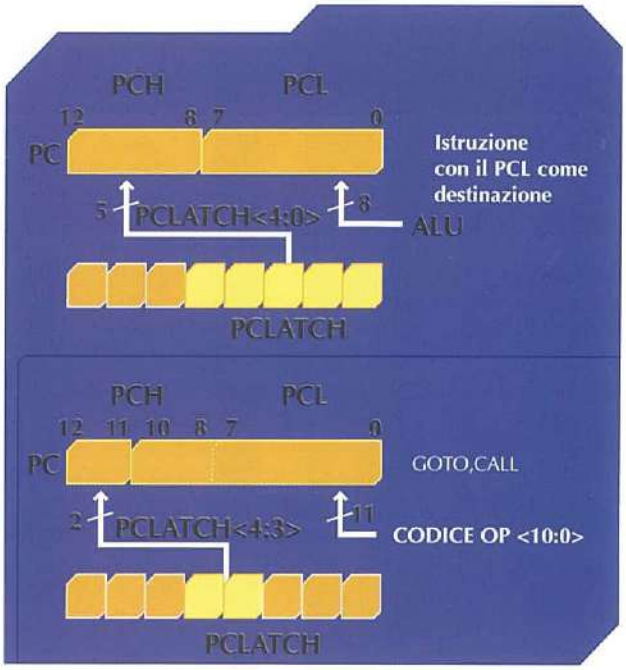
Anche se nei diversi modelli della famiglia 16F87X il numero dei registri di utilizzo generale cambia, i registri specifici sono chiamati con lo stesso nome e occupano le stesse posizioni della memoria. Questa particolarità assicura la compatibilità del software tra i diversi modelli.

Nella figura della pagina successiva è riportata la distribuzione dei registri nei diversi banchi del PIC16F870. Come possiamo vedere i registri specifici occupano le prime posizioni sui banchi e alcuni sono ripetuti su due banchi o su tutti e quattro. Questo serve a semplificare l'accesso ai registri di utilizzo comune, in modo da non dover cambiare banco ogni volta che vogliamo accedere a essi (STATUS, PCL, ecc).

Esistono degli indirizzi che non sono implementati fisicamente (hanno un altro colore nella figura) e sono letti sempre come zero.

	RP1	RP0
BANCO 0	0	0
BANCO 1	0	1
BANCO 2	1	0
BANCO 3	1	1

Selezione del banco della memoria con RP1 e RP0.



Possibili situazioni per il carico del PC.

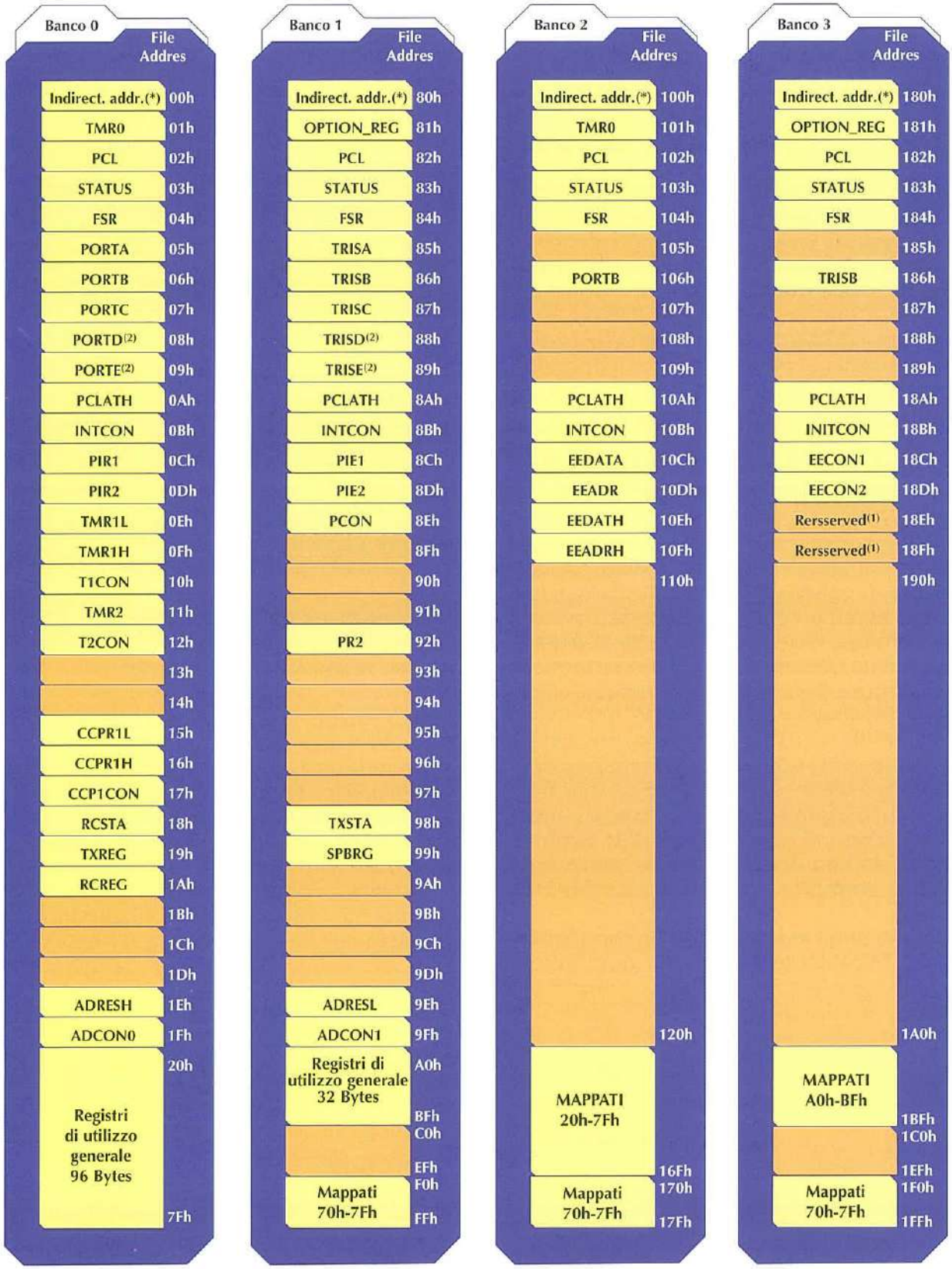
Altri registri, invece, sono mappati, ovvero quando si accede a essi in realtà si accede a un altro indirizzo, in pratica sono mascherati da altri indirizzi. I registri segnati con (2) non sono implementati nel nostro modello e i registri segnati con (1) sono riservati e devono essere mantenuti a zero.

Per quanto riguarda i registri di utilizzo generale del PIC16F870 ve ne sono 96 sul banco 0, che occupano gli indirizzi da 20h a 7Fh e 32 sul banco 1, che occupano gli indirizzi da A0h a BFh.

I registri specifici

I registri specifici regolano il comportamento del microprocessore e dei suoi dispositivi. Ogni registro è associato a una funzione generale e, all'interno di essa, ogni bit ha una funzione specifica.

Conosceremo le funzioni e il modo di gestire ognuno di essi e per fare questo dobbiamo iniziare analizzando i registri che servono per indirizzare le memorie, ovvero i registri che ci permetteranno di accedere a esse.



Distribuzione dei registri della memoria RAM nel PIC16F870.

Indirizzamento della memoria di programma

Vi sono alcuni indirizzi specifici per l'indirizzamento della memoria di programma. Per accedere alla memoria FLASH, in cui si scrivono le istruzioni, utilizziamo il registro contatore di programma (PC). Questo registro da 13 bit contiene l'indirizzo della cella di memoria dove si trova l'istruzione successiva da eseguire. Il valore di questo registro è contenuto in due registri specifici:

– PCL: contiene gli 8 bit meno significativi del PC <7:0> può essere letto e scritto.

– PCLATH: contiene i 5 bit rimanenti, quelli più significativi, del PC <12:8>. Non può essere letto, può essere scritto solamente mediante un indirizzamento indiretto. Questi bit sono impostati a zero quando si produce un RESET.

Esistono due modi per caricare il PC, che capiremo osservando le corrispondenti illustrazioni.

Nella prima situazione il PC è caricato scrivendo il PCL e passando i 5 bit corrispondenti del PCLATH alla sua destinazione, il PC.

Nel secondo caso possiamo vedere come si carica il PC quando si produce un'istruzione di salto CALL o GOTO. Dei 14 bit che formano l'istruzione precedente, gli 11 meno significativi si caricano sul PC. Con questi 11 bit è possibile eseguire un salto all'interno della pagina da 2 K posizioni in cui si sta eseguendo il programma.

Se è necessario un cambio di pagina il progettista dovrà caricare i bit <4:3> del PCLATH adeguatamente, per fare in modo che si produca il salto dopo aver passato questi ultimi ai due bit più significativi del PC.

Indirizzamento della RAM

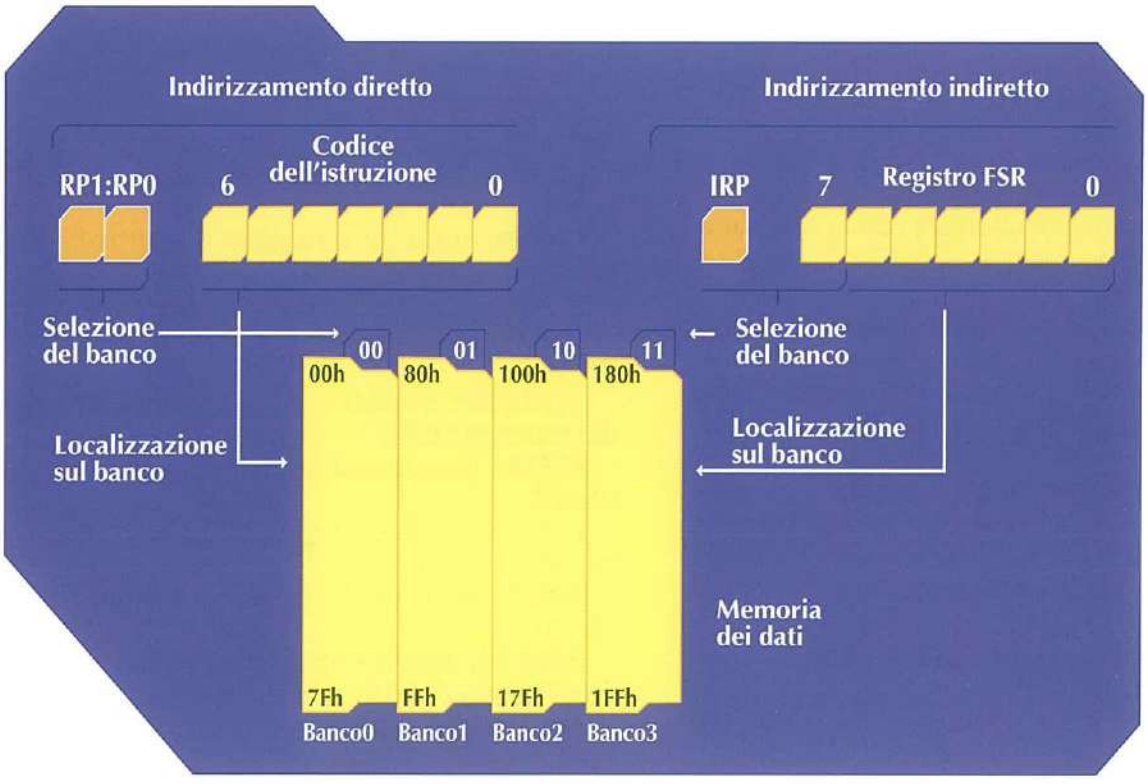
Vediamo ora quali sono i registri specifici che si utilizzano per l'indirizzamento della memoria RAM.

Per accedere a un indirizzo della memoria dei dati dobbiamo specificare il banco a cui vogliamo accedere e l'indirizzo all'interno di esso. Per la selezione del banco abbiamo bisogno di 2 bit (22 = 4 posizioni) e per l'indirizzo 7 bit (27 = 128 indirizzi su ogni banco).

È possibile accedere alla memoria RAM in due modi diversi:

– Indirizzamento diretto.

– Indirizzamento indiretto.



Indirizzamento diretto e indiretto per l'accesso alla memoria RAM.

Indirizzamento diretto

In questo caso i bit RP0 e RP1 del registro STATUS <6:5> indicano il banco e i 7 bit meno significativi del codice dell'istruzione che si sta eseguendo indicano l'indirizzo all'interno del banco.

Indirizzamento indiretto

Nelle istruzioni con l'indirizzamento indiretto si utilizza come operando il registro INDF, che occupa l'indirizzo 0 di tutti i banchi. Il registro INDF non è implementato fisicamente. Ogni volta che si fa riferimento a esso, si utilizza il contenuto del registro FSR per indirizzare l'operando. Il registro FSR è ubicato all'indirizzo 4 e come l'INDF, si trova implementato in tutti i banchi per facilitarne l'accesso. I 7 bit meno significativi dell'FSR indicano l'indirizzo all'interno del banco e il bit più significativo, insieme al bit IRP del registro di stato STATUS <7>, selezionano il banco a cui accedere.

Per capire meglio, osservando anche il disegno illustrato, proviamo ad accedere a una posizione della memoria nei due modi che vi abbiamo presentato.

Il programmatore deve caricare i valori sul registro STATUS e su FSR. L'indirizzo 21H in esadecimale corrisponde al valore binario 0010 0001b.

Banco 0, riassunto

Vediamo ora una breve descrizione dei registri del banco 0 da tener presente in caso di dubbi al momento di lavorare con la memoria di questo banco.

TMR0: contiene il valore del temporizzatore TMR0.

STATUS: registro di Stato, contiene flag di eventi speciali quando si eseguono le istruzioni.

PORTA, PORTB E PORTC: contengono i valori di ingresso e di uscita a cui si accede tramite le Porte A, B e C.

INTCON: questo registro controlla gli interrupt.

PIR1 e PIR2: questi registri gestiscono il controllo dei flag degli interrupt.

TMR1L e TMR1H: registrano il valore che utilizza il temporizzatore TMR1.

Address	Symbol	Value	Binary
02	PCL	H'31'	b'00110001'
0A	PCLATH	H'00'	b'00000000'

Esempio del contenuto dei registri specifici dell'indirizzamento della memoria di programma durante l'esecuzione dello stesso.

Address	Symbol	Value	Binary
04	FSR	H'00'	b'00000000'
00	INDF	H'00'	b'00000000'
03	STATUS	H'18'	b'00011000'

Esempio del contenuto dei registri che intervengono nell'indirizzamento della memoria dei dati durante l'esecuzione di un programma.

Indirizzo a cui accedere: 21H del banco 0		
	Indirizzamento diretto	Indirizzamento indiretto
Selezione del banco	RP1=0, RP0=0	IRP (STATUS<7>)=0, FSR <7>=0
Posizionamento all'interno del banco	Codice istruzione <6:0> =010 0001	FSR<6:0>=010 0001

Esempio di indirizzamenti.

T1CON: controlla il funzionamento del TMR1.

TMR2: questo registro contiene il valore del TMR2.

T2CON: questo è il registro di controllo del TMR2.

CCPR1L-CCPR1H: questi registri contengono il valore del modulo di capture, compare e PWM.

CCP1CON: questo è il registro di controllo del modulo CCP1.

RCSTA: registro di controllo e stato della USART.

TXREG: questo è il registro per la trasmissione della USART.

RCREG: registro di ricezione dei dati della USART.

ADRESH: questo registro contiene il byte più significativo del risultato del convertitore AD.

ADCON0: è il registro di controllo del convertitore AD.

Registri di controllo

La conoscenza dettagliata del funzionamento dei registri di controllo è fondamentale per un buon progettista. I bit di questi registri gestiscono le funzioni fondamentali e i dispositivi del processore.
Anche se non siamo pronti per analizzare la maggior parte dei registri, dato che non abbiamo ancora approfondito i dispositivi del PIC, possiamo però presentare due dei registri di controllo: STATUS, con cui dovremo familiarizzare perché ne avremo bisogno in qualsiasi programma, e PCON che – anche se non è molto importante – è ugualmente indispensabile conoscere.

Il registro di stato, STATUS

Il registro di stato occupa la quarta posizione nei quattro banchi della memoria. Al suo interno contiene lo stato delle operazioni aritmetiche della ALU, lo stato del RESET e i bit per la selezione del banco della memoria dei dati.

Se il bit è indicato come R/W (Read/Write) significa che può essere letto e scritto. Se ha solamente la R significa che è di sola lettura.

Il valore che prende il bit dopo un reset per collegamento dell'alimentazione (POR, Power On Reset) è indicato mediante '-n'.

A titolo di esempio analizzeremo alcuni dei bit di questo registro:

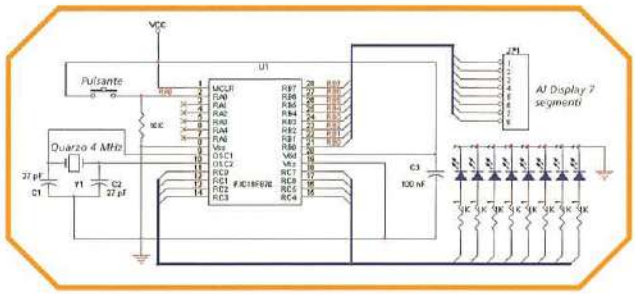
– IRP (R/W-0): il bit può essere letto o scritto e dopo un reset per accensione prenderà il valore 0.

– C (R/W-x): il bit può essere letto o scritto e dopo un reset per accensione prenderà un valore casuale.

Quando si produce una rotazione (RRF o RLF) il bit di carry (C) si carica con il bit meno significativo o con quello più significativo, secondo il verso della rotazione.

Il registro PCON

Il registro PCON si trova all'indirizzo 8Eh sul banco 1 della memoria dei dati. Contiene due bit con i quali si può differenziare se il reset è stato originato da un buco di tensione (BOR, Brown-Out) o per collegamento della stessa (POR).



Schema elettronico di un'applicazione.

Conclusioni

Abbiamo presentato due dei registri di controllo della memoria dei dati, il registro STATUS, di utilizzo comune e di grande importanza, e il registro PCON, poco utilizzato ma che dobbiamo conoscere. Il resto dei registri lo vedremo quando studieremo i dispositivi forniti dal nostro PIC16F870, lavoro che inizieremo a partire da ora.

Non è necessario memorizzare i bit che compongono i registri, infatti, quando programmiamo, potremo utilizzare questa pubblicazione come guida, dove troveremo tutti i registri spiegati bit a bit. Tuttavia man mano

STATUS

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

Struttura del registro di stato, STATUS.

Bit 7	IRP	Bit per la selezione del banco dei registri (utilizzato nell'indirizzamento indiretto)
	1	Banchi 2, 3 (100h – 1FFh)
	0	Banchi 0, 1 (00h – FFh)
Bit 6-5	RP1:RP0	Bit per la selezione del banco dei registri (utilizzato negli indirizzamenti diretti)
	11	Banco 3 (180h – 1FFh)
	10	Banco 2 (100h – 17Fh)
	01	Banco 1 (80h – FFh)
	00	Banco 0 (00h – 7Fh)
Bit 4	$\overline{TO}$	Bit di fine del tempo (Time Out)
	1	Dopo l'accensione o grazie alle istruzioni CLRWDT o SLEEP
	0	Quando termina il tempo del watchdog
Bit 3	$\overline{PD}$	Bit di spegnimento (Power Down)
	1	Dopo l'accensione o grazie all'istruzione CLRWDT
	0	Per l'esecuzione di un'istruzione SLEEP
Bit 2	Z	Bit di zero
	1	Il risultato di un'operazione aritmetica o logica è zero
	0	Il risultato di un'operazione aritmetica o logica non è zero
Bit 1	DC	Bit di Carry/Borrow sul primo digit (istruzioni ADDWF, ADDLW, SUBLW, SUBWF)
	1	Se c'è riporto sul risultato dell'operazione sui 4 bit meno significativi (digit)
	0	Se non c'è riporto sul risultato dell'operazione sui 4 bit meno significativi (digit)
Bit 0	C	Bit di Carry/Borrow (istruzioni ADDWF, ADDLW, SUBLW, SUBWF)
	1	Se c'è riporto sul risultato dell'operazione sul bit più significativo
	0	Se non c'è riporto sul risultato dell'operazione sul bit più significativo

Definizione dei bit del registro STATUS.

che avanzeremo nella preparazione dei programmi, vi renderete conto di quanto sia semplice gestire questi registri di configurazione, potendoli programmare senza la necessità di utilizzare alcun documento di appoggio.

Progetto di una applicazione

Dopo aver visto come si lavora con i registri di controllo, arriva il momento di pensare al progetto di una applicazione. Lo dobbiamo fare in modo ordinato e dobbiamo seguire sempre la stessa procedura, applicando una metodologia che ci faciliti lo sviluppo finale. Indipendentemente dal linguaggio di programmazione e dalla grandezza del progetto, il progettista deve seguire delle norme che ne facilitino l'esecuzione e la comprensione da parte dell'utente finale.

Definizione del problema

Il primo passo per realizzare un progetto è aver chiaro ciò che vogliamo fare. Il sistema da sviluppare dovrà soddisfare una serie di requisiti e di specifiche.

Quindi la prima cosa che dovrà fare un progettista quando affronta un nuovo progetto sarà l'analisi del problema.

Schema elettronico

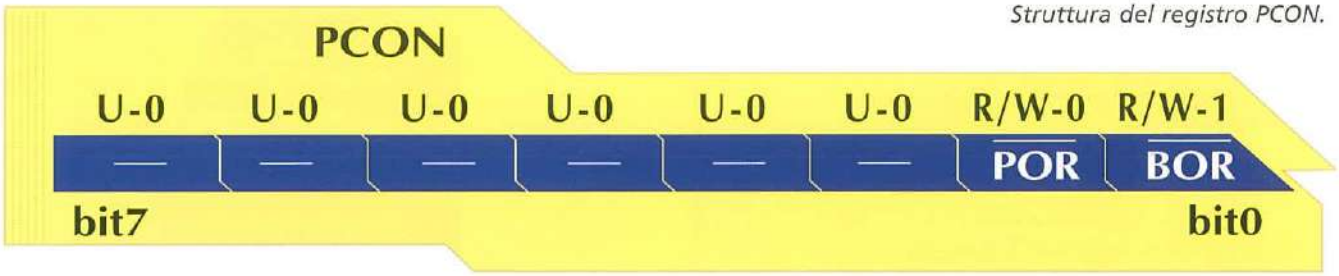
Abbiamo enunciato i punti principali che caratterizzano il nostro progetto, sappiamo ciò che vogliamo fare, ora però dobbiamo chiederci come farlo.

In molti sistemi sarà necessario implementare un circuito elettronico che soddisfi le necessità del nostro progetto.

Definizione del problema:
- Voglio visualizzare in sequenza i numeri da 0 a 9.
- Voglio che quando lo desidero appaia visualizzato un numero casuale.
- Voglio che questa visualizzazione duri un tempo preciso dopodiché si ripeta la visualizzazione sequenziale.

Dobbiamo avere ben chiaro ciò che vogliamo fare.

Struttura del registro PCON.



Nel nostro caso abbiamo bisogno di visualizzare un numero, a questo scopo, useremo un display a 7 segmenti e/o dei diodi LED per la rappresentazione binaria. Avremo bisogno anche di un pulsante per determinare il momento in cui desideriamo che si visualizzi il numero casuale e ovviamente il PIC, con il suo oscillatore e la sua alimentazione.

Tutte queste idee si plasmano in un circuito elettronico in cui il PIC è la parte centrale dove saranno collegati tutti gli elementi.

Gli schemi elettronici possono essere più o meno complessi secondo l'applicazione e possono essere fatti a mano oppure utilizzando un software specifico (ORCAD, Electronics Workbench, ecc.).

Lo schema elettronico ci permetterà di definire una soluzione al problema iniziale e determinare l'hardware necessario per lo sviluppo, oltre alla configurazione che dovremo programmare sul microcontroller.

Organigrammi

Un organigramma è la rappresentazione grafica di ciò che deve fare il programma. Dobbiamo creare una traccia che ci permetta di seguire un certo ordine durante la programmazione, e questo lo otterremo creando precedentemente un organigramma.

Un organigramma si rappresenta mediante:

– Rettangoli, che contengono le azioni da eseguire.

– Rombi, dove si specificano le condizioni che determinano il percorso da seguire.

– Frecce, che indicano l'ordine seguito dalle istruzioni. Se fuoriescono da un rombo devono essere etichettate per scegliere l'alternativa correttamente.

Gli organigrammi avranno un unico punto di ingresso e un unico punto di uscita.

Normalmente per un'applicazione complessa si creano diversi organigrammi. Esisterà sempre l'organigramma generale dell'applicazione, però conviene realizzare organigrammi dettagliati di ogni parte o modulo che compone l'applicazione.

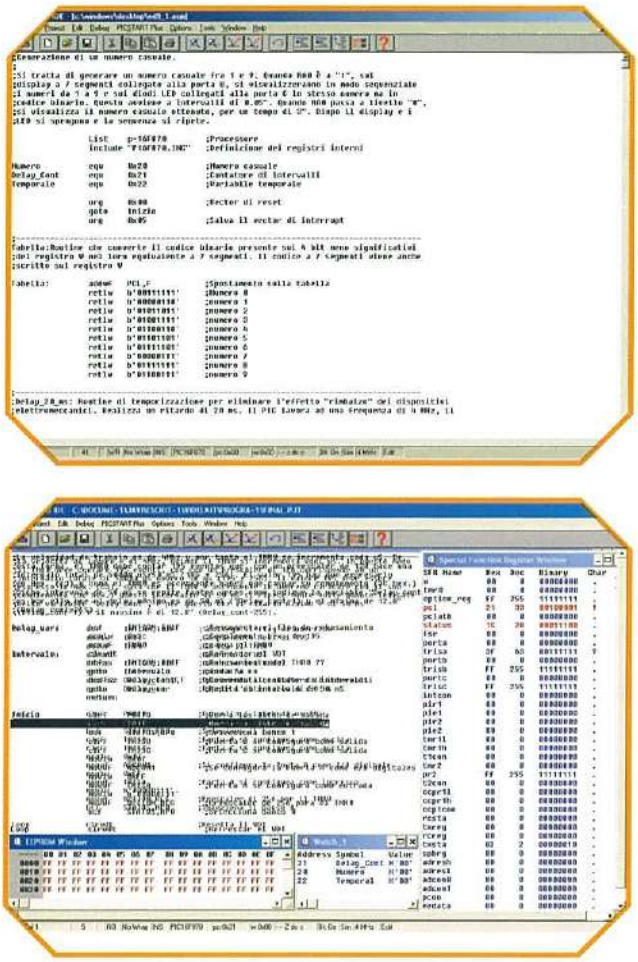
Come potrete verificare, ciò che in principio era solamente un'idea sta già prendendo forma. Si è concretizzata, è stata definita e possiamo già iniziare a programmare.

Programma

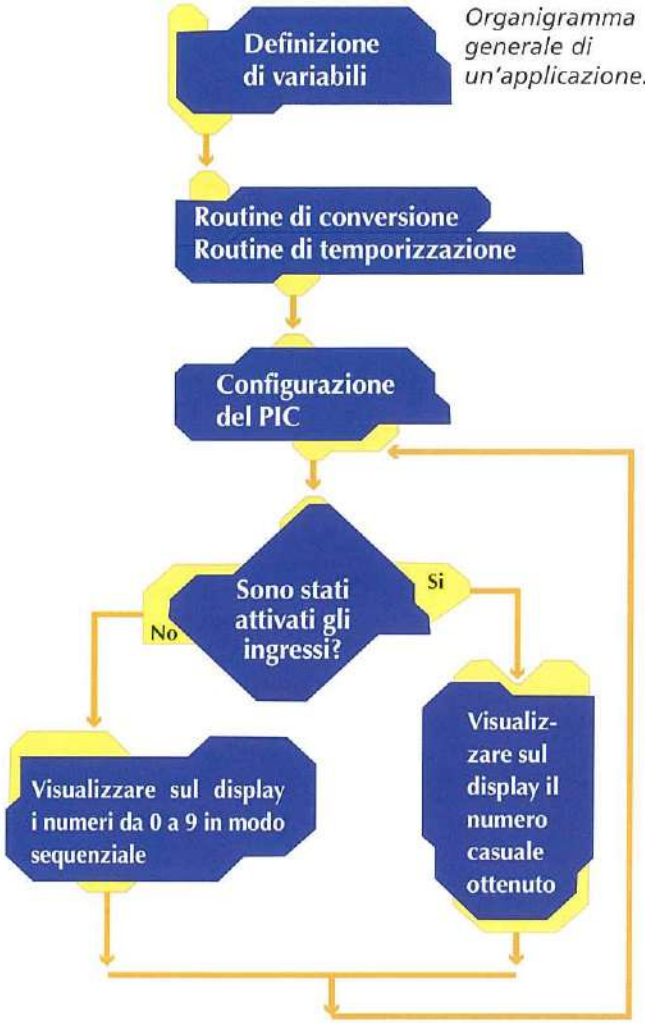
Quando affrontiamo la programmazione conviene tenere lo schema elettronico e gli orga-

Bit 7.2		**Non implementati (Si leggono come 0)**
Bit 1	**POR**	**Flag di "Power On Reset"**
	1	Non c'è stato un reset per collegamento dell'alimentazione
	0	C'è stato un reset per collegamento dell'alimentazione (deve essere reimpostato a 1 via software)
Bit 0	**BOR**	**Flag di "Brown-out Reset"**
	1	Non si è verificato un abbassamento di tensione
	0	Si è verificato un abbassamento di tensione (dovrà essere reimpostato a 1 via software)

Definizioni dei bit del registro PCON.



Possiamo verificare il corretto funzionamento del programma mediante un simulatore.



Organigramma generale di un'applicazione.

nigrammi a portata di mano. La programmazione non è altro che la scrittura delle nostre idee in un linguaggio comprensibile alla macchina. Se partiamo da organigrammi dettagliati e ben fatti non sarà molto difficile tradurli in istruzioni. Con lo schema elettronico potremo configurare il microcontroller, dato che conosceremo tutti i suoi requisiti hardware.

È molto importante commentare il programma mediante spiegazioni di ciò che fanno le istruzioni. Così il programma potrà essere capito e messo a punto dopo la sua realizzazione.

Messa a punto

Dopo aver terminato il programma, dobbiamo compilarlo prima di eseguirlo. È molto difficile, anche per un esperto programmatore, che un programma funzioni bene al primo colpo. Nella fase di compilazione si potranno correggere gli errori di sintassi, prodotti per assenza di parametri, parole scritte male, ecc. Il programma non verrà compilato fino a quando non saranno stati corretti gli errori.

Fatto questo, occorrerà verificare che il programma realizzato esegua realmente ciò che vogliamo. Prima di montare il sistema fisico finale conviene analizzare il funzionamento del programma su un simulatore (MPLAB), potremo così valutare il funzionamento di ogni istruzione e del programma nel suo insieme.

Infine scriveremo il programma sul microcontroller e monteremo il circuito, pronti a far fronte agli eventuali errori che ci potrebbero essere durante lo sviluppo hardware del sistema.

Le porte di ingresso e uscita

Le porte di ingresso e uscita (I/O) sono i dispositivi utilizzati dal microcontroller per interagire con il mondo esterno. In qualsiasi applicazione il microcontroller scambierà dati con l'esterno, sia per ottenere informazioni che per agire su elementi esterni.

Elementi esterni, periferiche

Le applicazioni con il PIC dipendono dai dati che il processore acquisisce dall'esterno e normalmente lo scopo delle applicazioni è di controllare o governare i dispositivi esterni collegati al sistema.

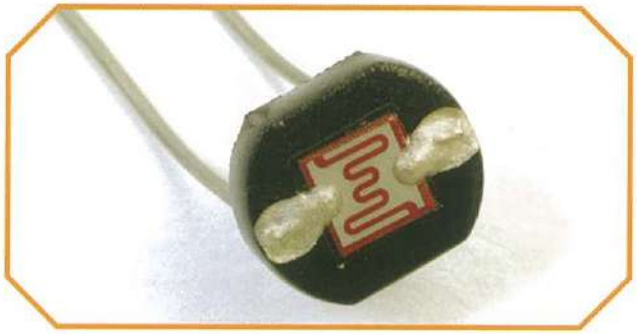
Di solito i dispositivi che si collegano alle porte di I/O sono digitali e lavoreranno quindi con due soli stati (1 e 0), modo lavoro tipico di dispositivi quali interruttori, pulsanti, tastiere, display LCD, display a LED, relé, ecc. Questi dispositivi digitali forniranno o riceveranno informazioni dal microprocessore in formato digitale o binario.

Esistono tuttavia molti dispositivi periferici analogici e il PIC deve interagire anche con essi. Per questa ragione alcuni microcontroller hanno delle linee di I/O predisposte per lavorare con queste periferiche e dispongono al loro interno di uno o più convertitori A/D (Analogico/Digitale) per poter lavorare internamente con l'informazione.

Le porte di I/O nel PIC16F870

Il PIC16F870 dispone di tre porte di I/O (A, B e C). Dato che il processore è digitale e dispone di un bus dei dati da 8 linee su cui circoleranno in modo bidirezionale 8 bit, la dimensione

Dispositivi digitali.



Sensore di luminosità analogico.

logica di una porta di I/O è di 8 linee. Le porte B e C del nostro microcontroller hanno 8 linee di I/O digitali, invece la porta A ha una configurazione diversa e dispone solamente di 6 linee di I/O.

Come qualsiasi altro dispositivo elettronico, un microcontroller deve essere costruito in modo che la sua dimensione possa essere la più piccola possibile e per questa ragione esiste una limitazione di piedini o linee che rende necessario il multiplexaggio della maggior parte delle linee di I/O, in modo da poter realizzare altre azioni. In base alla programmazione dei registri di controllo, queste linee multifunzionali eseguiranno diversi lavori.

Quando funzionano come linee di I/O digitale si chiamano RAx, RBx e RCx.

I bit di ogni porta si configurano mediante i bit corrispondenti di un registro di controllo associato che prende il nome di TRIS (TRISA, TRISB o TRISC). Mediante i registri PORTA, PORTB e PORTC, indirizzi 5, 6 e 7 dell'area dei dati, trasferiremo i dati nel verso che è stato definito sul registro TRIS.

La porta A

PORTA è una porta bidirezionale da 6 bit. Il suo registro corrispondente per indicare il verso dei dati è TRISA. Impostando a 1 un bit del registro TRISA configureremo il terminale corrispondente di PORTA come ingresso (si imposta il dri-

Sensori di temperatura.

ver di uscita a uno stato di alta impedenza). Se impostiamo uno 0 su un bit del TRISA, il terminale corrispondente della porta sarà configurato come uscita (porta il contenuto del latch di uscita sul terminale corrispondente).

Leggendo il registro PORTA, in realtà, leggiamo lo stato dei pin, tuttavia quando scriviamo il registro PORTA ciò che stiamo facendo è scrivere sul latch della porta. Un'azione di scrittura implica tre operazioni: lettura, modifica e scrittura. Quindi quando vorremo impostare un valore sulla porta, ciò che realmente fa il processore è leggere lo stato dei pin, modificare il valore di quelli che lo richiedono e, successivamente, scrivere su ogni latch il bit corrispondente.

Il terminale RA4 è differente da tutti gli altri, è multiplexato con l'ingresso del segnale di clock per il funzionamento del temporizzatore TMR0. La configurazione interna del terminale è diversa, come si può vedere nella figura della pagina successiva.

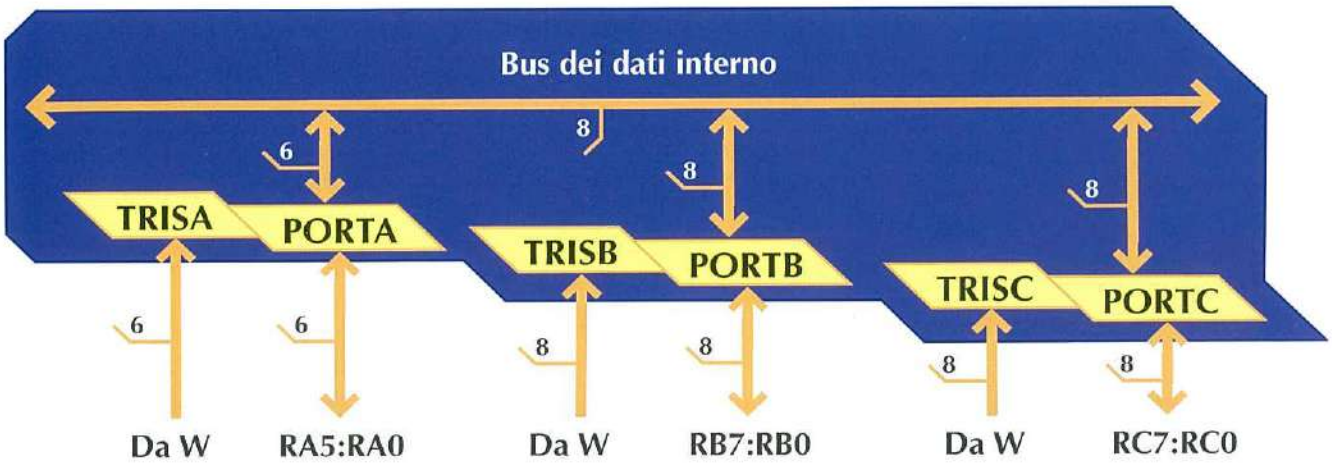
Porta A	Porta B	Porta C
Linee: 6	Linee: 8	Linee: 8
RA0/AN0	RB0/INT	RC0/T1OSO/T1CK1
RA1/AN1	RB1	RC1/T1OSI
RA2/AN2/Vref-	RB2	RC2/CCP1
RA3/AN3/Vref+	RB3/PGM	RC3
RA4/T0CK1	RB4	RC4
RA5/AN4	RB5	RC5
	RB6/PGC	RC6/TX/CK
	RB7/PGD	RC7/RX/DT

Nomenclatura delle linee di I/O del PIC16F870.

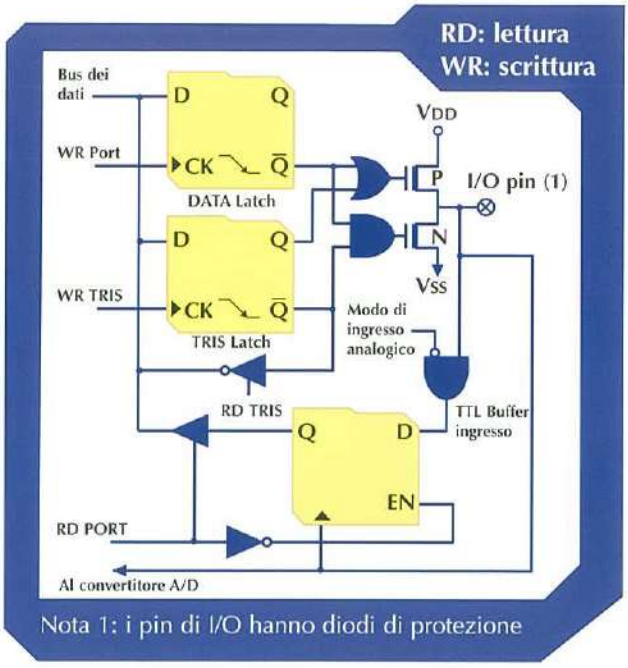
Il resto dei piedini della porta A ha multiplexate le funzioni di I/O digitale con quella del canale di ingresso analogico per il convertitore A/D interno. Inoltre RA2 e RA3 servono anche per ricevere la tensione di riferimento negativa (RA2) e positiva (RA3) per i dispositivi che ne hanno bisogno. Quando si genera un reset per il collegamento dell'alimentazione le linee della porta A rimangono configurate come linee di ingresso analogico.

Registri associati alla porta A

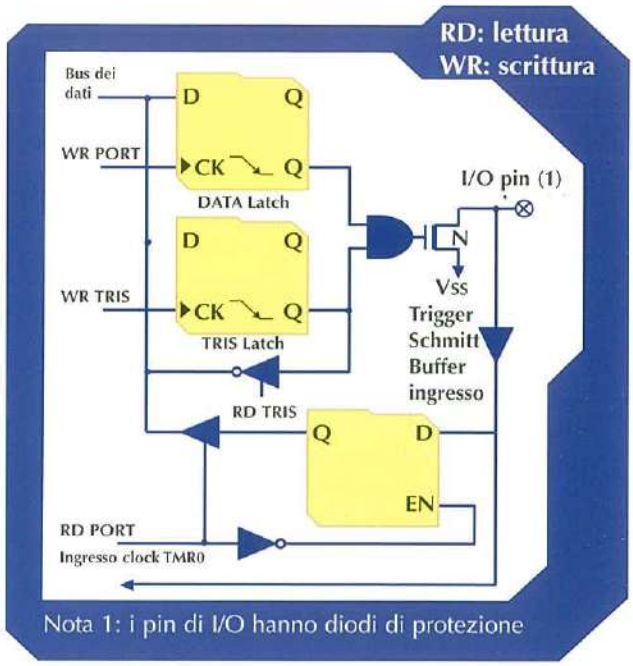
Abbiamo visto che mediante il registro PORTA si trasferiscono i dati e che mediante TRISA ne indichiamo il verso, ovvero se i pin sono di in-



Registri associati a ogni porta di I/O.



Schema a blocchi dei terminali RA3, RA0 e RA5.



Schema a blocchi del terminale RA4.

gresso o di uscita. Però sappiamo anche che tutti questi pin hanno multiplexate le loro funzioni. Per scegliere il modo lavoro delle linee della porta A bisogna programmare il registro di controllo del convertitore chiamato ADCON1.

ADCON1 si trova all'indirizzo 9Fh del banco 1 della memoria dei dati. Degli 8 bit che formano il registro, tre non sono utilizzati e il resto, il bit ADFM (bit 7 più significativo) serve per selezionare il formato del risultato della conversione, e PCFGx sono i bit di configurazione dei pin della porta A.

Quando utilizzeremo il convertitore A/D ripasseremo nuovamente il registro ADCON1 e capiremo meglio la funzionalità del bit ADFM. Per quanto riguarda i bit di configurazione, in funzione del valore che ha ognuno di essi avremo una configurazione di pin diversa, come potremo vedere nella tabella della pagina successiva.

In questa tabella A significa ingresso analogico e D ingresso/uscita digitale. Inoltre gli indirizzi segnati con (1) non sono implementati sul nostro PIC16F870. La colonna segnata con (2) indica il numero di canali analogici disponibili come ingressi analogici e il numero di questi canali utilizzato come ingressi di tensioni di riferimento.

La flessibilità quando si utilizza lo stesso programma su diversi modelli di microcontroller PIC, fa sì che molti registri siano predisposti per contenere o controllare più dispositivi di quelli che offre il nostro modello.

Nome	Bit #	Buffer	Funzione
RA0/AN0	bit 0	TTL	Ingresso/Uscita digitale o ingresso analogico
RA1/AN1	bit 1	TTL	Ingresso/Uscita digitale o ingresso analogico
RA2/AN2/Vref -	bit 2	TTL	Ingresso/Uscita digitale, ingresso analogico o Vref
RA3/AN3/Vref +	bit 3	TTL	Ingresso/Uscita digitale, ingresso analogico o Vref
RA4/T0CK1	bit 4	ST	Ingresso/Uscita digitale o ingresso clock TMR0
RA5/AN5	bit 5	TTL	Ingresso/Uscita digitale o ingresso analogico

Modi di funzionamento dei pin della porta A.

ADFM: Bit di selezione del formato del risultato della conversione A/D		
1	Giustificato a destra	I 6 bit più significativi di ADRESH sono letti come '0'
0	Giustificato a sinistra	I 6 bit meno significativi di ADRESL sono letti come '0'

Modo lavoro del bit ADFM.

Direzione	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore POR o BOR	Valore negli altri reset
05h	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0	—0x 0000	—0u 0000
85h	TRISA	-	-	Configurazione dei pin della porta						—11 1111	—11 1111
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	—0- 0000	—0- 0000

Registri associati alla porta A.

Configurazione della porta A

Per capire meglio il funzionamento della porta A faremo un esercizio: supponiamo di avere un sistema di climatizzazione regolato da un termostato. Quando la temperatura supera il massimo consentito dobbiamo mettere in funzione tre ventilatori.

Un termostato è un dispositivo che quando rileva una determinata temperatura chiude un contatto, quindi avremo bisogno di un ingresso digitale (abbiamo solo due stati, contatto aperto e contatto chiuso). Per attivare i ventilatori (due stati, ON e OFF) avremo bisogno di tre uscite digitali. La porta A rimane configurata nel modo indicato dalla figura.

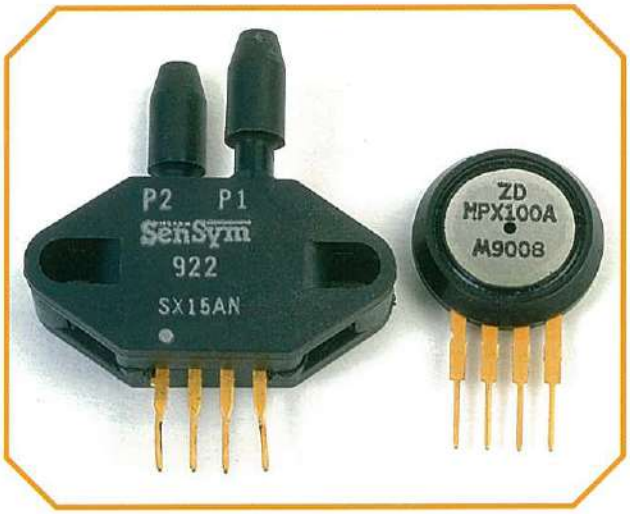
Anche se non conosciamo le istruzioni del microprocessore, è facile seguire i passi che abbiamo indicato per configurare la porta.

Passo 1: Cancellare le porte dai valori residui. Per evitare possibili errori di funzionamento è consigliabile pulire i latch o flip-flop delle porte del microcontroller che vogliamo utilizzare.

Passo 2: Dato che i registri con cui vogliamo lavorare (TRISA e ADCON1) si trovano su un banco di memoria diverso da PORTA, cambieremo banco.

PCFG3: PCFG0	AN7[1] RE2	AN6[1] RE1	AN5[1] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs[2]
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss	2/1
011x	D	D	D	D	D	D	D	D	VDD	Vss	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

Configurazione dei terminali della porta A in funzione dei bit di configurazione.



Sensori analogici di pressione.

Passo 3: Configurazione della porta. Scriveremo ADCON1 con i valori adeguati per il modo lavoro dei pin richiesti nell'applicazione. Grazie al valore impostato i pin della porta A saranno I/O digitali.

Passo 4: Definire gli ingressi e le uscite. Scrivendo su TRISA selezioneremo quali pin vogliamo far lavorare come ingressi e quali come uscite. Nel nostro esempio abbiamo selezionato un unico ingresso e il resto dei pin come uscite. L'esempio verrà risolto con una semplice lettura ciclica del bit RA0, in modo che quando diventa 1 significa che il termostato ha chiuso il suo contatto e devono essere attivate le tre uscite per i ventilatori.

```
clrf    PORTA           ;Cancella i latch di uscita
bsf     STATUS,RP0      ;Seleziona banco 1
movlw   b'00000111'
movwf   ADCON1          ;Si configura la porta A come I/O
                         digitali
movlw   b'00000001'
movwf   TRISA           ;RA0 si configura come ingresso e
                         il resto come uscite
```

La porta B

Un altro dei dispositivi del PIC 16F870 è la porta B, che ha 8 linee bidirezionali. L'indirizzo dei dati si indica sul registro TRISB. Se impostiamo a 1 un bit del registro TRISB il suo corrispondente bit del registro PORTB rimarrà configurato come ingresso. Se disattiviamo, impostandolo a 0, un bit del registro TRISB, il corrispondente bit del registro PORTB rimarrà configurato come uscita. Fino a ora l'unica differenza tra la porta A e la porta B consiste nel numero dei pin, però se analizziamo le funzioni e l'architettura interna di ognuna di queste porte potremo osservare grandi differenze.

Funzioni dei terminali della porta B

Come abbiamo già detto parlando della porta A, le caratteristiche dimensionali dei microcontroller impongono che la maggioranza dei loro pin possa realizzare più di una funzione. Come nella porta B, oltre a I/O digitali, alcuni pin hanno multiplexate altre funzioni.

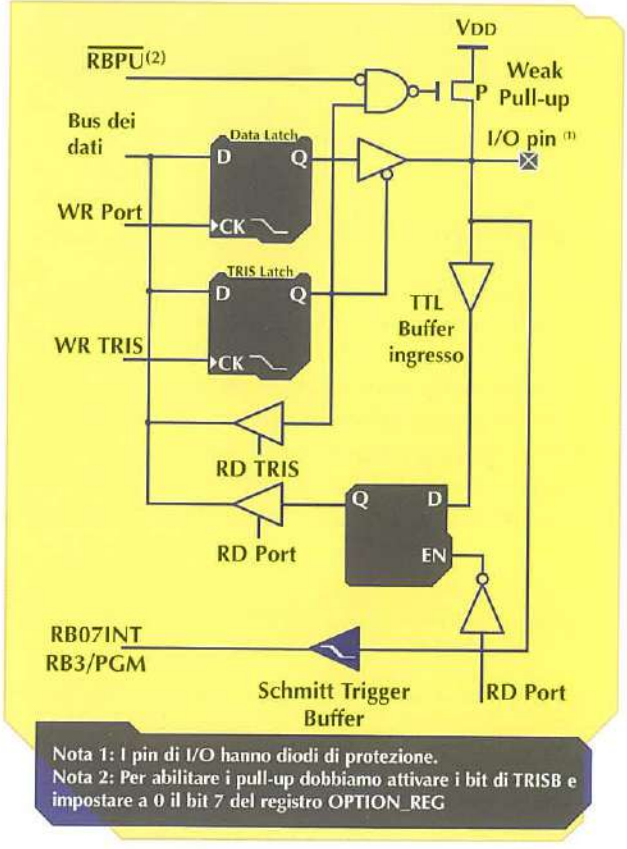
Il terminale RB0 si può configurare come un ingresso/uscita digitale o come ingresso per un interrupt esterno. In questo secondo modo, attivando il terminale, il processore genera un interrupt che ferma l'esecuzione del programma in corso e risolve la routine di interrupt, che sono altre istruzioni il cui indirizzo è indicato sul vector di interrupt posto all'indirizzo 0004h della memoria di codice.

Studiando le caratteristiche del PIC abbiamo visto che ha la possibilità di essere programmato per via elettrica, oltre a permettere la programmazione a bassa tensione (5 volt). Se vogliamo utilizzare questa caratteristica applicheremo il positivo dell'alimentazione al pin 1 del PIC ($\overline{MCLR}$/Vpp) e la massa al pin RB3/PGM. La scrittura del codice sulla memoria si realizza sempre per via seriale: si applicano gli impulsi del clock per il sincronismo tra l'ingresso dei dati e il processore sul terminale RB6/PGC, e i dati vengono caricati via seriale tramite il terminale RB7/PGD.

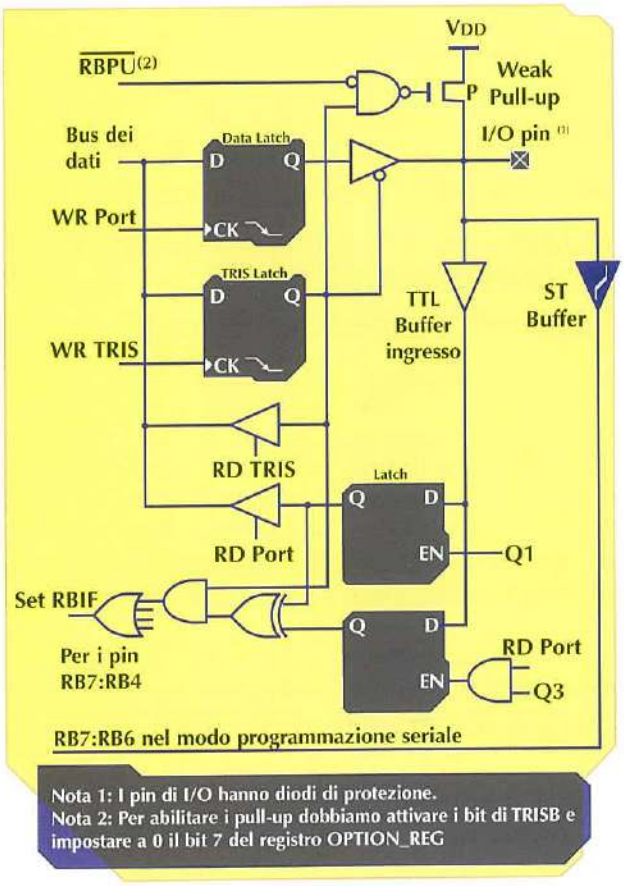
Nome	Bit #	Buffer	Funzione
RB0/INT	bit 0	TTL/ST(1)	Ingresso/Uscita digitale o richiesta esterna di interrupt
RB1	bit 1	TTL	Ingresso/Uscita digitale
RB2	bit 2	TTL	Ingresso/Uscita digitale
RB3/PGM	bit 3	TTL/ST(1)	Ingresso/Uscita digitale o massa per la programmazione a bassa tensione
RB4	bit 4	TTL	Ingresso/Uscita digitale
RB5	bit 5	TTL	Ingresso/Uscita digitale
RB6/PGC	bit 6	TTL/ST(2)	Ingresso/Uscita digitale o clock nella programmazione seriale
RB7/PGD	bit 7	TTL/ST(2)	Ingresso/Uscita digitale o dati nella programmazione seriale

(1) Questo buffer avrà un ingresso tipo ST (Schmitt Trigger) quando sarà programmato come interrupt o programmazione in stato basso (bit 0 e bit 3).
(2) Questo buffer sarà un ingresso tipo ST quando si utilizza per programmazione seriale.

Nome e funzione dei terminali della porta B.



Architettura interna dei terminali RB <3:0>.



Architettura interna dei terminali RB <7:4>.

Architettura interna

Tutti i terminali della porta B dispongono di un pull-up interno. Mediante il bit $\overline{\text{RBPU}}$, bit più significativo del registro OPTION_REG, i terminali della porta possono essere collegati al positivo dell'alimentazione (impostando a 0 $\overline{\text{RBPU}}$) grazie al pull-up. Quando i terminali della porta sono configurati come uscita vengono automaticamente scollegati dal pull-up. Se si genera un reset per collegamento dell'alimentazione (POR) si scollegano tutti i pull-up.

Osservando la figura possiamo vedere che il pull-up è un transistor CMOS di tipo P, quindi per l'abilitazione del pull-up è necessario che RBPU sia disattivato (a 0) e che il latch di TRISB contenga un 1 (ingresso).

Quattro terminali della porta B, RB <7:4>, possono essere programmati per generare un interrupt quando cambia il loro stato logico. Questo interrupt si potrà generare solamente se i terminali saranno configurati come ingressi. I terminali di ingresso RB7:RB4 si comparano con il valore acquisito (mantenuto sul latch) dall'ultima lettura di PORTB e se non coincidono i valori e il bit di abilitazione lo autorizza, verrà eseguita una richiesta di interrupt al processore. Quando si produce un interrupt di questo tipo, il bit RBIF del registro INTCON (INTCON<0>) si attiva (passa a 1) automaticamente. L'interrupt può risvegliare il microcontroller da uno stato di SLEEP.

L'utente, nella routine di servizio all'interrupt, può disattivare questo flag nei seguenti modi:

— Eseguendo una lettura o una scrittura di PORTB. Terminerà la condizione della comparazione.

— Disattivando il bit RBIF.

Questa caratteristica è molto utilizzata quando si lavora con una tastiera e vogliamo rilevare se è stato premuto un tasto. Premendo il tasto cambia lo stato dell'ingresso e si produce un interrupt che richiama la routine specifica per individuare il tasto premuto.

Per configurare il terminale RB0/INT per lavorare con un interrupt esterno, dobbiamo utilizzare il bit INTEDG del registro OPTION_REG (OPTION_REG <6>).

Dopo aver visto la PORTB, o porta B, dato

Indirizzo	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore in POR o in BOR	Valore negli altri reset
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	Configurazione dei pin della porta B								11111111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	11111111	1111 1111

x: sconosciuto, u: inalterato; le celle ombreggiate non intervengono nella configurazione della porta B. Registri associati alla porta A.

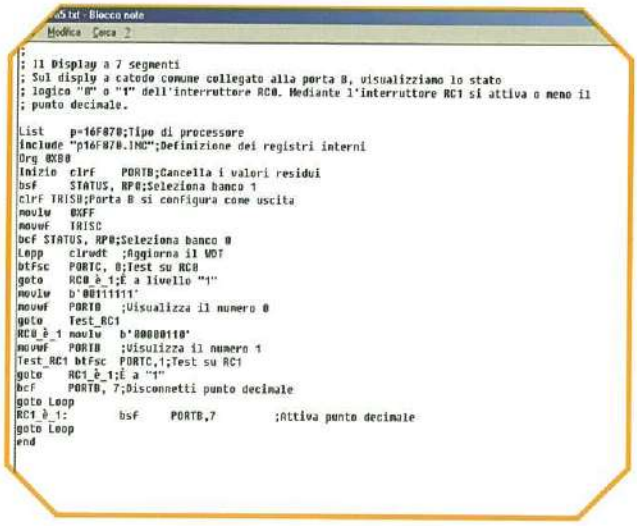
clrf	PORTB	Cancella i latch di uscita
bsf	STATUS,RP0	Seleziona banco 1
movlw	b'11111111'	
movwf	TRISB	La porta B si configura come ingressi digitali

Esempio di configurazione della porta B.

che si utilizzano indistintamente entrambe le denominazioni, chiariamo una delle idee fondamentali per ottenere una buona programmazione. Quando la complessità di un programma è alta, è necessario essere molto ordinati per evitare di creare una situazione confusa, dobbiamo ricordare che i grandi programmi esigono il lavoro perfettamente organizzato di molti programmatori.

Metodologia

Con la varietà di linguaggi di programmazione esistente e le caratteristiche particolari di ogni progettista, se non si rispettano le regole diventa molto difficile interpretare i programmi. Le stesse regole si possono applicare alla maggioranza dei linguaggi di programmazione e praticamente ai progettisti di tutto il mondo. Così, se dovremo mettere mano a un programma che non abbiamo sviluppato noi, saremo capaci di interpretarlo e anche di modificarlo.



Classificazione dei linguaggi di programmazione.

Si chiama "Metodologia della Programmazione" l'insieme di regole che si seguono nella fase di progetto del programma.

È fondamentale strutturare il codice di un'applicazione e applicare delle regole. Nelle figure possiamo osservare la differenza esistente tra un codice che non segue nessuna regola di programmazione e lo stesso codice strutturato. Supponiamo ora un programma molto più esteso che venga realizzato senza seguire alcuna regola, risulterà un caos anche per lo stesso programmatore che lo avrà creato.

```
; Il Display a 7 segmenti
; Sul disply a catodo comune collegato alla porta B, visualizziamo lo stato
; logico "0" o "1" dell'interruttore RC0. Mediante l'interruttore RC1 si attiva o meno il
; punto decimale.

List    p=16F870;Tipo di processore
include "p16F870.INC";Definizione dei registri interni
Org 0X00
Inizio  clrf    PORTB;Cancella i valori residui
bsf     STATUS, RP0;Seleziona banco 1
clrf TRISB;Porta B si configura come uscita
movlw   0XFF
movwf   TRISC
bcf STATUS, RP0;Seleziona banco 0
Lopp    clrwdt  ;Aggiorna il WDT
btfsc   PORTC, 0;Test su RC0
goto    RC0_è_1;È a livello "1"
movlw   b'00111111'
movwf   PORTB   ;Visualizza il numero 0
goto    Test_RC1
RC0_è_1 movlw   b'00000110'
movwf   PORTB   ;Visualizza il numero 1
Test_RC1 btfsc  PORTC,1;Test su RC1
goto    RC1_è_1;È a "1"
bcf     PORTB, 7;Disconnetti punto decimale
goto Loop
RC1_è_1:        bsf     PORTB,7         ;Attiva punto decimale
goto Loop
end
```

Esempio di programma senza seguire le regole di programmazione.

```
; Il Display a 7 segmenti

; Sul disply a catodo comune collegato alla porta B, visualizziamo lo stato
; logico "0" o "1" dell'interruttore RC0. Mediante l'interruttore RC1 si attiva o meno il
; punto decimale.

                List    p=16F870        ;Tipo di processore
                include "p16F870.INC"   ;Definizione dei registri interni

                Org 0X00

Inizio          clrf    PORTB           ;Cancella i valori residui
                bsf     STATUS, RP0     ;Seleziona banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   0XFF
                movwf   TRISC
                bcf     STATUS, RP0     ;Seleziona banco 0

Lopp            clrwdt                  ;Aggiorna il WDT
                btfsc   PORTC, 0        ;Test su RC0
                goto    RC0_è_1         ;È a livello "1"
                movlw   b'00111111'
                movwf   PORTB           ;Visualizza il numero 0
                goto    Test_RC1

RC0_è_1         movlw   b'00000110'
                movwf   PORTB           ;Visualizza il numero 1

Test_RC1        btfsc   PORTC,1         ;Test su RC1
                goto    RC1_è_1         ;È a "1"
                bcf     PORTB, 7        ;Disconnetti punto decimale

                goto Loop
RC1_è_1:        bsf     PORTB,7         ;Attiva punto decimale
                goto Loop
```

Lo stesso esempio di programma, però strutturato.

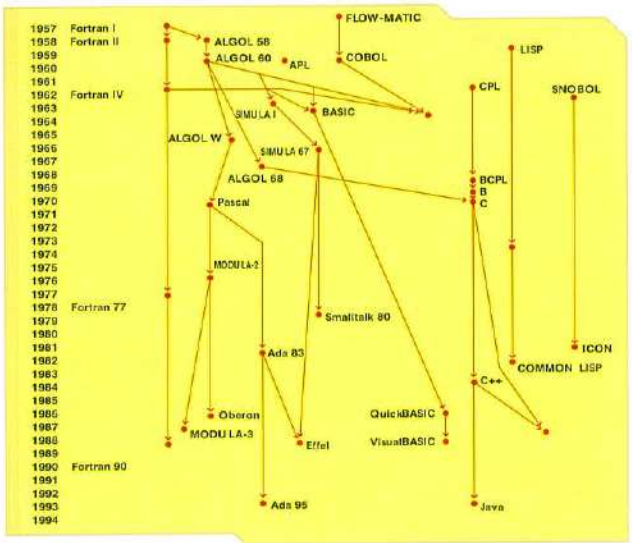
Linguaggi di programmazione

Il linguaggio di programmazione è il mezzo col quale comunichiamo con il computer e comprende l'insieme di note, regole, simboli, espressioni, ecc. che descrivono algoritmi e strutture dei dati. Benché esistano moltissimi concetti comuni a tutti i linguaggi, la grande varietà di applicazioni fa sì che esistano molti linguaggi di programmazione differenti e che vengano applicante anche diverse metodologie.

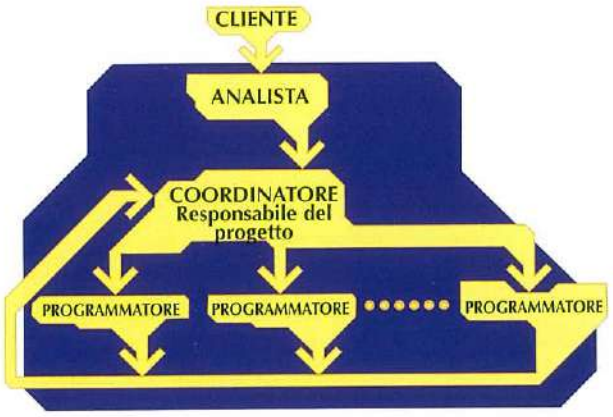
Esistono linguaggi specifici per il calcolo numerico (Fortran, Matlab), gestione (Cobol), programmazione Web (HTML), orientati a oggetti (Visual Basic), per il real time (C, Assembler), di utilizzo generale (Pascal, C), ecc. Alcuni di questi linguaggi si adattano a quello del proprio programmatore e in seguito traducono il codice creato in uno leggibile per il computer, sono chiamati linguaggi di alto livello; esistono anche linguaggi di basso livello i cui algoritmi sono vicini al codice macchina ed è il programmatore a doversi adattare a essi per sviluppare il codice.

I linguaggi di basso livello utilizzano un compilatore per tradurre le istruzioni in codice binario della macchina. I linguaggi di alto livello utilizzano un interprete che traduce il codice creando istruzioni di basso livello che successivamente, a loro volta, verranno trasformate in binario.

Per programmare il nostro microcontroller utilizzeremo l'assembler. Il linguaggio assembler è un linguaggio di basso livello che fornisce l'opportunità di conoscere a fondo le operazioni del processore, mantenendo uno stretto controllo su di esse. Anche se l'utilizzo dell'assembler richiede l'impiego di un maggior numero di istruzioni per eseguire un'azione, dota i programmi di maggior rapidità e li rende più compatti, ottimizzando la loro esecuzione.



Evoluzione storica dei linguaggi di alto livello.



Struttura tipica di sviluppo di un progetto.

Organizzazione

Ora conosciamo l'importanza di utilizzare una metodologia al momento di programmare, però cosa succede se sviluppiamo il nostro lavoro di programmatori per un'azienda?

Molti dei progetti che si realizzano in un'azienda hanno un grado di complessità elevato. In questi casi il progetto viene affrontato congiuntamente da un gruppo di programmatori.

Quindi gli analisti avranno l'incarico di specificare cosa deve fare il nostro programma. Essi definiranno il progetto nel suo insieme indicando i modelli dello sviluppo.

Ci sarà anche un altro coordinatore con il compito di assegnare a ogni programmatore la parte da sviluppare e, solitamente, di assemblare tutte le parti in un insieme globale. Ogni programmatore svilupperà un compito specifico.

È molto importante che tutti seguano le stesse regole di programmazione: identificazione dell'autore, utilizzo di variabili, tabulati, commenti, utilizzi di maiuscole e minuscole, ecc., sono alcuni degli aspetti che si raccolgono nei capitolati tecnici di un'azienda.

La porta C

La porta C completa l'analisi di uno dei dispositivi più importanti del PIC16F870, le porte di I/O. Inoltre avanziamo nel vasto campo della programmazione analizzando i possibili errori che potremmo commettere al momento di confezionare un programma.

La porta C* dispone di otto linee bidirezionali. Per indicare se i terminali lavorano come ingressi o come uscite si utilizza il registro TRISC. Attivando un bit del registro TRISC si indica quindi che il terminale corrispondente della porta verrà utilizzato come ingresso, e se imposteremo a 0 un bit del registro TRISC il terminale corrispondente del registro PORTC rimarrà configurato come uscita.

Il PIC16F870 dispone di tre porte, A, B e C (altri modelli superiori possono avere anche le porte D ed E). Studiando le due porte precedenti abbiamo potuto verificare come le differenze tra le porte siano radicate nelle funzioni che hanno multiplexate e nell'architettura interna delle stesse. Con la porta C avviene la stessa cosa, il funzionamento di base è uguale, cambia però la possibilità di multiplexare le sue funzioni di I/O di dati digitali con altre, necessarie per altri dispositivi del microcontroller.

Nome	Bit #	Buffer	Funzione
RC0/T1OSO/T1CKI	bit 0	ST	Ingresso/Uscita digitale o uscita dell'oscillatore del TMR1 o ingresso del clock per il TMR1
RC1/T1OSI	bit 1	ST	Ingresso/Uscita digitale o ingresso dell'oscillatore del TMR1
RC2/CCP1	bit 2	ST	Ingresso/Uscita digitale o ingresso capture/uscita compare/uscita PWM
RC3	bit 3	ST	Ingresso/Uscita digitale
RC4	bit 4	ST	Ingresso/Uscita digitale
RC5	bit 5	ST	Ingresso/Uscita digitale
RC6/TX/CK	bit 6	ST	Ingresso/Uscita digitale o Trasmissione USART asincrona o clock modo sincrono
RC7/RX/DT	bit 7	ST	Ingresso/Uscita digitale o Ricezione USART asincrona o dati modo sincrono

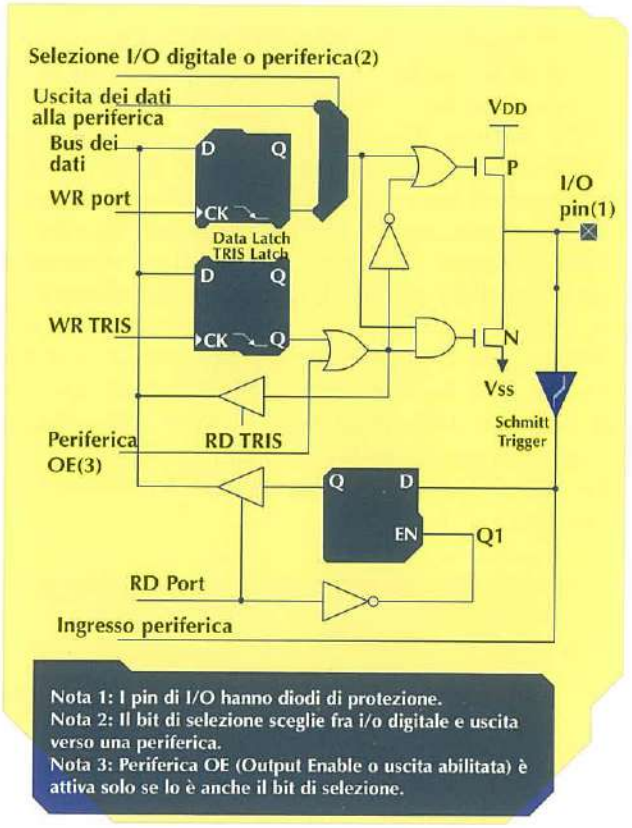
Nome e funzione dei terminali della porta C.

Funzioni dei terminali della porta C

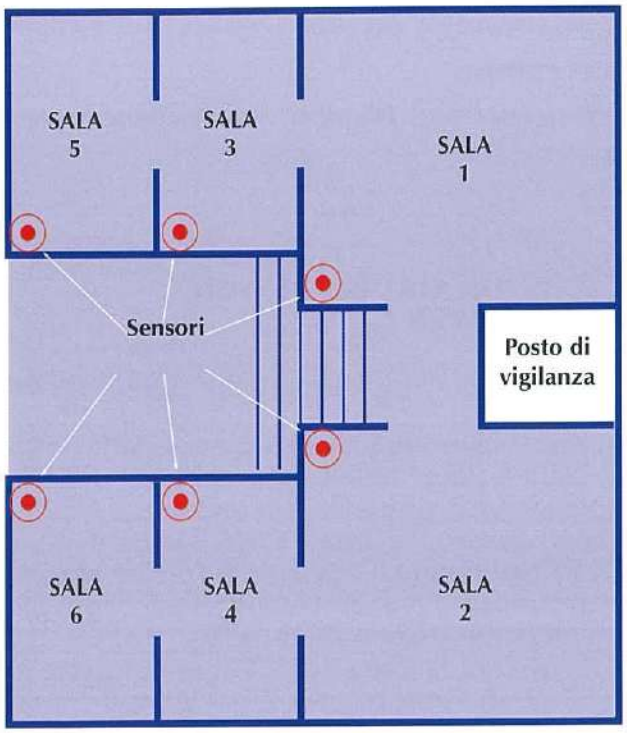
I terminali della porta C hanno multiplexate diverse funzioni che sono utilizzate da alcune periferiche interne. Cinque piedini della porta C, oltre a poter funzionare come I/O digitali, possono assolvere ad altre funzioni.

Per questo, le linee RC0/T1OSO/T1CKI e RC1/T10SI hanno il compito di fornire al temporizzatore 1 (Timer 1 o TMR1) dei collegamenti richiesti verso l'esterno.

Mediante la linea RC2/CCP1 si permette al modulo di Capture/Compare/PWM di cui dispone il PIC16F870, di poter interagire con l'esterno. Questa linea permetterà al disposi-



Architettura interna dei terminali della porta C.



Un progetto molto semplice per applicare le conoscenze acquisite.

```
        List    p=16F870        ;Processore
        include "P16F870.INC"   ;Definizione dei registri interni

        ORG     0xB0
inizio  clrf    PORTB           ;Azzera i valori casuali dato che la utilizzeremo come uscita
        bsf     STATUS,RP0      ;Seleziona banco 1
        clrf    TRISC           ;Porta C configurata come uscita
        movlw   0xFF
        movwf   TRISB           ;Porta B configurata come ingresso
        bcf     STATUS,RP0      ;Seleziona banco 0
Loop    movf    PORTB,W         ;Legge gli ingressi
        movwf   PORTC           ;Li porta sull'uscita
        goto    Loop            ;Ciclo senza fine

        end                     ;Fine del programma
```

Configurazione delle porte per risolvere l'esempio.

tivo di avere un ingresso per realizzare un'acquisizione, poter offrire il risultato di una comparazione o fornire un'uscita in modo PWM. Infine, tramite la porta C, è anche possibile la comunicazione in modo asincrono e sincrono. Il terminale RC6/TX/CK ha multiplexate la funzione di I/O digitale con quella di trasmissione seriale in modo sincrono per il modulo USART e quella di ricezione degli impulsi di clock che segneranno il sincronismo nel trasferimento seriale in modo sincrono. Il terminale RC7/RX/DT può lavorare come I/O digitale, come ricevitore di dati in modo asincrono per la USART e come I/O di trasferimento seriale in modo sincrono.

Architettura interna della porta C

I terminali della porta C hanno buffer di ingresso Trigger di Schmitt. Nella figura della pagina precedente è possibile osservare lo schema interno di ogni linea della porta C. Mediante il bit di selezione della funzione del terminale diamo l'ordine al multiplexer per fare in modo che il terminale funzioni come I/O digitale, ovvero a disposizione della periferica. Il segnale periferico OE si attiva solamente quando vogliamo lavorare con il dispositivo interno; si tratta di un segnale di abilitazione che attiva l'uscita della periferica interna. Quando, con la porta C, si lavora con un dispositivo interno, que-

Indirizzo	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore in POR o in BOR	Valore negli altri reset
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	Configurazione dei pin della porta C								11111111	1111 1111

x: sconosciuto, u: non cambia

Registri associati alla porta.

st'ultimo può richiedere che il terminale a cui fa riferimento sia configurato come ingresso oppure, in altri casi, come uscita e per questo il progettista dovrà prestare particolare attenzione nel configurare il registro TRISC in modo adeguato. In molti casi dovremo modificare il valore di questo registro diverse volte nel corso di un programma.

Esempio pratico

Supponiamo di dover eseguire un progetto in cui si voglia sviluppare un sistema di allarme grazie al quale il vigilante di un museo possa controllare, da un posto di guardia, le sei sale di cui è composto il museo.

Se montiamo un sensore o un rilevatore di presenza in ogni sala, questi darà un segnale digitale quando rileverà un intruso. Avremo bisogno di sei ingressi digitali da dedicare ai sensori e, ad esempio, sei uscite digitali collegate a dei diodi LED che indicheranno in quale sala è stato generato il segnale di allarme.

Se osserviamo come sono state configurate le porte, vedremo che con la porta B avremo otto ingressi digitali e, grazie alla porta C, potremo fornire otto uscite digitali. Quindi l'unica cosa che dovrà fare il programma sarà quella di leggere continuamente gli ingressi e passare il valore letto alle uscite. Il sistema può essere ampliato, dato che ci avanzano due ingressi e due uscite. Possiamo controllare così, due variabili in più e lavorare su due elementi diversi. È possibile anche complicare un po' il programma in modo che, mediante temporizzatori, si attivi l'allarme se il segnale si mantiene per un determinato tempo, creare un allarme intermittente, attivare un cicalino, ecc.

Linea di codice	movf	PORTC,W	
Token 1	mov	Istruzione	Istruzione del processore
Token 2	PORTC	Operando	Registro
Token 3	W	Operando	Registro

Esempio tokens in una linea di codice.

Errori nella programmazione

Quando si realizza un programma, è facile commettere errori. Dobbiamo prestare attenzione quando programmiamo, perché molte volte mettere a punto un programma risulta un lavoro molto complicato. L'utilizzo di una metodologia di programmazione è un buon aiuto per evitare gli errori e per rilevarli o eliminarli, nel caso ne avessimo fatto qualcuno.

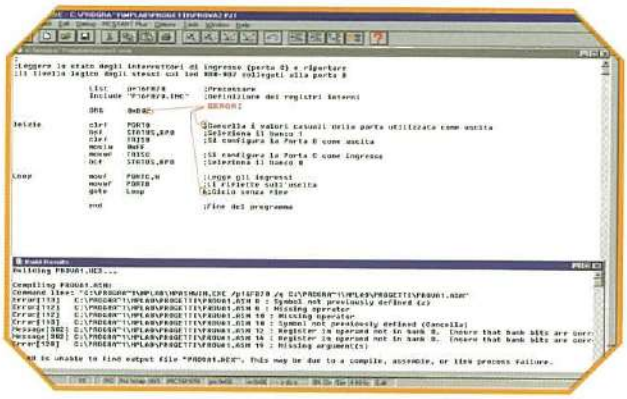
Molti compilatori ci indicano se abbiamo commesso errori, dove e di che tipo, però un compilatore non sa qual è lo scopo del nostro programma, non sa se ciò che sta traducendo in linguaggio macchina farà realmente quello che il programmatore desidera fare.

Errori lessicali, sintattici e semantici

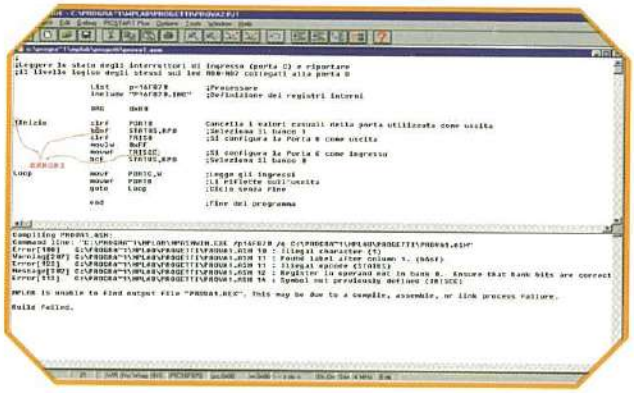
Gli errori che possiamo commettere programmando si possono classificare in tre tipi: lessicali, sintattici e semantici.

Livello lessico

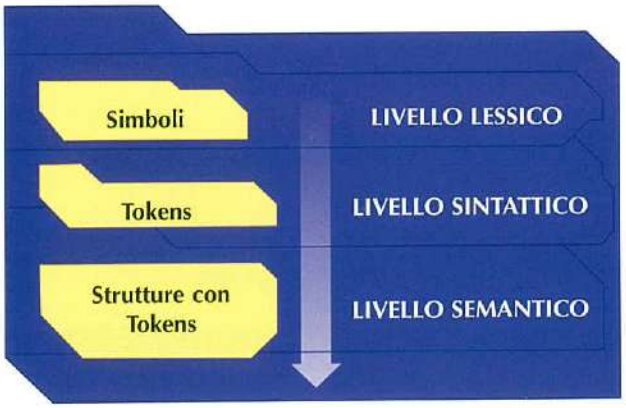
Quando si parla del livello lessico si fa riferimento alla minore entità possibile da analizzare ovvero, nel nostro caso, i simboli e i carat-



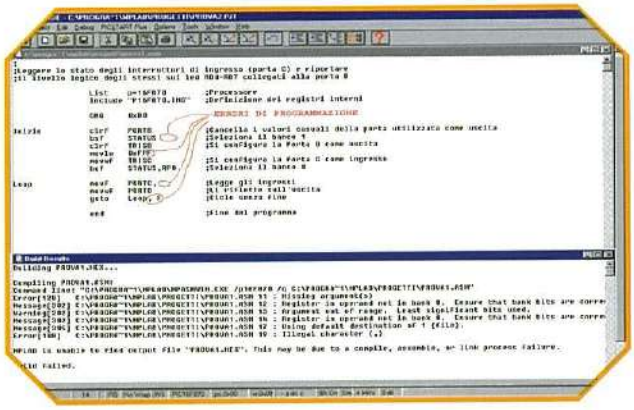
Programma con errori lessicali.



Programma con errori di sintassi.



Diversi livelli di analisi.



Programma con errori semantici.

teri. Quando utilizziamo un simbolo non permesso dal linguaggio di programmazione, stiamo commettendo un errore lessicale. Nel programma della figura sono stati inseriti caratteri che non sono accettati dal compilatore. Come potrete verificare, mediante il compilatore è possibile identificare la linea dove è stato commesso l'errore e in cosa consiste quest'ultimo. Quando il compilatore rileva un errore non esegue la traduzione in codice macchina, e in questo esempio ne sono stati trovati tre. Il compilatore avvisa anche di possibili situazioni che potrebbero provocare un funzionamento anomalo del programma, gli avvisi o warning non impediscono la compilazione.

Livello sintattico

Una combinazione di simboli raggruppati creerà ciò che si chiama un token, ovvero ognuna delle parti di una linea di codice in assembler. Nella figura della pagina precedente possiamo vedere un esempio di ciò che si intende per token. Gli errori lessicali si verificano quando l'associazione di simboli o caratteri non creano token validi.

Osservate ciò che dice il compilatore quando cerchiamo di compilare il programma della figura. Il primo errore lo dà il token "1Inizio" che è un'etichetta e come tale non può iniziare con un numero. Il secondo errore si trova sulla linea di codice successiva con il token "bt6s". il compilatore ci informa che si tratta di un'istruzione non valida, codice OP illegale. Il terzo errore ce lo dà quando stiamo indirizzando un'operazione a un registro inesistente (TRISCC).

Livello semantico

Il raggruppamento di diversi token crea una struttura, una linea di codice, che ha un significato. Nel livello semantico si vuole analizzare il significato delle strutture. Il primo errore che contiene il programma è sulla linea 11, per errore di operando: stiamo utilizzando un'istruzione che gestisce un bit, forniamo il registro su cui si trova questo bit, ma non il bit su cui deve agire.

Il compilatore ci avvisa mediante una warning dell'errore successivo di programmazione, che consiste nel fatto che il valore indicato nell'operando è fuori range (si vuole inserire un valore di 12 bit –FFFh- su un registro da 8 bit –W-). Le warning non impediscono al compilatore di tradurre il codice in linguaggio macchina. L'errore successivo ci verrà indicato con un messaggio. Il compilatore nei messaggi ci informa delle cose che non capisce ma che sono corrette, e che possono essere volontà del programmatore; praticamente consiste in un semplice: Sei sicuro di fare questo? Infine, abbiamo commesso un altro errore, perché l'istruzione situata sulla linea 19 (goto) accetta solamente un operando, un solo token dopo di essa. Corretti gli errori e compilato il programma potrebbe succedere che quest'ultimo non lavori come realmente desideriamo. L'errore in molti di questi casi si può trovare in un'errata impostazione iniziale del progetto.

Errore	Errore inaccettabile per il compilatore	NON compila
Warning	Avviso di possibile errore o incongruenza nel codice	Compila
Message	Messaggio al programmatore che avvisa di una possibile situazione di errore	Compila

Messaggi del compilatore di MPLAB.

Composizione di un programma

Indipendentemente dal linguaggio di programmazione che si utilizza, alcune caratteristiche sono comuni in tutti i programmi. Possono esistere alcune differenze tra le parti che compongono un programma, l'ordine di queste, l'obbligatorietà dell'utilizzo o la loro denominazione, però possiamo raggrupparle per la loro funzione e, in questo modo, renderle comuni per qualsiasi linguaggio.

Routine

Un programma deve avere come minimo una routine di esecuzione e, benché dipenda dalla complessità dell'applicazione, normalmente sarà composta anche da diverse subroutine o sottoprogrammi. In una subroutine o sottoprogramma si inserisce un codice che risolve un'azione specifica. Ogni subroutine contiene istruzioni che sono organizzate in strutture di controllo. Sia le subroutine che le istruzioni e le strutture di controllo richiedono, per il loro funzionamento, l'utilizzo di variabili e costanti.

Altri elementi comuni a qualsiasi programma sono gli organigrammi che, come già sappiamo, sono rappresentazioni visive dello sviluppo dei programmi stessi.

Variabili e costanti

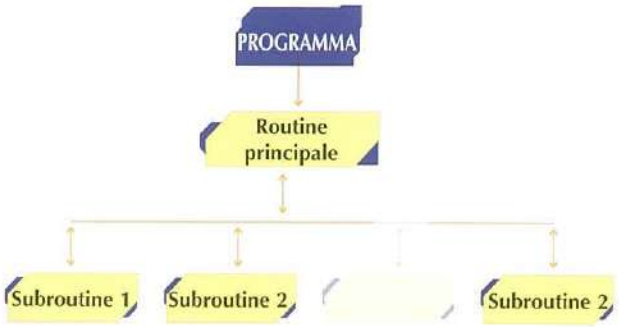
Una variabile è un contenitore che ospita un valore, il quale può essere cambiato lungo il corso del programma. Identificheremo la variabile, che potrà avere valori diversi nel tempo, con un nome.

Una costante, invece, conterrà un valore che sarà sempre uguale. Daremo un nome anche alla costante, però il valore contenuto in essa non verrà mai modificato.

L'utilizzo di variabili e di costanti facilita molto la programmazione, evitando di far riferimento a valori specifici e possibili confusioni nello sviluppo del codice. Ad esempio, immaginate un programma che risolva uno sviluppo matematico in cui utilizzeremo il valore pi (3,14159...). Se definiamo una costante e la chiamiamo PI e a essa assegniamo il valore 3,14159..., ogni volta che dovremo utilizzarlo nel programma faremo riferimento a PI e non al suo valore.

Immaginate ora un controllo di temperatura in cui un sensore registra i valori di questa ogni dieci secondi. Ogni minuto presenteremo sul display la media delle temperature. Immaginate la propensione agli errori che risulterebbe in un codice in cui si operi con tutti i valori acquisiti in forma numerica. Risulta molto più semplice definire una variabile dove scrivere questi valori e successivamente operare sulla variabile.

Sia le variabili che le costanti possono essere di diverso tipo, secondo la classe di valore che contengono. Non è la stessa cosa contenere un numero decimale invece che un testo, quindi in molti linguaggi di programmazione è necessario specificare di quale tipo sono le variabili e le costanti. A seconda del tipo si utilizzerà più o meno spazio per memorizzare il contenuto. In alcuni linguaggi di programma-



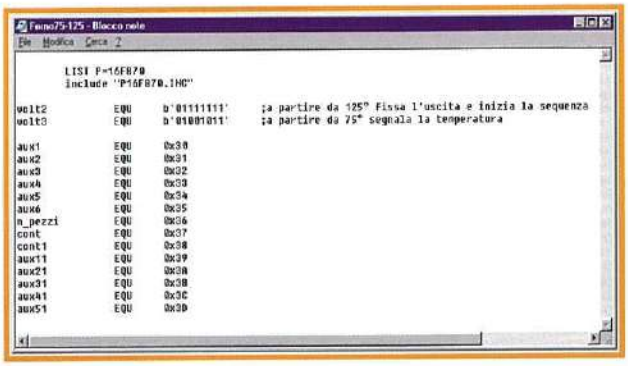
Composizione generale di un programma.

Campioni	1	2	3	4	5	6	Operazione
Valore	22,4	22.6	22.6	22.5	22.3	22.1	$\frac{22,4+22,6+22,6+22,5+22,3+22,1}{6}$
Variabile	Temp	Temp+1	Temp+2	Temp+3	Temp+4	Temp+5	$\frac{\sum \text{Temp}}{6}$

Esempio di utilizzo delle variabili per semplificare le operazioni.

Tipi di dati	Esempio di valori	Esempio di dichiarazione
Numeri esadecimali	4D	tempo EQU 4Dh
Numeri binari	10110001	valore EQU b'10110001'
Numeri decimali	26	int var=26
Simboli	HOLA	string messaggio='HOLA'
Dispositivi	16F870	LIST P=16F870

Tipi di dati ed esempi di dichiarazione.



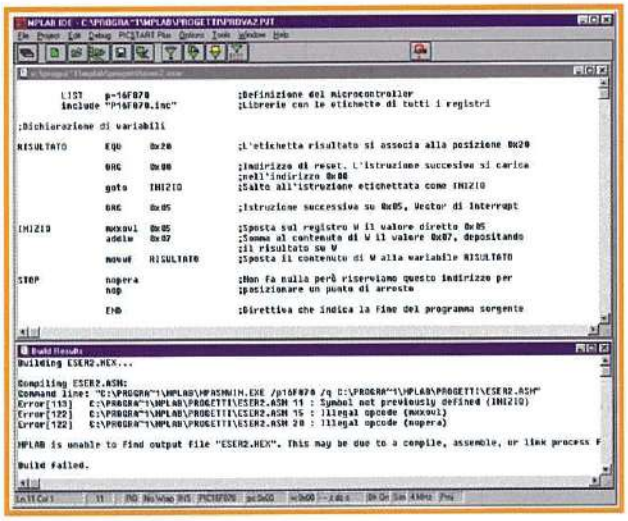
Esempio di dichiarazione di costanti e variabili in assembler.

zione è necessario inizializzare la variabile con un valore, anche se nel corso del programma verrà modificato.

Istruzioni

Le istruzioni sono il modo che abbiamo di indicare al processore di realizzare un'azione specifica. Dalla corretta definizione dell'istruzione dipenderà l'esecuzione dell'azione nel modo desiderato. All'interno di un programma un'istruzione ha un nome specifico (mnemonico), dei parametri e un determinato ordine e fa riferimento a un'azione.

Nei linguaggi di alto livello un'istruzione può indicare al processore di realizzare diversi lavori, invece, nei linguaggi di basso livello vicini al linguaggio macchina, le istruzioni non contengono operazioni multiple.



Errori che rileva il compilatore a causa di un errato utilizzo delle istruzioni.

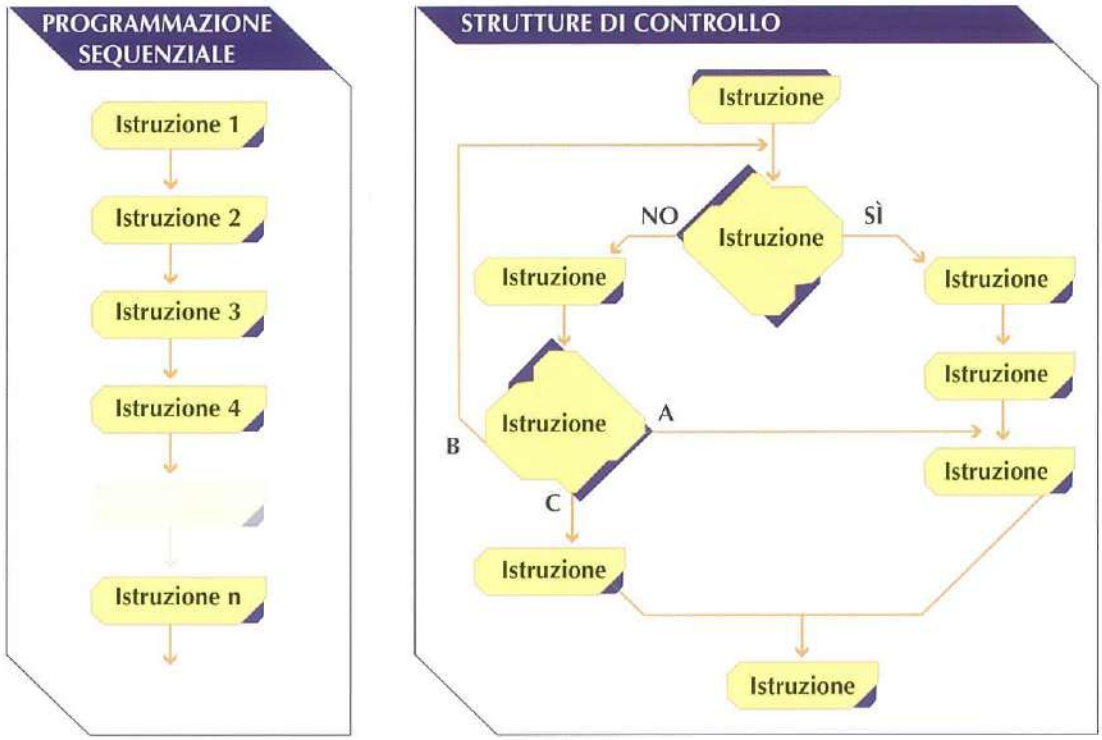
Abbiamo visto in precedenza che quando ci sbagliamo con il mnemonico dell'istruzione o con gli operandi di questa, si produce un errore che impedisce la compilazione del programma. Un errore si produrrà anche se, utilizzando le istruzioni, creeremo una struttura di controllo sbagliata, dobbiamo fare, invece, molta attenzione a quando non commettiamo un errore di questo tipo, ma utilizziamo le istruzioni in modo non corretto.

Immaginate il microcontroller che controlla un ascensore. Un utente all'interno dell'ascensore preme il pulsante di un piano, arriva al piano e poi si aprono le porte; non sarebbe la stessa cosa se l'utente preme il pulsante, si aprono le porte e poi si va al piano.

Bisogna fare particolare attenzione con gli errori semantici, anche se molti di essi possono essere corretti utilizzando gli organigrammi.

Strutture di controllo

Quando sviluppiamo un programma dobbiamo porre le istruzioni in un ordine preciso. Se non esistessero le strutture di controllo, un programma sarebbe formato da una serie di istruzioni messe una dietro l'altra, in modo sequenziale (verrebbero eseguite le istruzioni per ordine solamente una volta). Le strutture di controllo permettono al programmatore di realizzare comparazioni, creare strutture condizionali o alternative in modo che venga eseguita una serie di istruzioni solamente se si compie una determinata condizione o in presenza di un determinato evento. L'utilizzo di queste strutture ottimizza l'utilizzo dello spazio e del tempo nei programmi.

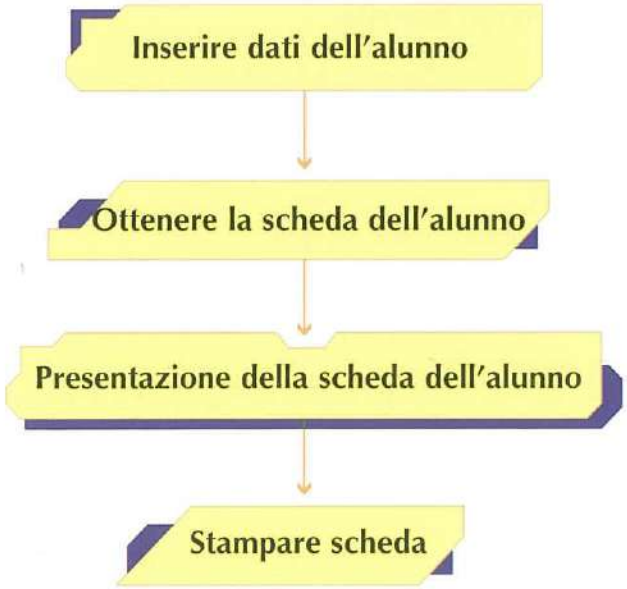


Diversi modi di sviluppare un programma.

Attualmente non si utilizza la programmazione sequenziale e tutti i linguaggi accettano la programmazione basata su strutture di controllo.

Normalmente un programma disporrà di diversi rami o percorsi, perché, anche se all'inizio non sappiamo come verrà eseguito, dobbiamo contemplare tutte le possibili opzioni.

Ogni linguaggio avrà differenti forme per esprimere le diverse strutture di controllo (condizionali, interattive, a risposta multipla...), però l'utilizzo di queste strutture risulta comune e fondamentale in tutti i linguaggi.

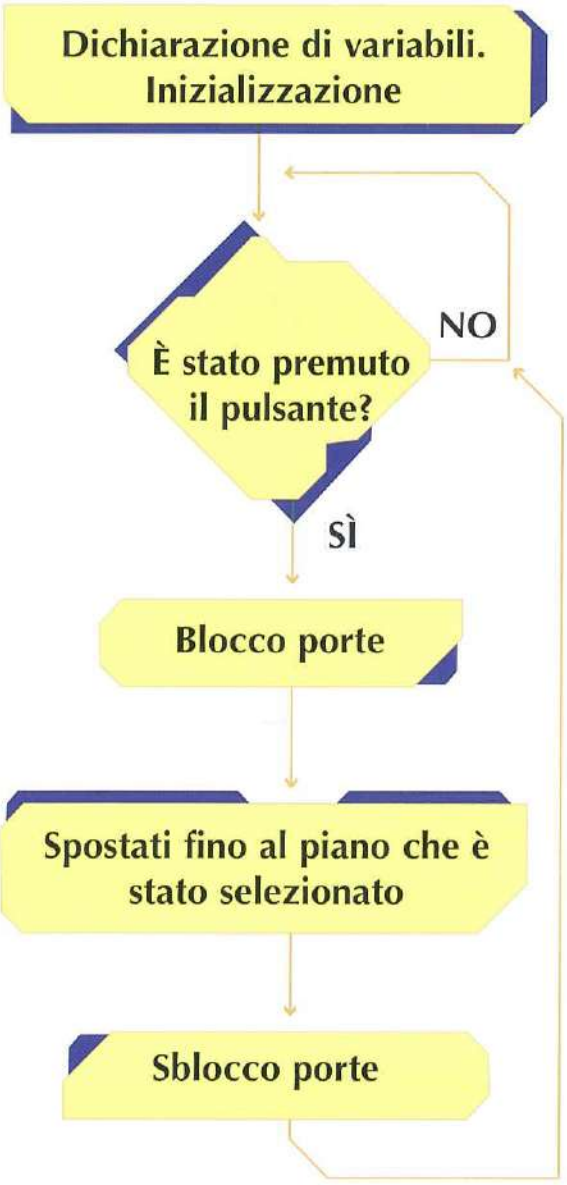


Un programma può essere composto da diverse subroutine.

Subroutine

Sappiamo che le istruzioni si organizzano perché vengano eseguite mediante strutture di controllo, ma che succede quando il programma è molto lungo o ci sono parti che devono essere ripetute in diversi punti? In questo caso i frammenti di programma che sviluppano una funzione specifica vengono raggruppati sotto un nome e si separano dal programma principale ricorrendo a essi quando è necessario. Questi raggruppamenti di codice sono conosciuti come subroutine, procedimenti o funzioni, a seconda del linguaggio di programmazione che stiamo utilizzando.

Quando si desidera eseguire questa parte del programma non sarà più necessario scriverla per intero nuovamente, ma dovremo

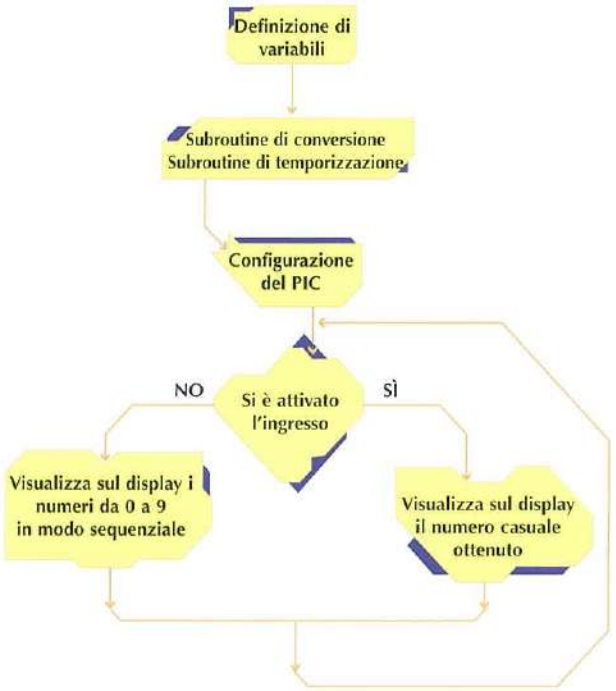


L'utilizzo di organigrammi implica una miglior disposizione nel programma delle istruzioni ed evita di commettere errori semantici.

semplicemente far riferimento al suo nome, chiamando così in esecuzione la subroutine.

Molti programmatori hanno l'abitudine di lasciare la minor quantità possibile di codice nel programma principale, in modo da ridurlo a semplici chiamate a subroutine e lasciare che queste realizzino il lavoro.

Utilizzando le subroutine evitiamo di scrivere più codice del necessario, dato che non si producono ripetizioni di codice e si ottimizza la strutturazione del programma, rendendo più semplice la messa a punto e le successive modifiche.



Un altro organigramma. Strumento fondamentale per un programmatore.

Organigrammi

Anche se non fanno parte del programma, gli organigrammi sono molto utili al programmatore. Un organigramma è la rappresentazione grafica di ciò che fa il programma e sarà la prima cosa che il programmatore crea prima di iniziare a sviluppare il codice.

Ripassiamo le regole per confezionare un organigramma:

— Ha un solo punto di ingresso e un solo punto di uscita.

— Un rettangolo indica un'azione che deve essere eseguita.

— Un rombo segnala una condizione e permette di seguire diversi percorsi.

— Mediante le frecce si ordinano le istruzioni.

— Le frecce che escono da un rombo devono essere etichettate per scegliere l'alternativa in modo corretto.

Il modo riposo: SLEEP

Molte delle applicazioni che svilupperemo resteranno in attesa che si verifichi un determinato evento. Durante questa attesa il microcontroller non avrà nulla da fare, sarà disoccupato, come, ad esempio, nel caso di un telefono cellulare. Se un microcontroller entra in un periodo di attesa non è necessario che i suoi dispositivi rimangano attivi, e sarebbe raccomandabile che il consumo non fosse uguale a quando è in funzione. Per gestire questa situazione i PIC implementano il modo riposo.

Generalità del modo riposo

Il modo riposo permette di risparmiare energia elettrica quando il microcontroller entra in lunghi periodi di attesa. Immaginate il telecomando della porta del garage con alimentazione a pile. Se mancasse il modo riposo, il microcontroller consumerebbe costantemente e le batterie dovrebbero essere sostituite di frequente. Invece, grazie al modo riposo, il microcontroller consuma il minimo, fino a quando non si preme un pulsante, momento in cui attiva tutti i suoi dispositivi ed esegue l'applicazione.

I PIC16F87X hanno un consumo tipico di 2 milliampere (mA) quando funzionano normalmente, alimentati con una tensione di 5 Vdc a una frequenza di 4 MHz. Se entrano in modo riposo il consumo si riduce a meno di 1 microampére (μA).

Per fare in modo che un PIC entri in modo riposo bisognerà utilizzare l'istruzione SLEEP, con la quale il PIC si "addormenta", diminuendo notevolmente il suo consumo di energia, dato che si ferma l'esecuzione delle istruzioni, cessa di funzionare l'oscillatore principale, i temporizzatori, i dispositivi e le linee di ingresso e uscita passano in stato di alta impedenza se non sono utilizzate o mantengono il loro stato nel caso siano utilizzate.

Sarebbe conveniente che i terminali di I/O non utilizzati siano collegati al positivo dell'alimentazione oppure a massa, per evitare qualsiasi consumo quando, in modo riposo, passano allo stato di alta impedenza. Conviene anche collegare a massa o al positivo il piedino T0CKI,

Dispositivi che utilizzano il modo riposo per ridurre il loro consumo.

	PIC16F87X	
Modo di lavoro	Normale	Riposo
Consumo	2 mA	<1 μA

Differenza di consumo tra funzionamento normale e modo riposo.

	MODO RIPOSO
Watchdog (WDT)	Attivato: si aggiorna il suo valore, si cancella, però continua a funzionare normalmente
Oscillatore principale	Smette di funzionare: non si eseguono istruzioni
Porte di I/O	Attivate: mantengono lo stato precedente Disattivate: passano a stato di alta impedenza
Bit TO e PD del registro di stato	Passano a valore 1 e 0 rispettivamente

Azioni che avvengono quando eseguiamo l'istruzione SLEEP.

ABBANDONO DEL MODO RIPOSO
Ingresso di reset esterno sul pin MCLR
Overflow del Watchdog (nel caso in cui il WDT sia attivato)
Interrupt sul pin INT, interrupt da dispositivi interni del PIC o cambio di stato sulla porta B

Eventi che provocano l'uscita del microcontroller dal modo riposo.

Bit	Posizione	Valore	Significato
PD	STATUS<3>	0	Esecuzione di SLEEP. Modo riposo
		1	Il PIC è attivo
TO	STATUS<4>	0	Il Watchdog (WDT) è in overflow
		1	Il Watchdog (WDT) non è in overflow

Bit che indicano come è stato riattivato il microcontroller.

INTERRUPT DA DISPOSITIVI INTERNI DEL PIC
Interrupt tramite TMR1. Il Timer 1 deve funzionare come contatore asincrono
Interrupt da modulo di capture CCP
Attivazione speciale di evento (Timer 1 in modo asincrono usando un clock esterno)
SSP (START/STOP) bit di rilevazione di interrupt
SSP trasmissione o ricezione in modo slave (SPI/I2C)
USART RX O TX (Modo sincrono slave)
Conversione A/D (quando la sorgente del clock è un oscillatore RC)
Operazione di scrittura nella EEPROM

Interrupt che possono fare uscire il PIC dal modo riposo.

che è l'ingresso del clock esterno per il temporizzatore Timer 0 (TMR0), così come il piedino $\overline{\text{MCLR}}$/Vpp a livello logico alto (positivo). Il Watchdog o WDT (Watchdog Timer) è un temporizzatore inizialmente caricato con un determinato valore che diminuisce fino ad arrivare a zero, momento in cui si produce un reset. Si utilizza per evitare che il processore, pur essendo in esecuzione del programma, non possa più uscire da un ciclo. Per non provocare il reset si inseriscono istruzioni di aggiornamento o di reinizializzazione in punti strategici del programma. Se, al momento di entrare nel modo riposo, il Watchdog è in funzione, esso viene aggiornato ma continua a lavorare.

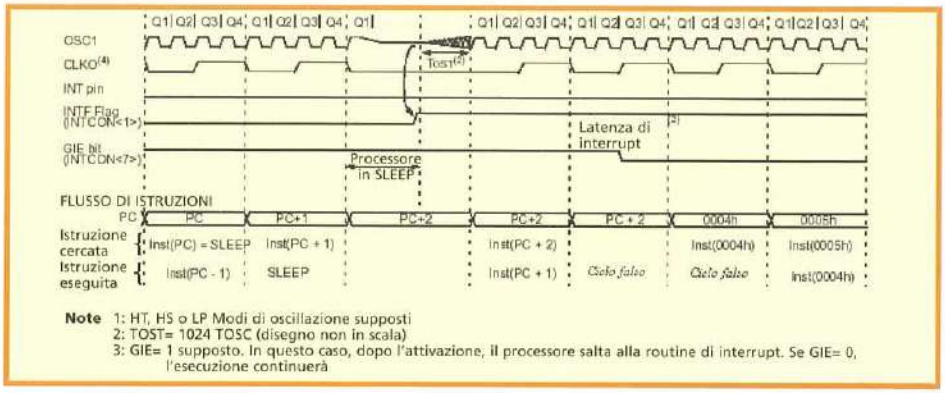
Risveglio dal modo riposo

Abbiamo visto come possiamo portare il microcontroller in modo riposo o di minimo consumo, ma cosa bisogna fare per riportarlo nello stato di funzionamento normale? Per risvegliare il PIC dal modo riposo è possibile utilizzare uno dei tre eventi che sono riportati nella figura.

Quando si riattiva il PIC grazie a uno di questi eventi, il funzionamento normale si ottiene nuovamente dopo un tempo pari a 1024.Tosc, controllato dall'oscillatore interno chiamato OST, in modo da permettere la stabilizzazione della frequenza di funzionamento.

Attivazione del terminale di reset $\overline{\text{MCLR}}$

Quando si produce questo evento, il microprocessore si riattiva ed esegue la prima istruzione del programma che si trova nel vector di Reset, generando un reset.



Note 1: HT, HS o LP Modi di oscillazione supposti
2: TOST= 1024 TOSC (disegno non in scala)
3: GIE= 1 supposto. In questo caso, dopo l'attivazione, il processore salta alla routine di interrupt. Se GIE= 0, l'esecuzione continuerà

Funzionamento del processore quando entra in modo riposo e ne esce grazie a un interrupt.

Per overflow del Watchdog

In questo caso, quando il Watchdog che era attivo al momento dell'esecuzione dell'istruzione SLEEP va in overflow, genera un reset nel processore.

Per identificare come si è verificata l'uscita dal modo riposo si eseguono dei test sui bit TO e PD del registro di stato STATUS.

Per attivazione di un interrupt

Esistono molti modi per provocare un interrupt che faccia uscire il microcontroller dallo stato di riposo. Nella figura possiamo vedere alcuni di questi modi.

Al momento dell'esecuzione dell'istruzione SLEEP, è già stata cercata l'istruzione successiva (PC+1). Per fare in modo che il PIC riparta grazie a un evento generato tramite un interrupt, il bit corrispondente di abilitazione degli interrupt (GIE) deve essere abilitato.

Se questo bit è disabilitato, il PIC continuerà a eseguire il codice con l'istruzione successiva alla SLEEP, quando si risveglierà da essa. Se il bit è stato abilitato, al risveglio il PIC eseguirà l'istruzione successiva alla SLEEP saltando poi al vector di interrupt (0004h). Se non vogliamo che il PIC esegua un'istruzione dopo lo SLEEP, utilizzeremo l'istruzione nop.

Nella figura possiamo vedere un chiaro esempio di ciò che succede quando il processore si riattiva dal modo riposo grazie a un interrupt. Come possiamo vedere, il bit GIE è attivato, e questo permette l'abilitazione degli interrupt. Entrando nel modo riposo mediante l'istruzione SLEEP, sul contatore di programma (PC) rimane scritto l'indirizzo dell'istruzione successiva alla SLEEP (PC+2) e l'oscillatore smette di funzionare. Quando si produce un interrupt il microcontroller si risveglia, però, per recuperare il funzionamento normale, necessita di un tempo TOST. Trascorso questo tempo si esegue l'istruzione indicata dal contatore di programma passando successivamente a eseguire la prima istruzione del vector di interrupt.

La parola di configurazione

Viene definita parola di configurazione una posizione riservata della memoria di programma che occupa l'indirizzo 2007h e che funziona come un registro CONFIG di configurazione. In essa si determina il modo di lavoro del microcontroller. I due bit più significativi determineranno il grado di protezione della memoria di programma, altri bit indicheranno il tipo di oscillatore che determinerà la frequenza di lavoro, altri la protezione delle memorie FLASH e EEPROM dei dati, ecc.

Nella figura della pagina successiva possiamo vedere la funzionalità di ognuno dei bit che formano la parola di configurazione, uni-

PAROLA DI CONFIGURAZIONE

CP1	CP0	DEBUG	—	WRT	CPD	LVP	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit 13													bit 0

Distribuzione dei bit della parola di configurazione.

Bit 13-12	CP1:CP0	Bit di protezione della memoria di programma anti lettura. Le coppie CP1:CP0 devono avere lo stesso valore per rendere la protezione effettiva
Bit 5-4	CP1:CP0	Bit di protezione della memoria di programma anti lettura
	11	Codice di protezione disattivato
	10	Non supportato
	01	Non supportato
	00	Codice di protezione attivato
Bit 11	DEBUG	Modo di messa a punto "in circuit"
	1	Disattivato. RB7:RB6 funzionano come linee di I/O
	0	Attivato: RB7:RB6 funzionano in modo messa a punto
Bit 10	—	Non implementato. Si legge come 1
Bit 9	WRT	Abilitazione di scrittura nella memoria FLASH
	1	Si può scrivere nella parte non protetta della memoria FLASH
	0	Scrittura disabilitata
Bit 8	CPD	Codice di protezione della memoria EEPROM dei dati
	1	Non c'è protezione nella EEPROM
	0	Protezione del codice nella EEPROM
Bit 7	LVP	Bit di abilitazione per la programmazione seriale a bassa tensione "in circuit"
	1	RB3/PGM abilita la scrittura a bassa tensione
	0	RB3/PGM funziona come I/O digitale. La programmazione si realizza ad alta tensione
Bit 6	BOREN	Bit di abilitazione per il reset per caduta di tensione
	1	BOR attivato
	0	BOR disattivato
Bit 3	PWRTEN	Bit di abilitazione per il Timer di connessione dell'alimentazione
	1	PWRTEN attivato
	0	PWRTEN disattivato
Bit 2	WDTEN	Bit di abilitazione del Timer del Watchdog
	1	WDT attivato
	0	WDT disattivato
Bit 1-0	FOSC1:FOSC0	Selezione del tipo di oscillatore
	11	RC (Resistenza-condensatore)
	10	HS (Alta Velocità. Più di 4 Mhz)
	01	XT (Standard. Da 100 KHz a 4 MHz)
	00	LP (Bassa potenza. Da 35 a 200 KHz)

Funzionalità dei bit della parola di configurazione.

tamente ai valori che devono assumere per funzionare in un modo o nell'altro.

Parole di identificazione

Si tratta di quattro parole o posizioni riservate della memoria di programma comprese tra gli indirizzi 2000 e 2003h, i cui quattro bit meno significativi possono essere utilizzati dall'utente per operazioni di verifica, codici di identificazione, numeri di serie, schede, modello, ecc. sono accessibili solamente in lettura e scrittura durante la fase di programmazione/verifica della memoria FLASH.

Conclusioni

Abbiamo visto uno dei dispositivi più importanti dei PIC, che alcune volte rappresenta un grande vantaggio rispetto ai microcontroller della concorrenza. Il risparmio di consumo è fondamentale in molte applicazioni a cui sono destinati i microcontroller. Mediante l'istruzione SLEEP il PIC entra in modo riposo, garantendo un consumo minimo. Per uscire dal modo riposo possiamo fare un reset, utilizzare il WDT o servirci di un interrupt. Questi due ultimi concetti verranno studiati più dettagliatamente nei capitoli successivi.

Programmazione: variabili ed espressioni

Ritorniamo alla programmazione spiegando i concetti delle variabili e delle espressioni perché, anche se apparentemente sono concetti semplici, in realtà sono un po' più complessi di ciò che sembra. Ogni linguaggio di programmazione lavorerà in modo diverso con le variabili e con le espressioni, ma in questo capitolo parleremo del loro utilizzo generale e vi mostreremo la vastità di questi due concetti.

Definizione di variabile

Abbiamo definito una variabile come un deposito contenente un valore che può essere cambiato lungo il corso del programma. Una definizione più tecnica tratterebbe la variabile come un nome per indicare una o più celle di memoria, dove verranno contenuti in modo temporale i diversi valori generati nel programma. In questo modo tali valori potranno essere recuperati quando sarà necessario.

Dichiarazione di una variabile

Quando vogliamo utilizzare una variabile all'interno di un programma normalmente la dobbiamo dichiarare anticipatamente, indicando al processore che vogliamo utilizzare questa variabile nel programma e il tipo di dato che conterrà (a volte è necessario anche inizializzarla).

Per definire una variabile dobbiamo darle un nome e un tipo, e secondo quest'ultimo il compilatore gli riserverà più o meno spazio nella memoria.

Linguaggio	Dichiarazione	Commenti
C	int b	Si dichiara la variabile intera "b"
Visual Basic	Public tempo As Integer	Variabile pubblica "tempo" che sarà di tipo intero
Assembler	PEPE EQU 0A	Si dichiara la variabile "PEPE" che occuperà la posizione 0A della memoria dei dati
PHP	$intero=2004	Si definisce un intero e gli si assegna il valore 2004
FORTRAN 77	Dimensione A (3,3)	Si dichiara un array di 3x3 posizioni

Dichiarazione di variabili nei diversi linguaggi di programmazione.

Esistono linguaggi di programmazione in cui non è necessario definire le variabili. Ad esempio, nel Visual Basic se non si dichiara una variabile il compilatore le assegnerà un tipo per default, con il suo conseguente spazio di memoria.

Anche se non è necessario, è sempre conveniente definire le variabili in precedenza per ottimizzare l'uso della memoria (non utilizzarne più di quella necessaria) e per evitare errori causati dall'incompatibilità dei dati da gestire o dall'aver nominato la variabile in modo inadeguato.

Esistono norme per l'utilizzo delle variabili che devono essere rispettate, perché altrimenti il compilatore darà errore, e dipendono dal linguaggio di programmazione utilizzato. Esisteranno quindi nei diversi linguaggi differenze nella lunghezza del nome della variabile, il tipo di carattere che accettano nel nome, la sensibilità all'utilizzo di maiuscole e minuscole, ecc. In generale, esistono una serie di regole al momento di nominare e utilizzare le variabili e alcune di esse sono consigliabili, mentre altre sono obbligatorie.

Il nome di una variabile deve essere chiaro e descrittivo, deve far riferimento alla sua funzione all'interno del programma
Non possono far parte del nome spazi bianchi o caratteri speciali
Non si devono utilizzare parole riservate del linguaggio stesso per nominare una variabile
Utilizzare sempre maiuscole o sempre minuscole per dare nomi alle variabili, dato che alcuni linguaggi sono sensibili a questa differenza
Non utilizzare all'interno di un programma variabili con lo stesso nome per usi diversi
Indicare la funzione di ogni variabile con un commento
Dichiarare le variabili al livello a cui si utilizzeranno: classe, metodo, procedimento, modulo...

Norme comuni per l'utilizzo delle variabili.

TIPO DI DATO	RANGE 1	DIMENSIONE (byte)
char	-128...127	1
signed char	-128...127	1
unsigned char	0...255	1
byte	0...255	1
int, integer	-32768...32767	2
unsigned int	0...65536	2
signed short	-32768...32767	2
unsigned short	0...65536	2
long	-2147483648...2147483647	4
unsigned long	0...4294967295	4
float	1.40129E-45...3.402823E+38	4
double	4.94065E-324...1.7976931E+308	8
long double	1.2E-4932..	19
boolean	TRUE-FALSE	2
date	Data/ora	8
currency	Valori tipo valuta	
object	Riferimento a un oggetto	4
string	Catene di caratteri	1 per carattere

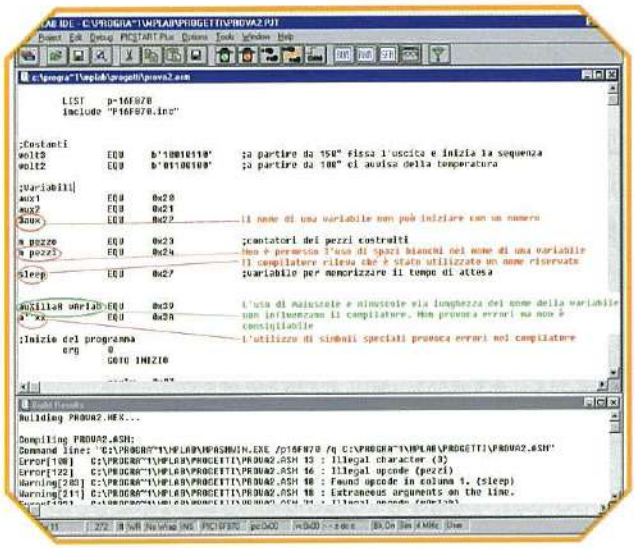
Classificazione dei tipi di dati.

Tipi di dati

In base ai dati che contengono, le variabili possono essere di molti tipi. Una variabile che contenga i caratteri "CIAO" non può essere considerata uguale a un'altra che contenga il numero "3,14159". Il numero di tipi disponibili dipenderà dal linguaggio di programmazione, e normalmente sono i linguaggi di alto livello quelli che contemplano una maggior gamma di variabili.

Nella tabella della figura è possibile vedere una classificazione dei tipi di dati che possiamo trovare, oltre al range dei valori che possono contenere e alla dimensione della memoria associata a ogni tipo. Come si può supporre, tutte queste caratteristiche variano in base al linguaggio di programmazione.

Oltre ai tipi di dati già visti, alcuni linguaggi possiedono quelli che sono chiamati array o matrici di dati, ovvero insiemi di variabili dei tipi visti, posizionati in modo sequenziale nel-



Errori che rileva il compilatore dichiarando le variabili in linguaggio assembler.

```
typedef struct {
        NOME string (100);
        NOTE int(8);
        RIPETITORE boolean;
}CLASSE;
```

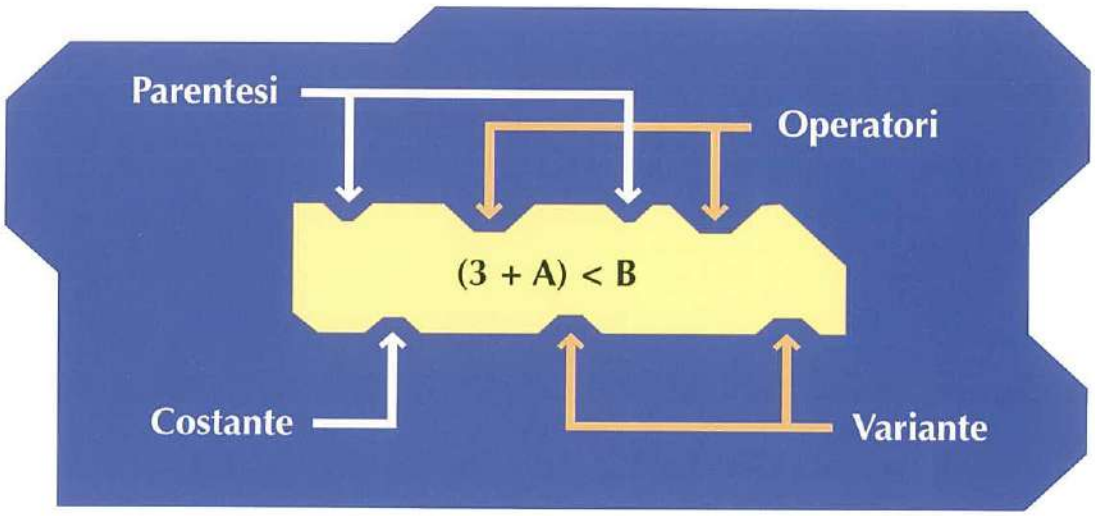
Esempio di una struttura di dati.

la memoria. Quando le variabili di diverso tipo hanno qualcosa in comune sono trattate come un solo dato; alcuni linguaggi permettono la creazione di strutture. Le strutture sono raggruppamenti di variabili in relazione fra loro. Nella figura possiamo vedere un esempio di struttura.

Quando due o più variabili di diverso tipo si trovano nella stessa operazione o espressione, alcuni linguaggi realizzano automaticamente una conversione dei tipi di dati cambiandoli tutti al tipo più restrittivo, invece altri linguaggi in questa situazione generano un errore al momento della compilazione.

Ambito di una variabile

L'ambito di una variabile è la zona del programma in cui questa variabile è visibile e uti-



Esempio di un'espressione e parti che la compongono.

lizzabile. Possiamo quindi fare una distinzione tra variabili globali (pubbliche) o locali (private). Le variabili globali possono essere utilizzate in qualsiasi punto del programma, inclusi tutti i procedimenti e le funzioni, mentre quelle locali vengono utilizzate esclusivamente all'interno del procedimento o funzione in cui sono state create.

Una variabile locale si può creare e inizializzare entrando nel procedimento e viene distrutta all'uscita. Per evitare questo e fare in modo che la variabile si inizializzi solamente la prima volta e poi conservi il suo valore per la chiamata successiva al procedimento, è necessario dichiararla "static".

Operatore	Significato	Tipo di operando	Tipo di risultato
+	Somma	Intero o reale	Intero o reale
-	Sottrazione	Intero o reale	Intero o reale
*, x	Moltiplicazione	Intero o reale	Intero o reale
/, ÷	Divisione	Intero o reale	Reale
**, ^	Esponenziale	Intero o reale	Intero o reale
Div	Quoziente della divisione	Intero o reale	Intero
Mod	Resto della divisione	Intero o reale	Intero
Abs	Valore assoluto	Intero o reale	Intero o reale

Operatori aritmetici più comuni.

a	b	NOT a	a AND b	a OR b
Falso	Falso	Vero	Falso	Falso
Falso	Vero	Vero	Falso	Vero
Vero	Falso	Falso	Falso	Vero
Vero	Vero	Falso	Vero	Vero
		Valore contrario a quello dell'operando	Solo se i due operandi sono veri lo è il risultato	Se uno degli operandi è vero anche il risultato lo è

Tabella della verità del funzionamento dei principali operatori logici.

Definizione di un'espressione

Un'espressione è una composizione ben formata da operandi (variabili e costanti) e operatori (operazioni). Nella figura possiamo vedere un esempio delle parti che possono formare un'espressione semplice: costanti, variabili, operatori, parentesi, ecc.

È necessario conoscere quali simboli sono utilizzati nel linguaggio di programmazione e con quali tipi lavora per poter formare espressioni corrette. Se cercassimo di fare una moltiplicazione scrivendo l'espressione "axb" commetteremmo un errore sintattico, perché l'operatore "x" non è stato implementato nel linguaggio (invece di "x" il linguaggio utilizza "*"). Un errore di tipo sarebbe cercare di sommare "4+False", trattandosi di tipi diversi.

Operatore	Significato
<	Minore di
>	Maggiore di
=, ==	Uguale a
<=	Minore o uguale a
>=	Maggiore o uguale a
<>, !=, /=	Diverso da

Operatori relazionali.

FORTRAN	PASCAL	C
**	*, /, div, mod	Postfisso ++, --
*, /	+, -	Prefisso ++, --
+, -		+, - (unario)
		*, /, %
		+, - (binario)

Regole di precedenza degli operatori aritmetici in funzione del linguaggio di programmazione utilizzato.

Tipi di espressioni

Le espressioni si possono classificare in due grandi gruppi: aritmetiche e logiche. Le operazioni aritmetiche sono simili a quelle matematiche, lavorano con tipi numerici e i simboli che utilizzano sono operatori aritmetici. Si cerca di ottenere un risultato finale che determinerà il valore dell'espressione. Le espressioni logiche, anche dette espressioni booleane, sono composte da costanti e variabili logiche, da operatori logici e da operatori relazionali.

Gli operatori logici più comuni sono "not", "and" e "or". Il primo realizza una negazione del valore su cui opera, il secondo è la congiunzione di due valori (a E b) e l'ultimo la disgiunzione (a O b). Nella figura possiamo vedere come funzionano questi operatori mediante la tabella della verità. In una espressione logica è normale trovare operatori logici insieme a operatori relazionali. Gli operatori relazionali sono mostrati nella tabella allegata.

Valutazione delle espressioni

Per ottenere il risultato di un'espressione è necessario valutarla e, a questo scopo, dob-

A=1, B=2, C=3		
Espressione senza parentesi	Espressione con parentesi	Risultato
A+B*C	A+(B*C)	7
A+B**C/C+A	A+((B**C)/(C+A))	3
A>B AND B<C	(A>B) AND (B<C)	Falso
A+B<=C OR B*A+C<A	((A+B)<=C) OR ((B*(A+C))<A)	Vero
NOT div C/A<B	NOT ((div (C/A))<B)	Falso

Esempio dell'utilizzo degli operatori.

biamo seguire un determinato ordine. Dobbiamo sapere quali operazioni si realizzano per prime, in quanto l'ordine in cui si eseguono influenza il risultato finale.

Nelle operazioni aritmetiche l'ordine in cui si valutano gli operandi è definito dalle regole di precedenza e di associatività. La precedenza indica quale operatore si applicherà per primo a fronte dell'esistenza di diversi operatori in un'espressione e l'associatività indica se la valutazione si realizza da sinistra a destra, da destra a sinistra o combinando entrambi i casi. Sia la precedenza che l'associatività dipendono dal linguaggio di programmazione.

In tutti i linguaggi di programmazione le parentesi hanno la massima priorità. Quindi le operazioni interne alle parentesi saranno da valutare per prime. È consigliabile utilizzare le parentesi per evitare qualsiasi dubbio possibile quando utilizziamo un'espressione in un programma.

Nella tabella a sinistra possiamo osservare le regole di precedenza in diversi linguaggi di programmazione.

Negli operatori logici, l'operatore NOT sarebbe il prioritario, seguito da OR e da AND.

Quando si combinano i diversi operatori viene normalmente data la precedenza a quelli aritmetici, poi a quelli relazionali e, in ultimo, a quelli logici.

Le figure allegate riportano diversi esempi di espressioni e dell'utilizzo di operatori al fine di familiarizzare con i concetti appena esposti.

Gli interrupt

Un interrupt consiste nell'interruzione del programma in corso per realizzare una determinata routine che risolve la causa che ha provocato l'interrupt. Non tutti i microcontroller dispongono di questo dispositivo, però man mano che si capisce il concetto ci si rende conto di quanto sia importante e pratico. Verificheremo che nella maggior parte delle applicazioni sviluppate con il PIC l'utilizzo degli interrupt risulta fondamentale.

Generalità

Un interrupt è un'interruzione nel flusso di controllo del programma principale motivata da una causa speciale.

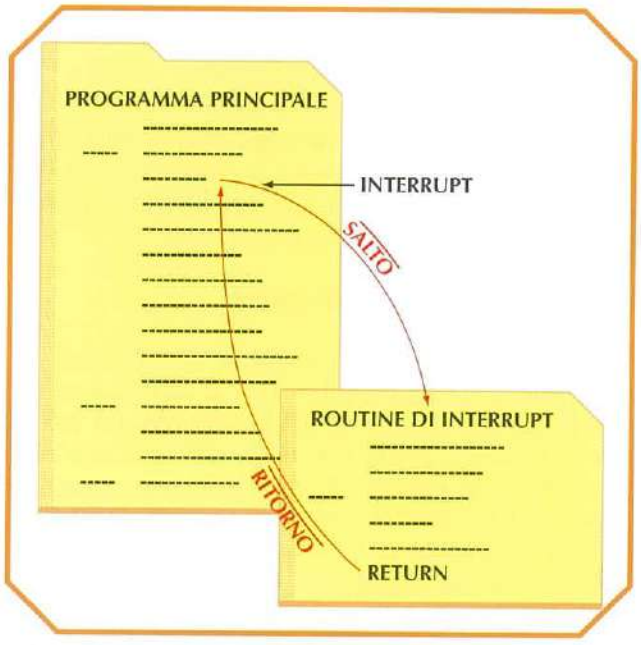
Quando si genera questo evento speciale, se il processore accetta l'interrupt, salva il valore del contatore di programma (PC) nello stack, e lo carica con il Vector di Interrupt a cui corrisponde l'indirizzo 0004h della memoria delle istruzioni. In questa posizione si trova la prima istruzione di una routine dedicata all'interrupt e alla causa che l'ha provocato. Più avanti vedremo che questa istruzione è un'istruzione di salto.

Al termine della routine di interrupt si ritorna al programma principale, nel punto in cui lo si è abbandonato.

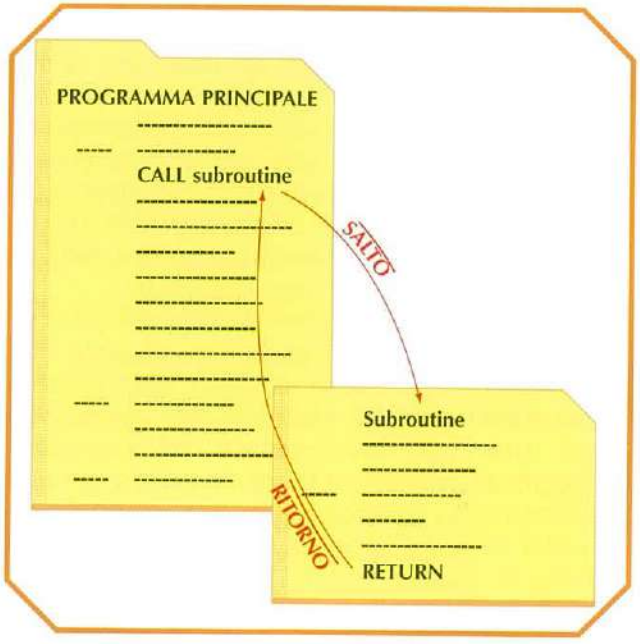
Un interrupt è come una chiamata a subroutine, originato da una causa diversa da un'istruzione di tipo CALL. In entrambi i casi si salva il valore del PC sullo stack, si esegue la routine associata e, terminata l'esecuzione, si ritorna al programma principale, però gli interrupt sono delle rotture asincrone del flusso di programma (non si conosce il momento in cui si generano), mentre le chiamate a subroutine sono sincrone (si sa quando si verificheranno). Nelle figure possiamo vedere le similitudini tra un interrupt e una chiamata a subroutine.

Applicazione dell'interrupt

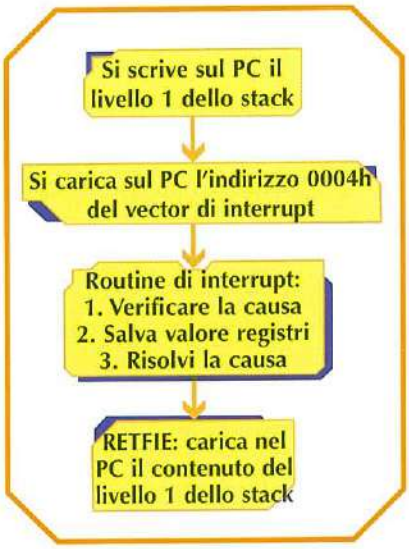
La ragione per cui affermiamo che l'utilizzo degli interrupt risulta fondamentale per un programmatore è che, grazie a questi, è possibile far fronte in modo immediato agli avvenimenti del mondo esterno. Una delle possibili cause, come vedremo successivamente, di ge-



Quando si genera un interrupt si esegue la routine di interrupt e si ritorna al programma.

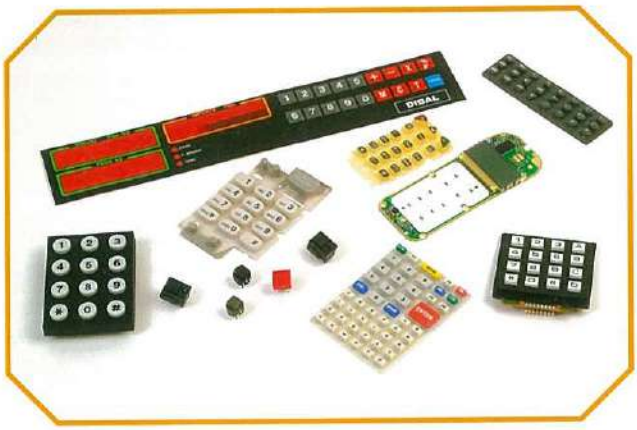


In una chiamata a subroutine lavoriamo in modo sincrono.



Sequenza di operazioni che comporta un interrupt.

Sensori. Quando cambiano il loro stato si produce un interrupt.



Diverse tastiere e pulsanti che si gestiranno utilizzando interrupt.

nerazione di interrupt, è l'applicazione di un livello o di un particolare fronte su uno dei pin del microcontroller. Qualsiasi applicazione che utilizzi un sensore digitale come il sensore di pioggia di un'automobile, il sensore di presenza in un sistema di allarme, il sensore di fumo in un sistema antincendio, un sensore finecorsa, ecc., utilizzerà gli interrupt per comunicare al microprocessore lo stato in cui si trova. Le applicazioni che utilizzano una tastiera o dei pulsanti, come il telecomando di un garage, il telefono cellulare, una cassa del supermercato, ecc., provocano anch'essi, ogni volta che si preme un pulsante, un interrupt che il processore si incarica di gestire.

In qualsiasi progetto potremmo trovare la necessità di interagire con elementi esterni e molti di essi saranno gestiti tramite interrupt. Nelle figure successive sono presentati alcuni esempi di elementi esterni più comuni con cui può lavorare il nostro microcontroller.

Gli elementi esterni sono molto utili per capire le diverse applicazioni che si possono dare agli interrupt, esistono, però, anche delle cause interne al PIC che possono provocare degli interrupt. Immaginate, ad esempio, un'applicazione che, a un certo punto, debba realizzare una funzione specifica, come il calcolo del tempo di possesso del pallone in una partita di pallacanestro. Quando termina il conteggio è necessario attivare immediatamente un avviso acustico. Il PIC, grazie ai suoi temporizzatori interni, realizzerà il conteggio e quando questo termina provocherà un interrupt. Nella routine di interrupt si farà suonare l'avvisatore acustico e si reinizializzeranno i valori del conteggio, indicando che il possesso di palla ha cambiato squadra.

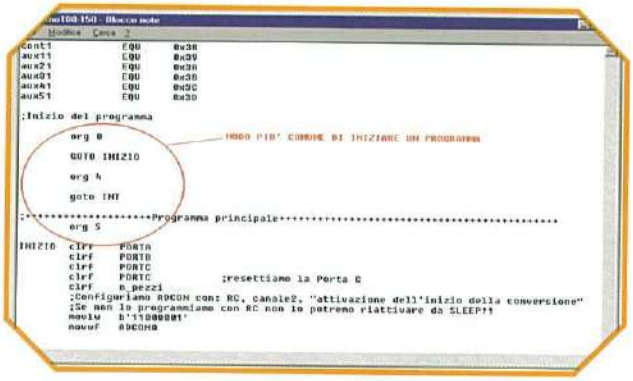
Il Vector di Interrupt

Qualunque sia la causa che provochi l'interrupt, nei microcontroller PIC, si utilizza sempre lo stesso Vector di Interrupt che ha assegnata la posizione di memoria 0004h. Quando si genera un interrupt, il primo lavoro della routine di interrupt è identificare la causa che lo ha provocato, per poter quindi saltare alla subroutine corrispondente, dato che ci saranno tante subroutine quante le possibili cause di interrupt.

Se un programma inizia all'indirizzo 0000h, che corrisponde all'indirizzo del Vector di Re-

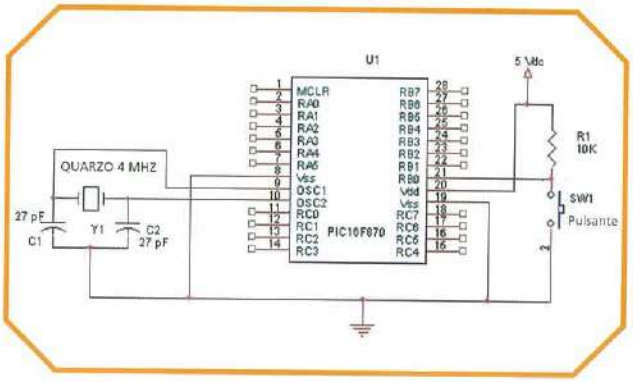
Esempio di interrupt interni nel PIC.

set, e abbiamo detto che il Vector di Interrupt occupa l'indirizzo 0004h, cosa deve fare il nostro programma principale per evitare di sovrascrivere il Vector di Interrupt? Nel nostro programma occorrerà inserire nella posizione 0000h un'istruzione di salto (GOTO) all'indirizzo 0005h, in modo da schivare e non inquinare il dato presente nel Vector di Interrupt. Anche all'indirizzo 0004h dovrà esserci un'istruzione di salto a un altro indirizzo dove ci sia uno spazio libero adatto a ospitare la routine di interrupt. Normalmente si iniziano tutti i programmi nello stesso modo, così come mostrato nell'esempio della figura.



```
cont1           EQU     0x38
aux11           EQU     0x39
aux21           EQU     0x3A
aux01           EQU     0x3B
aux41           EQU     0x3C
aux51           EQU     0x3D
;Inizio del programma
        org 0                   MODO PIU' COMUNE DI INIZIARE UN PROGRAMMA
        GOTO INIZIO
        org 4
        goto INT
;*****************Programma principale*************************************
        org 5
INIZIO  clrf    PORTA
        clrf    PORTB
        clrf    PORTC
        clrf    PORTC           ;resettiamo la Porta C
        clrf    n_pezzi
        ;Configuriamo ADCON con: RC, canale2, "attivazione dell'inizio della conversione"
        ;Se non lo programmiamo con RC non lo potremo riattivare da SLEEP!!
        movlw   B'11000101'
        movwf   ADCON0
```

Tipico modo con cui inizieremo un programma.



Esempio di interrupt per RB0/INT.

Cause di interrupt

La famiglia PIC16F870/1 accetta 14 possibili cause che possono provocare un interrupt. Solo queste richieste avranno il permesso del microcontroller e solo queste cause saranno riconosciute dal PIC come generatrici di interrupt:

Interrupt per cause esterne:

1ª. Attivazione del pin RB0/INT: è la più utilizzata e la più semplice. Per abilitare questo interrupt si imposta a 1 il bit INTE (INTCON<4>). In questo modo, quando l'elemento esterno collegato al pin RB0 genera un fronte di salita – bit INTEDG =(OPTION_REG<6>) impostato a 1– o un fronte di discesa (bit INTEDG posto a 0) si attiva il bit INTF (INTCON<1>), si rileva che si è generato un interrupt. Se il bit di abilitazione globale GIE (INTCON<7>) è abilitato, il processore salterà a eseguire la routine di servizio all'interrupt (ISR).

Nella figura si può vedere un esempio di come possiamo generare un interrupt di questo tipo. Quando si attiva il pulsante si produrrà un fronte di discesa che provocherà un interrupt, sempre se i registri sono stati configurati per accettare un interrupt di questo tipo. Immaginate un microrobot con un sensore finecorsa sulla parte frontale. Quando il microrobot incontra un ostacolo sulla sua traiettoria, si chiederà il contatto del finecorsa e produrrà un fronte di discesa in RB0.

2ª. Cambio di stato di uno dei pin RB7:4 della porta B: quando si produce un cambio di stato su qualcuno dei pin di ingresso RB7, RB6, RB5 o RB4, il flag RBIF si imposterà a 1 (bit 0 del registro INTCON). Questi interrupt possono abilitare o disabilitare attivando o ponendo a 0 il bit RB1E (INTCON<4>) di abilitazione. La principale applicazione per cui questo tipo di interrupt è utilizzato è il possibile collegamento di sensori alla porta. Ad esempio, se abbiamo collegato quattro sensori di presenza per un'applicazione di sicurezza, nel momento in cui uno di essi si atti-

	BIT REGISTRO<POS>	Descrizione
RB0/INT	PORTB<0>	Pin dove si collegherà l'elemento esterno
INTE	INTCON<4>	Bit di abilitazione di questo tipo di interrupt
GIE	INTCON<7>	Bit di abilitazione di interrupt (permette che si possano verificare)
INTEDG	OPTION_REG<6>	Selezioniamo quale fronte sarà la causa di interrupt (a 1 quello di salita, a 0 quello di discesa)
INTF	INTCON<1>	Flag che indica che si è verificato questo tipo di interrupt

Bit che intervengono quando si produce un interrupt tramite RB0.

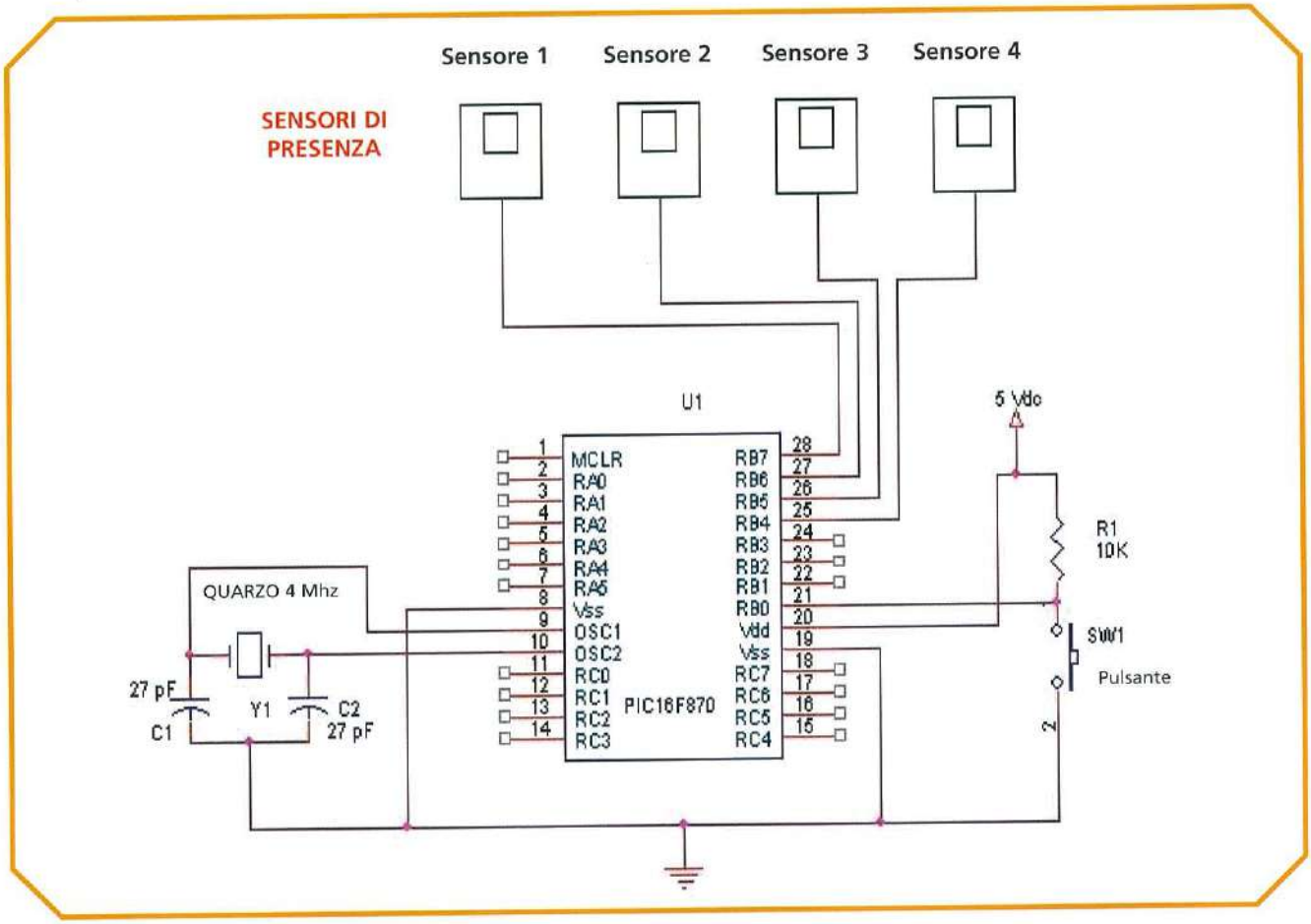
va, cambia lo stato e attiverà il flag, e se gli interrupt saranno stati abilitati, verrà eseguita la routine di gestione dell'interrupt.

Per gli interrupt generati da elementi esterni, come per i precedenti, la latenza dell'interrupt durerà tre o quattro cicli di istruzione. Il tempo esatto di latenza dipenderà dal momento in cui si produce l'interrupt.

Interrupt per cause interne:

3ª. Termine di un'operazione di scrittura nella EEPROM:

In questo caso, quando il processo interno di scrittura nella memoria di programma EEPROM termina, il bit flag EEIF (PIR2<4>) si attiva e se il bit di abilitazione dell'interrupt EEIE (PIE2<4>) è attivato, si produce l'interrupt.



Esempio di un sistema di allarmi che utilizza gli interrupt tramite la porta B.

Gli interrupt (II)

Continueremo con lo studio degli interrupt analizzando le cause che li generano ed entrando nel dettaglio dei registri che sono coinvolti e come funzionano. In questo modo potremo capire la funzionalità e l'importanza di questo dispositivo e saremo pronti per utilizzarlo dopo aver terminato con l'introduzione alla programmazione e al repertorio di istruzioni, e inizieremo a progettare programmi.

Cause di interrupt (interne)

4ª. Overflow del temporizzatore TMR0: il TMR0 (Timer 0) è un temporizzatore interno del PIC. Quando azzera il suo registro (da FFh a 00h) si attiva il bit T0IF (INTCON<2>), bit di flag, e si genera un interrupt se il suo bit di abilitazione è attivato, T0IE (INTCON<5>).

5ª. Overflow del temporizzatore TMR1: è uguale al precedente però con il temporizzatore interno Timer 1. Il bit di flag è TMR1IF (PIR1<0>) e quello di abilitazione TMR1IE (PIE1<0>).

6ª. Overflow del temporizzatore TMR2: il PIC 16F870 dispone di tre temporizzatori: TMR0, TMR1 e TMR2. Quest'ultimo funziona come i precedenti, in modo che quando resetta il valore del conteggio si possa produrre un interrupt. Il bit di flag è TMR2IF (PIR1<1>) e quello di abilitazione TMR2IE (PIE1<1>).

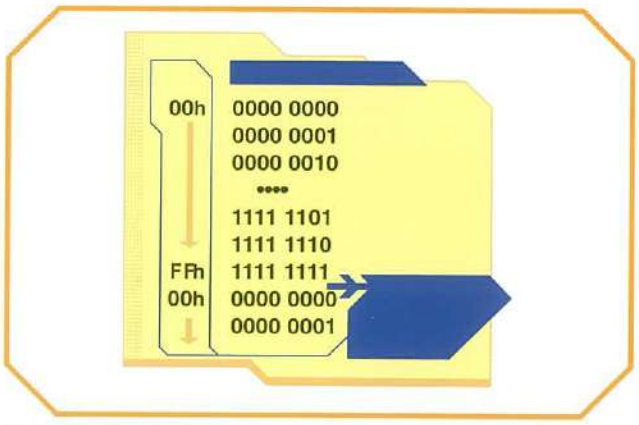
7ª. Capture/Compare/PWM sul modulo CCP1: se il modulo funziona in modo capture, il bit CCP1IF (PIR1<2>) si imposterà a 1 quando si acquisisce il temporizzatore TMR1. In modo comparazione il bit di flag viene impostato a 1 se il valore del TMR1 coincide con un valore preselezionato. Nel modo PWM non si generano interrupt. Il bit di abilitazione di questo interrupt è CCP1IE (PIE1<2>).

8ª. Capture/Compare/PWM sul modulo CCP2: è identico al precedente però con bit differenti, ovvero CCP2IF (PIR2<0>) quello di flag, e CCP2IE (PIE2<0>) quello di abilitazione.

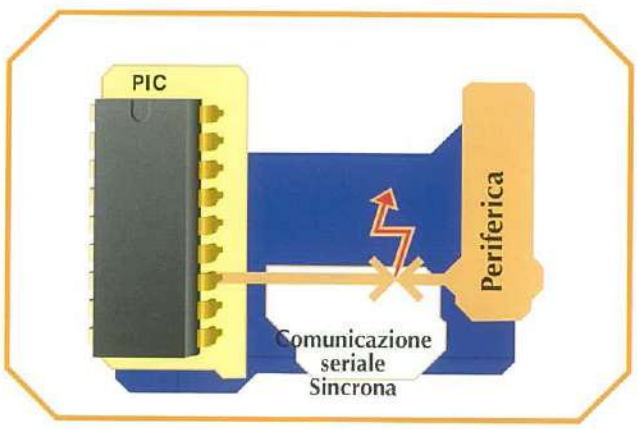
9ª. Trasferimento tramite la porta seriale sincrona: mediante un 1 sul flag SSPIF (PIR1<3>), si indicherà lo stato (attivato) della porta seriale sincrona. Come nel resto degli interrupt, quando il flag passa allo stato segnalato è necessario, mediante software, cancellarlo e riportarlo a zero. Il bit di abilitazione è SSPIE (PIE1<3>).

10ª. Collisione del bus sulla porta seriale sincrona: nella trasmissione in modo sincrono si verifica un interrupt quando esiste una collisione di dati sul bus. Il bit di flag è il BCLIF (PIR2<3>) e il bit di abilitazione è BCLIE (PIE2<3>).

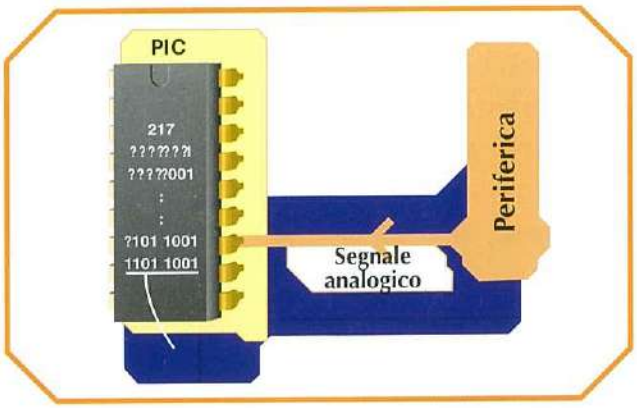
11ª. Termine della trasmissione dell'USART: quando termina una trasmissione di questo tipo, si avvisa il processore mediante un inter-



I temporizzatori provocano un interrupt quando vanno in overflow.



Interrupt per collisione sul bus nella porta seriale sincrona.



Interrupt per fine della conversione analogico/digitale.

rupt. I bit di flag e di abilitazione si possono vedere nella tabella allegata.

12ª. Termine della ricezione dell'USART: uguale al precedente, però quando termina la ricezione dei dati.

13ª. Termine della conversione A/D: anche quando termina un'operazione di conversione di un valore analogico a uno digitale si deve indicare al processore che la procedura è terminata, quindi utilizzeremo un interrupt.

14ª. Termine del trasferimento sulla porta parallela slave: un altro modo di trasmissione dei dati del processore con i dispositivi periferici che avviserà del termine mediante interrupt.

Il nostro PIC da 28 pin non dispone di tutti gli interrupt studiati, dato che non possiede i dispositivi della porta parallela slave, della porta seriale sincrona né del modulo CCP2. Il

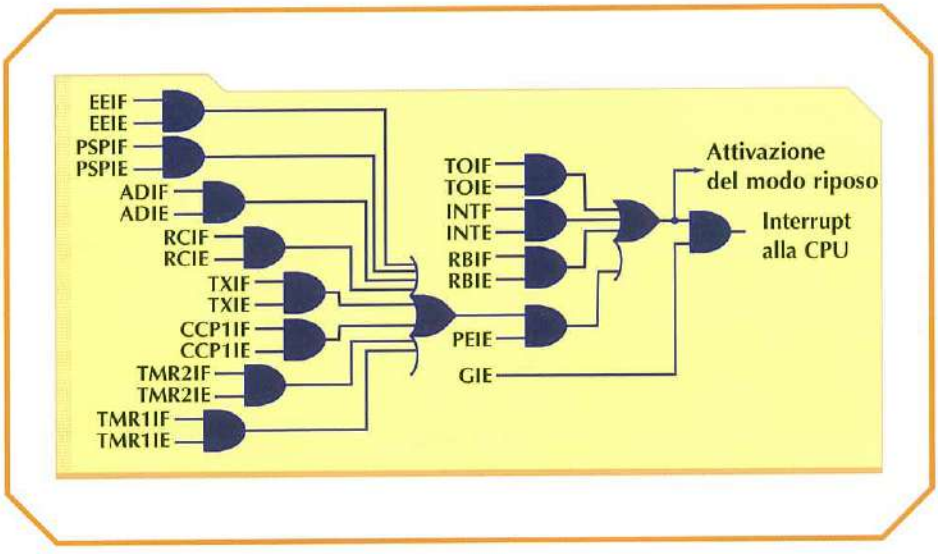
Interrupt	Flag	Abilitazione
Pin RB0/INT	INTF	INTE
Cambio di stato in RB7:RB4	RB1F	RB1E
Scrittura su EEPROM	EEIF	EEIE
Temporizzatore TMR0	T0IF	T0IE
Temporizzatore TMR1	TMR1IF	TMR1IE
Temporizzatore TMR2	TMR2IF	TMR2IE
Modulo CCP1	CCP1IF	CCP1IE
Modulo CCP2	CCP2IF	CCP2IE
Porta seriale sincrona (SSP)	SSPIF	SSPIE
Collisione del bus in SSP	BCLIF	BCLIE
Trasmissione USART	TXIF	TXIE
Ricezione USART	RCIF	RCIE
Conversione A/D	ADIF	ADIE
Porta parallela slave	PSPIF	PSPIE

Bit di flag e di abilitazione di tutti i possibili interrupt.

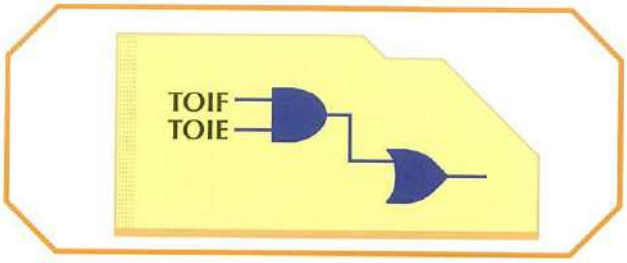
nostro PIC gestisce dieci cause possibili di interrupt.

Gestione degli interrupt

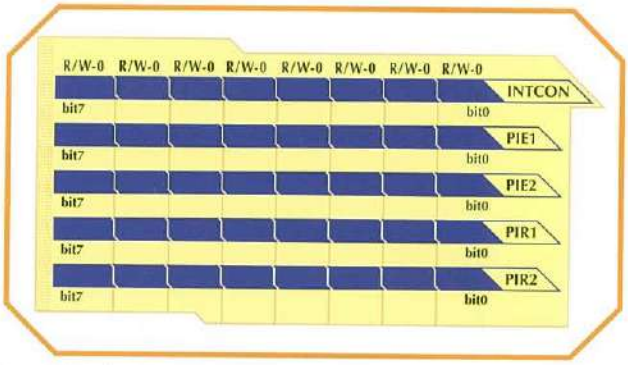
Ogni causa di interrupt è controllata da due linee o segnali. Uno funziona come flag e indica se si è verificato o no l'evento, e l'altro è l'abilitazione o la disabilitazione dell'interrupt in sé.



Logica di controllo per l'attivazione della linea di richiesta di interrupt.



Schema di controllo della generazione di un interrupt.



Registri che intervengono nella gestione degli interrupt.

Nell'esempio della figura vediamo come, affinché l'overflow del TMR0 generi un interrupt, sia necessario che i due segnali di ingresso alla porta AND abbiano livello alto.

Il valore che si applica ai segnali di ingresso del circuito di controllo degli interrupt deriva dallo stato dei bit dei registri INTCON, PIR1, PIR2, PIE1 e PIE2.

Il bit GIE di attivazione globale degli interrupt, situato nel registro INTCON, si cancella in modo automatico quando viene riconosciuto un interrupt, per evitare che se ne generi un altro mentre si sta risolvendo quello iniziale. Al ritorno dall'interrupt, il bit GIE si attiva nuovamente.

Esiste un altro bit di abilitazione ristretto ad alcuni interrupt di determinate periferiche che è il PEIE (INTCON<6>).

Dato che gli interrupt possono essere generati da cause diverse, il software iniziale della routine di interrupt deve iniziare verificando quale è stata la causa che lo ha provocato. Una volta identificato il flag, bisogna cancellarlo via software per prevenire falsi interrupt.

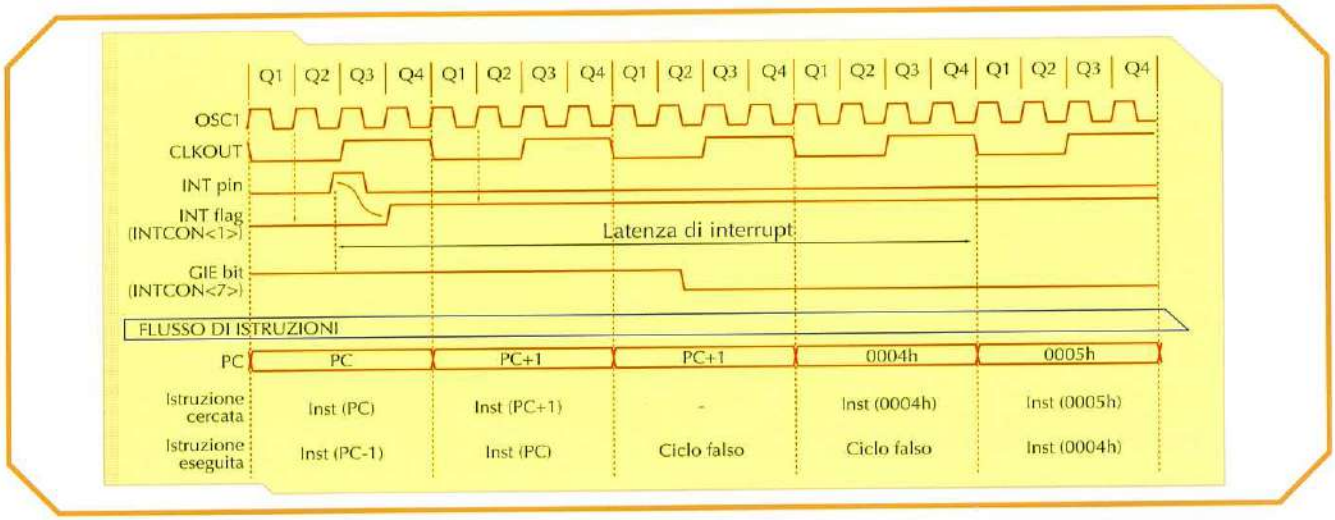
La segnalazione di un interrupt esterno prodotta dall'attivazione di un pin, deve essere

Organigramma di sviluppo di un interrupt che comprende il bit GIE.

mantenuta con il livello attivo almeno 3 o 4 cicli di istruzione, a seconda di quando si produce. Con il bit corrispondente del registro OPTION si programma che sia sensibile al fronte di salita o a quello di discesa. Nel grafico della figura possiamo verificare che il tempo di latenza di interrupt corrisponde a quanto detto in precedenza. Il tempo di latenza di interrupt è definito come il tempo che intercorre da quando si verifica l'evento di interrupt fino a quando inizia l'esecuzione dell'istruzione situata all'indirizzo 0004h. Per interrupt esterni il tempo di latenza sarà di 3 o 4 cicli di istruzione e per quelli interni è di 3 cicli di istruzione.

Salvare il contesto

Quando si genera un interrupt automaticamente, si memorizza il valore del contatore di programma nello stack, però cosa succede con il resto dei registri? Quando saltiamo alla routine di servizio di interrupt, dobbiamo salvare i registri con cui stiamo lavorando nel pro-



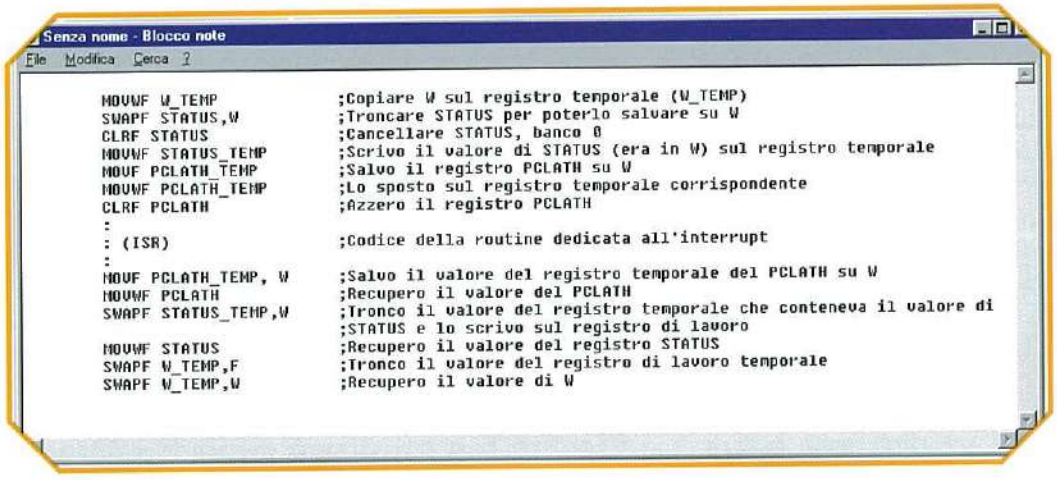
Funzionamento interno quando si produce un interrupt esterno.

gramma principale per non perdere i dati contenuti. Mediante l'utilizzo di variabili possiamo memorizzare i valori di questi registri per poterli recuperare al ritorno dalla routine di servizio all'interrupt. Nella figura possiamo vedere un esempio di come scrivere il contenuto del registro di lavoro W, del registro di stato STATUS, e il valore del registro PCLATH in variabili temporali, e come recuperare questi valori dopo aver terminato l'esecuzione della routine di servizio all'interrupt.

Un interrupt non deve provocare la perdita di dati gestiti dal programma principale. Il progettista dovrà tener conto di questo e memorizzare i valori dei registri con cui sta lavorando su variabili temporali, così come mostra l'esempio.

Conclusioni

Abbiamo terminato lo studio teorico di uno dei principali dispositivi del nostro PIC. Sappiamo cosa sono gli interrupt, che applicazioni possono avere, quali sono le cause che li possono provocare e come influenzano il processore. Sappiamo anche che ogni interrupt ha associati due bit, uno di abilitazione (IE, Interrupt Enable) e l'altro di flag (IF, Interrupt Flag), e che tutti questi dipendono da un bit di abilitazione generale. Quando impareremo il repertorio di istruzioni, faremo pratica con applicazioni reali e vedremo come gestire i registri per diversi tipi di interrupt. Mediante diversi programmi di esempio potremo applicare tutti i concetti imparati, affinando così queste conoscenze.



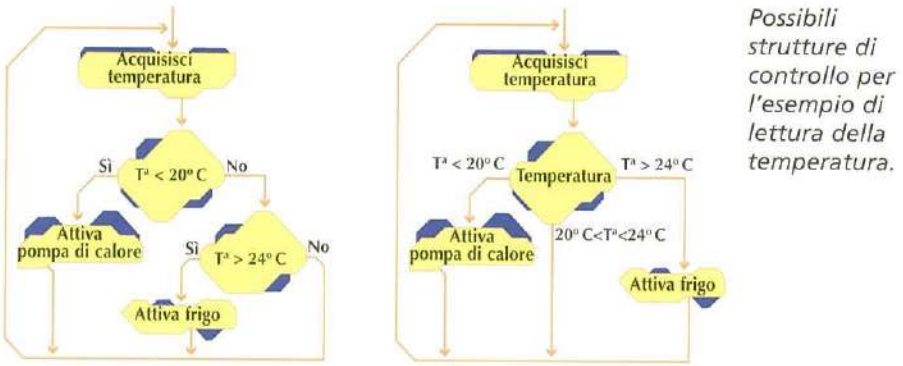

```
MOVWF W_TEMP            ;Copiare W sul registro temporale (W_TEMP)
SWAPF STATUS,W          ;Troncare STATUS per poterlo salvare su W
CLRF STATUS             ;Cancellare STATUS, banco 0
MOVWF STATUS_TEMP       ;Scrivo il valore di STATUS (era in W) sul registro temporale
MOVF PCLATH_TEMP        ;Salvo il registro PCLATH su W
MOVWF PCLATH_TEMP       ;Lo sposto sul registro temporale corrispondente
CLRF PCLATH             ;Azzero il registro PCLATH
:
: (ISR)                 ;Codice della routine dedicata all'interrupt
:
MOVF PCLATH_TEMP, W     ;Salvo il valore del registro temporale del PCLATH su W
MOVWF PCLATH            ;Recupero il valore del PCLATH
SWAPF STATUS_TEMP,W     ;Tronco il valore del registro temporale che conteneva il valore di
                        ;STATUS e lo scrivo sul registro di lavoro
MOVWF STATUS            ;Recupero il valore del registro STATUS
SWAPF W_TEMP,F          ;Tronco il valore del registro di lavoro temporale
SWAPF W_TEMP,W          ;Recupero il valore di W
```

È necessario salvare il contesto del programma prima di eseguire la routine di servizio dell'interrupt e recuperarlo in seguito, terminata l'esecuzione della routine.

Programmazione

Continuiamo con l'introduzione alla programmazione che stiamo alternando alla conoscenza del PIC16F870, affrontando, in questo numero, le strutture di programmazione, i salti condizionali e incondizionali, i procedimenti e le funzioni.

Strutture di programmazione



Possibili strutture di controllo per l'esempio di lettura della temperatura.

```
IF x=0 THEN GOTO ciao
...
...
ciao: ...
```

```
IF x=0 THEN GOTO ciao ELSE GOTO addio
...
...
ciao: ...
...
addio: ...
```

```
IF x=0 THEN
{...}
ELSE
{...}
```

```
CASE x OF
1: {...}
2: {...}
3: {...}
4: {...}
END
```

```
ESPERA: BTFSC   STATUS,Z
        CALL    TX2DAT
        GOTO    ESPERA
TX2DAT: ...
```

Diversi tipi di strutture condizionali.

Le strutture di programmazione, anche chiamate strutture di controllo, sono gli elementi che permettono di eseguire nel programma alcune azioni oppure altre, secondo determinate condizioni. L'utilizzo di queste strutture provoca la non linearità dei programmi che, pertanto, non saranno eseguiti sempre nello stesso modo.

Immaginate un programma che acquisisce la temperatura esterna. In funzione della temperatura attiverà la caldaia, oppure non farà nulla, o attiverà l'aria condizionata. Il modo più logico di realizzare questo programma è attraverso una struttura di controllo che potrà assumere diverse forme, come mostrato nella figura.

Le strutture di controllo si possono classificare in due grandi gruppi:

Strutture condizionali

Le strutture condizionali o prese di decisione indirizzano il programma verso un percorso oppure un altro in funzione delle differenti opzioni. Se si compie una serie di condizioni, il programma eseguirà alcune determinate istruzioni.

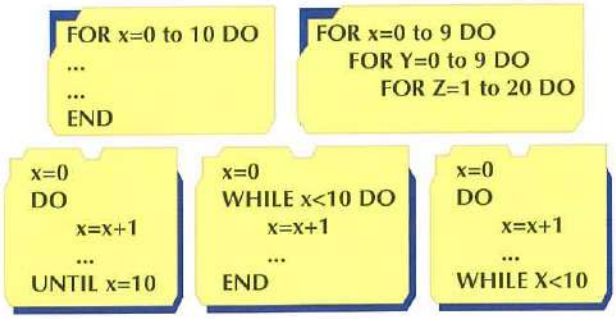
In base al linguaggio di programmazione utilizzato esisteranno più o meno tipi di strutture. Le più utilizzate sono le "IF...THEN..." e tutte le loro varianti (IF...THEN...ELSE, ELSEIF...) in cui, a seconda se una condizione sia vero o falsa, viene eseguita una determinata azione. Possiamo anche trovare strutture condizionali del tipo "CASE" o "SWITCH", dove le espressioni possono avere valori differenti da vero o falso.

In assembler, le strutture di questo tipo sono i salti condizionali che mantengono la filosofia di IF, dato che rispondono a: "Se si compie questa condizione salta la linea successiva".

Strutture cicliche

Le strutture cicliche o cicli permettono di ripetere lo stesso processo molte volte, senza dover scrivere la stessa linea o linee di codice più volte. Ve ne sono alcune che eseguono un insieme di istruzioni un numero di volte specificato (FOR...DO) e altre che eseguono queste istruzioni fino a o mentre si compie una condizione (WHILE...DO, DO...UNTIL).

È possibile annidare i cicli (mettere un ciclo



Esempi di strutture cicliche.

SALTO CONDIZIONALE	DECFSZ registro, spostamento	Decrementa il registro e salta se è 0
	INCFSZ registro, spostamento	Incrementa il registro e salta se è 0
	BTFSC bit, spostamento	Testa il bit e salta se è 0
	BTFSS bit, spostamento	Testa il bit e salta se è 1
SALTO INCONDIZIONALE	GOTO etichetta	Salta all'etichetta

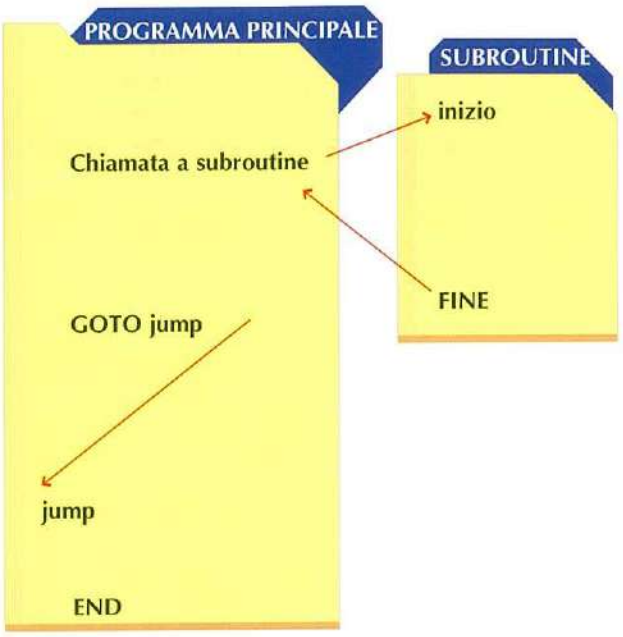
Istruzioni utilizzate in assembler per i salti.

all'interno di un altro) per moltiplicare le interazioni. A seconda del linguaggio di programmazione, avremo a disposizione più o meno istruzioni cicliche.

Nel caso non ci fossero, come succede nell'assembler, le potremo costruire a partire da strutture condizionali e l'istruzione "GOTO".

Salti condizionali e incondizionali

Un salto condizionale è quello che si verifica unicamente se, e solo se, si compie una determinata condizione. Nel linguaggio assembler sono quattro le istruzioni che realizzano un salto condizionale.



Differenze tra chiamata a una subroutine e l'istruzione GOTO.

I salti incondizionali non rispondono a nessuna condizione, si verificano sempre quando appare l'istruzione corrispondente al salto. Nel linguaggio assembler esiste l'istruzione GOTO che genera un salto incondizionale.

Una chiamata a subroutine si può considerare un salto incondizionale, dato che, se si genera, il programma abbandona l'esecuzione del programma principale e salta all'esecuzione della subroutine. La differenza tra l'istruzione GOTO e una chiamata a una subroutine sta nel fatto che, dopo aver eseguito la subroutine, si ritorna al programma principale, mentre con GOTO no.

Quanto più alto è il livello del linguaggio di programmazione, maggiore è la varietà di istruzioni di salto a disposizione.

Procedimenti

I procedimenti sono frammenti di codice che realizzano un determinato compito e che sono dichiarati una sola volta, infatti ogni volta che si vorranno utilizzare sarà sufficiente indicare il loro nome.

Per dichiarare un procedimento, nei linguaggi di programmazione che lo permettono, di solito si utilizza la parola riservata PROCEDURE, seguita dal nome che desideriamo dare al procedimento.

Se lo consideriamo opportuno, possiamo trasmettere variabili a un procedimento tramite il programma principale. A questo scopo dovremo solamente definire, nella dichiarazione di procedimento, una lista con le variabili che desideriamo trasferire. Questo si potrà fare per valore o per riferimento, nel primo caso le variazioni che si producono al termine del procedimento si perderanno; nel secondo

```
programma esempio;
var
base:integer;

procedure   somma(uno, due:integer);

            var
            sol:integer;

            begin
            sol=uno+due;
            writeln ('La somma è :',sol);
            end;

begin
clrscr;
base :=1;
somma(14,12);
somma(25,base);
end.
```

Esempio di programma con chiamata a procedimento in un linguaggio di alto livello.

```
Dichiarazione variabili
  INTERO: N, somma
  fine dichiarazione variabili

inizio

da N=1 fino a N=200 fare
  Somma <- Somma_N_Naturali(N);
  mostra sul display ('La somma dei ',N,'
  primi naturali è ','Somma)
  fine da

fine

funzione Somma_N_Naturali(INTERO N) :
INTERO
  variabili
  INTERO: Somma,i

  Somma <- 0
  da i<-1 fino a i=N fare
    Somma <- Somma+i

  Risultato <- Somma
fine funzione
```

Dichiarazione di una funzione.

```
funzione NOME (arg1,...,argN) : TIPO
  variabili
  ..... {si dichiarano}

    azione 1
    .....
    azione N
  Risultato <- Valore
Fine funzione
```

Esempio di programma che utilizza una funzione.

si trasmette un indirizzo e, nel caso si verifichino delle variazioni nel contenuto dello stesso, queste saranno permanenti, anche una volta terminata l'esecuzione del procedimento.

Nell'esempio della figura vediamo che nella dichiarazione del procedimento si definiscono due variabili: uno e due, variabili intere. Il procedimento ha la struttura di un programma normale e realizza una o più funzioni (nel nostro caso somma due numeri e presenta il risultato sul display). Nel programma principale si può ripetere la chiamata al procedimento scrivendo il suo nome e trasferendo i parametri con cui desideriamo che lavori.

Funzioni

Le funzioni sono molto simili ai procedimenti, con la differenza che restituiscono un valore al termine dell'esecuzione. Il tipo di questo valore si dichiara al momento di dichiarare la funzione. Quindi, una funzione si definirà nel modo indicato dalla figura.

TIPO è il tipo di dato che restituirà la funzione, NOME è il nome dato alla funzione e arg1...argN sono i parametri che passeremo alla funzione. Al termine di tutte le azioni la funzione restituirà un risultato che sarà definito quando chiameremo la funzione.

Per chiamare una funzione dovremo aver definito nelle nostre dichiarazioni di variabili una variabile dello STESSO tipo di quello restituito dalla funzione. Quindi, dovremo assegnare a questa variabile ciò che ci restituisce la funzione nel seguente modo: Variabile <- Nome_Funzione (arg1,...,argN).

La struttura commentata, che si può vedere nell'esempio della figura, è quella corrispondente ai linguaggi di alto livello, e si semplifi-

```
            org 0x00
            goto Inizio
            org 0x05

tabella:    addwf   PCL,F
            retlw   b'00111111'  ;Dig 0
            retlw   b'00000110'  ;Dig 1
            retlw   b'01011011'  ;Dig 2
            retlw   b'01001111'  ;Dig 3
            retlw   b'01100110'  ;Dig 4
            retlw   b'01101101'  ;Dig 5

Delay20:    bcf     INTCON.T0IF  ;Resetta il flag
                                  di overflow
            movlw   0xb1         ;Complemento hex.
                                  di 78
            movwf   TMR0         ;carica il TMR0
Del20_1:    clrwdt               ;Aggiorna il WDT
            btfss   INTCON,T0IF  ;Overflow del
                                  TMR0 ??
            goto    Delay_20_ms_1;Ancora no
            return

Inizio:     ...
            ...
            ...
            call    tabella
            ...
            ...
            call    Delay20
            ...
            ...
            end
```

Esempio in assembler di utilizzo di funzioni.

Si ottiene una maggior chiarezza del codice.
Si evita la ripetizione inutile di frammenti uguali di codice.
Si dividono le azioni complesse in sottoazioni più semplici.
Si ottiene una maggior modularità e quindi il programma si modifica più facilmente.
Si risparmia memoria utilizzando variabili locali invece di globali.

Vantaggi dell'utilizzo di procedimenti e funzioni.

ca molto quando utilizziamo l'assembler come linguaggio di programmazione. Nella figura allegata possiamo verificare come si dichiara una funzione e si realizza una chiamata in linguaggio assembler.

Recursività e concatenazione

La recursività è la proprietà di una funzione o di un subprogramma di eseguire una chiamata a se stessa. Quindi, all'interno di una funzione, potremo creare una chiamata alla stessa funzione, dato che questa conserverà il valore dell'esecuzione precedente.

Le funzioni o subprogrammi, così come le strutture di controllo, possono essere concatenate, in modo che una funzione sia inclusa all'interno di un'altra, quindi sarà visibile solamente per quest'ultima.

Generalità su procedimenti e funzioni

Molte volte si confondono questi termini e si utilizzano indistintamente per indicare una parte del codice che è fuori, o separata, dal programma principale.

Abbiamo imparato a differenziarle, e le uniche cose che hanno ancora in comune sono i vantaggi offerti dal loro utilizzo.

Ad esempio si risparmia memoria, perché una variabile che si definisce all'interno del procedimento o funzione (locale) avrà senso solamente all'interno di questa e sparirà al termine dell'esecuzione.

Conclusioni

Abbiamo terminato il ripasso della programmazione generale studiando le strutture di controllo, condizionali o cicliche, i salti condizionali e incondizionali, i procedimenti e le funzioni. Questi concetti vi saranno utili per qualsiasi linguaggio di programmazione che utilizzerete e costituiscono la base di un buon programmatore. Analizzeremo e studieremo il repertorio di istruzioni del nostro microcontroller, in modo che, una volta conosciute le 35 istruzioni che possiede, le caratteristiche e i dispositivi del PIC, potremo iniziare a sviluppare applicazioni con esso.

Il temporizzatore TMR0

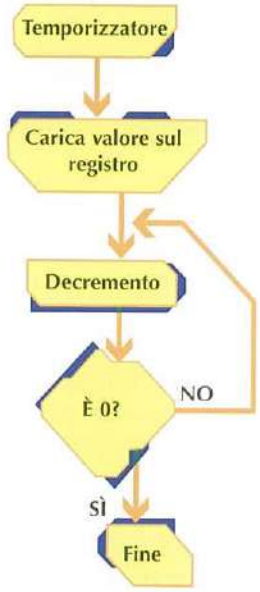
Il PIC16F870 dispone di tre temporizzatori: il Timer 0, il Timer 1 e il Timer 2. I temporizzatori sono dispositivi molto utilizzati, dato che si possono usare come contatori o temporizzatori. Molte delle applicazioni che realizzeremo necessitano, per il loro funzionamento, di uno o più temporizzatori. Analizzeremo la struttura e il funzionamento del temporizzatore principale, il Timer 0, e impareremo a utilizzarlo.

Il temporizzatore è un contatore che si carica con un valore all'inizio del conteggio del tempo e aumenta o diminuisce a ogni impulso di clock. Quando si supera il valore massimo che può contenere (va in overflow) o arriva a zero, possiede un bit di segnalazione che passa automaticamente a 1 indicando la fine del conteggio del tempo.

I temporizzatori evitano che il processore debba dedicarsi, all'interno del programma principale, a contare il tempo. Utilizzando un contatore il programma principale può continuare a essere eseguito, dato che il contatore ha un funzionamento parallelo.

Oltre a indicare il termine del conteggio con il cambiamento di stato del bit di flag, nel nostro PIC potremo anche provocare un interrupt al processore.

Nella figura è presentato, mediante un semplice organigramma, il funzionamento di un temporizzatore dedicato a misurare il tempo.



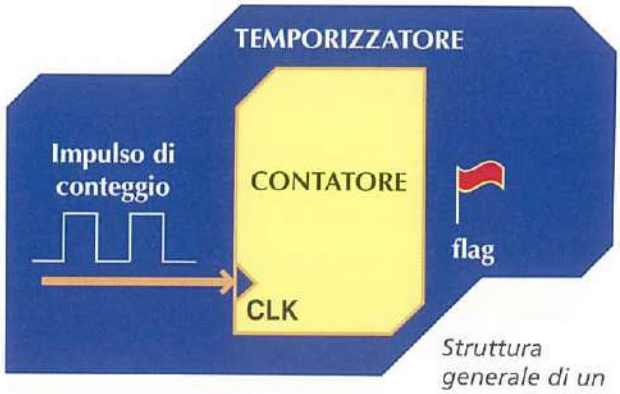
Funzionamento base di un temporizzatore.

Il TMR0 nel PIC16F870

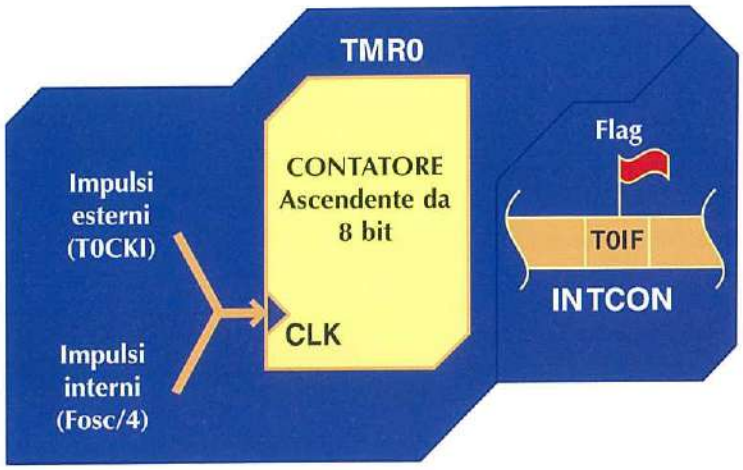
Il TMR0 è un contatore a 8 bit ascendente che, quando raggiunge il suo valore massimo (FFh in esadecimale) passa a 00h con il successivo impulso di clock, e attiva il suo flag (TMR0IF=T0IF=1).

Gli impulsi di clock per l'incremento del contatore si possono aggiungere dall'esterno del microcontroller tramite il pin RA4/T0CKI, oppure dall'oscillatore principale a una frequenza di Fosc/4 (durata del ciclo di un'istruzione).

Il TMR0 si comporta come un registro di utilizzo speciale posizionato all'indirizzo 1 del banco 0 dell'area dei dati, memoria RAM. Può essere letto o scritto, dato che si trova collegato direttamente al bus dei dati. Grazie al fatto che è un contatore ascendente, conviene caricare il temporizzatore con il valore degli impulsi che si desiderano contare, in forma di complemento a 2. Quindi, se vogliamo contare quattro impulsi di clock si carica il TMR0 con il complemento a 2 di 4, il che significa caricarli con -4. In questo modo, fornendo quattro impulsi arriveremo al valore 00h, e si attiverà il flag di segnalazione.



Struttura generale di un temporizzatore.



Struttura del temporizzatore TMR0.

Calcolo del tempo

Il tempo che misura il TMR0, da quando viene caricato con un valore fino a quando va in overflow, dipende dal valore del registro TMR0 e dal valore del Predivisore di Frequenza che si può applicare al temporizzatore. Il Predivisore di Frequenza divide per un determinato valore gli impulsi da applicare al temporizzatore e può essere applicato sia per il TMR0 che per il Watchdog.

Per lavorare con il TMR0 è possibile utilizzare le formule seguenti, nel caso in cui gli impulsi di clock provengano dall'oscillatore interno con un periodo di Tosc:

Tempo = 4 · Tosc · (Valore caricato sul TMR0) · (Range del Predivisore)

Sapendo che il processore funziona a 4 MHz e riceve dall'oscillatore interno gli impulsi di clock, con un prescaler di 16, il valore che dobbiamo caricare sul temporizzatore per ottenere un tempo di 2 ms sarà:

Valore da caricare sul TMR0 = Tempo / (4 · Tosc · Range del predivisore)

Sostituendo con i numeri si ottiene:

Valore da caricare sul TMR0 = $2 \cdot 10^{-3} / (4 \cdot 250 \cdot 10^{-9} \cdot 16) = 125$

Il TMR0 può essere letto in qualsiasi momento per conoscere lo stato del conteggio. Quando si scrive un nuovo valore sul TMR0 per iniziare una nuova temporizzazione, l'incremento dello stesso è ritardato per i primi due successivi cicli di clock.

Struttura interna

Nella figura sono riportati gli elementi che circondano e completano il TMR0. Per gestire lunghi intervalli di tempo è necessario aumentare la durata tra gli impulsi di clock, e questo si ottiene con il Predivisore di Frequenza o prescaler. Esso divide la frequenza degli impulsi entro un determinato range di valori. Si

PS2	PS1	PS0	Divisioni del TMR0	Divisioni del WDT
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

Equivalenza del Divisore di Frequenza rispetto ai bit del registro option.

	4 ⟶ 0000 0100	
Complemento a 1	1111 1011	(Invertire i bit)
Complemento a 2	1111 1100	(Sommare 1)
	-4 ⟶ 1111 1100	

Calcolo del complemento a 2 di 4.

può utilizzare sia per il Timer0 che per il Watchdog, determinando questo mediante un bit del registro OPTION.

Il divisore di frequenza può assumere i seguenti valori: 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256. La selezione del range di questo divisore si realizza tramite i tre bit meno significativi del registro OPTION.

Prima del temporizzatore troviamo un dispositivo che ritarda di due cicli il cui compito è quello di sincronizzare il momento dell'incremento prodotto dal segnale T0CKI con quello prodotto dagli impulsi del clock interno.

Quando non si utilizza il Divisore di Frequenza, l'ingresso del segnale di clock esterno è lo stesso di uscita di questo divisore.

L'interrupt del TMR0

Quando abbiamo studiato gli interrupt abbiamo visto che il Timer 0 poteva generare un interrupt. Quando il TMR0 va in overflow si attiva il flag T0IF, ed esistono due metodi per comunicare al processore questo avvenimento. Il primo consiste nel testare, tramite programma, lo stato del bit T0IF fino a quando troviamo un 1, anche se in questo modo non si ottiene un interrupt immediato, ma solo quando il programma eseguirà il test. Se vogliamo che il microprocessore reagisca immediatamente all'overflow del TMR0, dobbiamo generare un interrupt e, a questo scopo, è necessario attivare i bit di abilitazione che abbiamo già analizzato a suo tempo, ovvero i bit GIE e T0IE (GIE=T0IE=1).

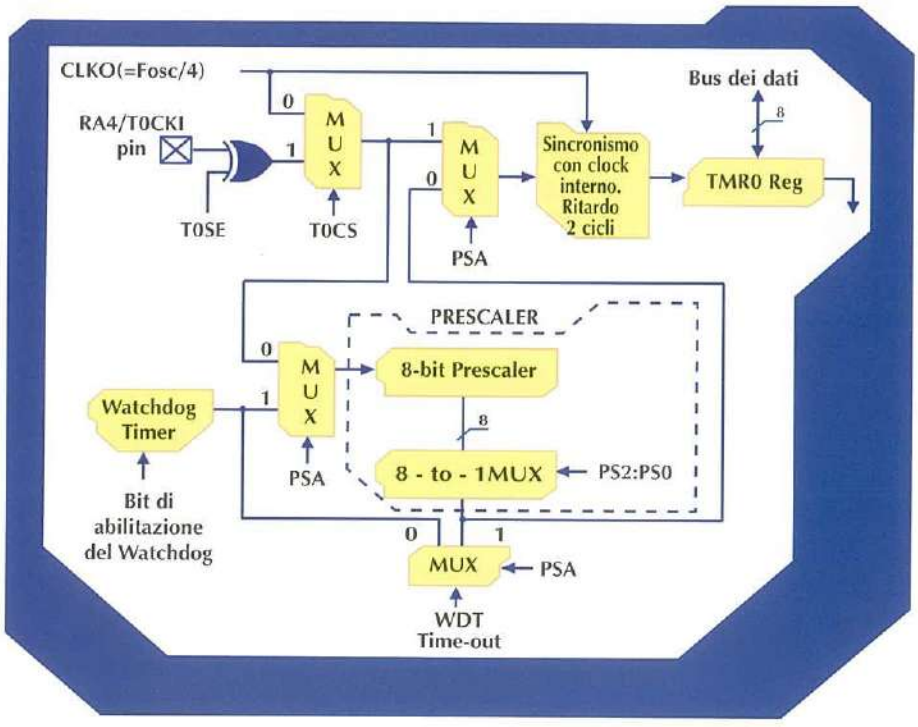
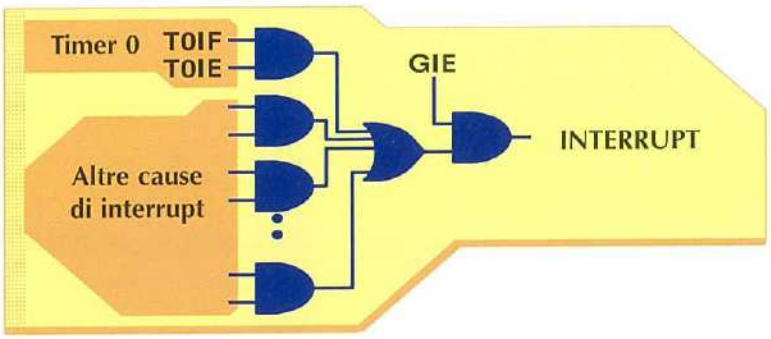


Diagramma a blocchi del Predivisore di Fequenza del TMR0 e del Watchdog.



Logica combinazionale necessaria per far generare un interrupt.

OPTION_REG

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
bit7							bit0

bit7	RBPU: Resistenza Pull-up Porta B
bit6	INTEDG: Fronte attivo interrupt esterno
bit5	T0CS: Tipo di clock per il TMR0 1= Impulso inserito tramite T0CK1 (contatore) 0= Impulso di clock interno Fosc/4 (temporizzatore)
bit4	T0SE: Tipo di fronte in T0CK1 1= Incremento del TMR0 ogni fronte di salita 0= Incremento del TMR0 ogni fronte di discesa
bit3	PSA: Assegnazione del Divisore di Frequenza 1= Divisore assegnato al WDT 0= Divisore assegnato al Timer 0
bit2-0	PS2:PS0: Valore con il quale funziona il divisore di frequenza

Distribuzione dei bit del registro OPTION.

Nella figura possiamo vedere la logica richiesta per fare in modo che si produca un interrupt quando va in overflow il TMR0.

Il registro OPTION

Il registro delle opzioni OPTION ha come compito principale la regolazione del comportamento del temporizzatore TMR0.

Il bit T0CS (Timer 0 External Clock Edge Select) è il bit di controllo che seleziona nel multiplexer la provenienza degli impulsi di clock, che possono arrivare dall'oscillatore interno (Fosc/4) o da quello esterno tramite il pin T0CK1.

Il bit T0SE (Timer 0 Clock Source Select) seleziona il tipo di fronte che provoca il conteggio del temporizzatore. Quindi, se il bit T0SE=1, il fronte attivo è quello discendente e se T0SE=0, è quello ascendente.

Il bit PSA del registro OPTION serve per assegnare il Divisore di Frequenza al TMR0 (PSA=0) o al Watchdog (PSA=1).

I tre bit meno significativi (PS2:PS0) selezionano il range con cui il Divisore di Frequenza divide gli impulsi applicati.

Contatore

Non dimentichiamo che possiamo utilizzare il TMR0, oltre che come temporizzatore, come un utile contatore. Potrà contare il numero di avvenimenti esterni inserendo impulsi dall'esterno tramite il pin T0CK1.

Conclusioni

Continueremo lavorando con gli altri due temporizzatori del microcontroller e studieremo le piccole differenze che li distinguono. I temporizzatori sono fondamentali in qualsiasi progetto o applicazione, e data la loro importanza, dobbiamo aver molto chiaro il loro funzionamento e le possibili applicazioni. Faremo pratica con alcuni esempi dopo aver imparato il repertorio delle istruzioni del nostro PIC.

Temporizzatori TMR1 e TMR2

Il PIC16F870 dispone di tre temporizzatori: il TMR0, il TMR1 e il TMR2. Abbiamo visto come funziona il primo di questi, ma dobbiamo conoscere e capire anche i restanti. Ogni temporizzatore è diverso, anche se il concetto generale è comune per tutti e tre. Iniziamo ad analizzare il TMR1, passando poi all'analisi del TMR2.

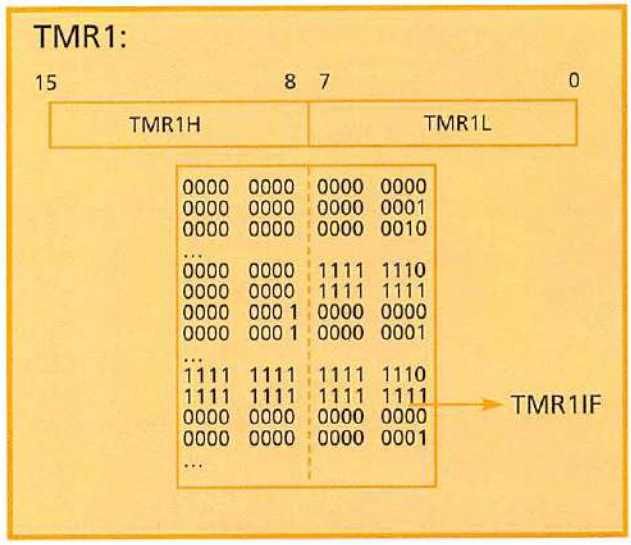
Il temporizzatore TMR1

Il TMR1 è un temporizzatore/contatore a 16 bit. Il valore del conteggio è contenuto in due registri da 8 bit concatenati. I due registri sono TMR1H, che contiene gli 8 bit più significativi, e TMR1L, che contiene gli 8 bit meno significativi. Essendo un registro a 16 bit, il contatore evolve da 0000h fino a FFFFh, istante in cui si attiva il flag di overflow TMR1IF, e ritorna al valore di conteggio iniziale.

Nel momento dell'overflow è possibile anche generare un interrupt, e per farloè necessario aver abilitato sia i bit di abilitazione globale che il bit di abilitazione specifico del Timer (GIE = PEIE = TMR1IE = 1).

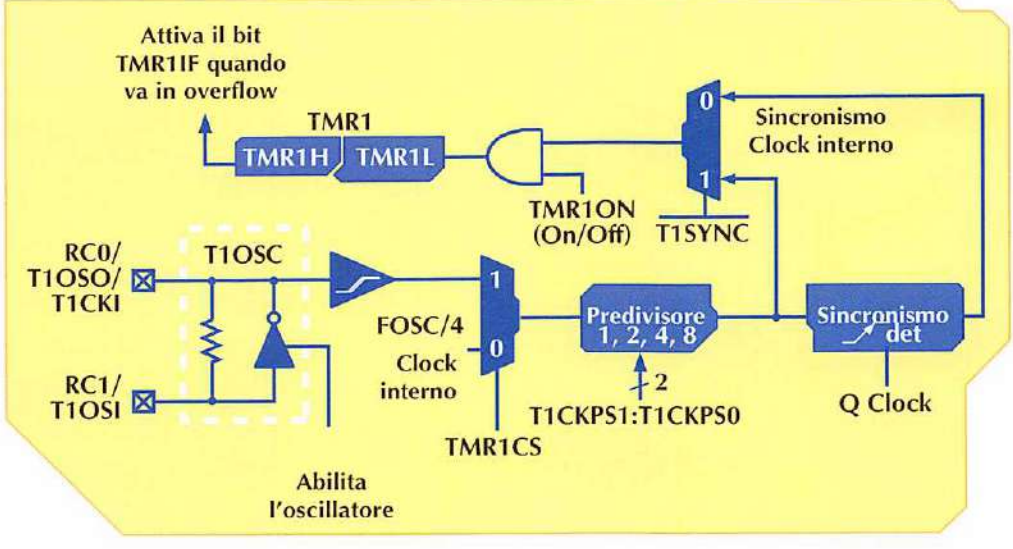
Il contenuto di TMR1H:TMR1L può essere letto e scritto. Gli impulsi di clock possono essere esterni o interni (Fosc/4). Nella figura possiamo vedere l'architettura interna del temporizzatore.

Il TMR1 può funzionare come temporizzatore e come contatore. Se funziona come contatore dobbiamo distinguere due modi di lavoro: sincrono e asincrono. Quando il TMR1 funziona come temporizzatore, i registri



Il temporizzatore TMR1 utilizza 16 bit.

TMR1H:TMR1L incrementano il loro valore a ogni ciclo di istruzione (Fosc/4). Funzionando come contatore, può ricevere gli impulsi di clock da un oscillatore esterno collegato ai pin RC0 e RC1 della porta C (come si può vedere nello schema a blocchi), in cui si applicano



Schema a blocchi del temporizzatore TMR1.

T1CON: Registro di controllo del TMR1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit7							bit0

bit7-6 Non implementati
bit5-4 T1CKPS1:T1CKPS0: Predivisore di frequenza del TMR1
bit3 T1OSCEN: Abilitazione dell'oscillatore del TMR1 (pin RC1-RC0)
1= Oscillatore abilitato
0= Oscillatore disabilitato
bit2 T1SYNC: Sincronizzazione degli impulsi esterni con il clock interno
1= Non sincronizza il clock esterno
0= Sincronizza
Quando TMR1CS=0:
Il bit è ignorato. Si usa il clock interno
bit1 TMR1CS: Selezione della sorgente per gli impulsi del contatore
1= Clock esterno tramite il pin RC0
0= Clock interno Fosc/4
bit0 TMR1ON: Abilitazione del TMR1
1= Abilita il TMR1
0= Disabilita il temporizzatore

Registro T1CON per il controllo del temporizzatore TMR1.

fronti di salita o tramite un clock esterno collegato al pin RC0.

Registro di controllo T1CON

Il registro di controllo T1CON regola il comportamento del temporizzatore TMR1. Questo registro occupa l'indirizzo 10h della memoria RAM. Nella figura possiamo vedere la sua struttura e come il valore di ciascuno dei suoi bit influenza il funzionamento del TMR1.

Il bit TMR1ON abiliterà l'utilizzo del temporizzatore, il TMR1CS ci permette di scegliere la sorgente degli impulsi per il conteggio (clock interno o esterno), il bit T1OSCEN si configura per accettare o meno gli impulsi che arrivano dall'esterno, il bit T1SYNC determinerà se sincronizzare o meno gli impulsi esterni con quelli del clock interno e i bit T1CKPS1:0 selezionano il valore del fattore applicato per la divisione della frequenza degli impulsi che si applicano al TMR1.

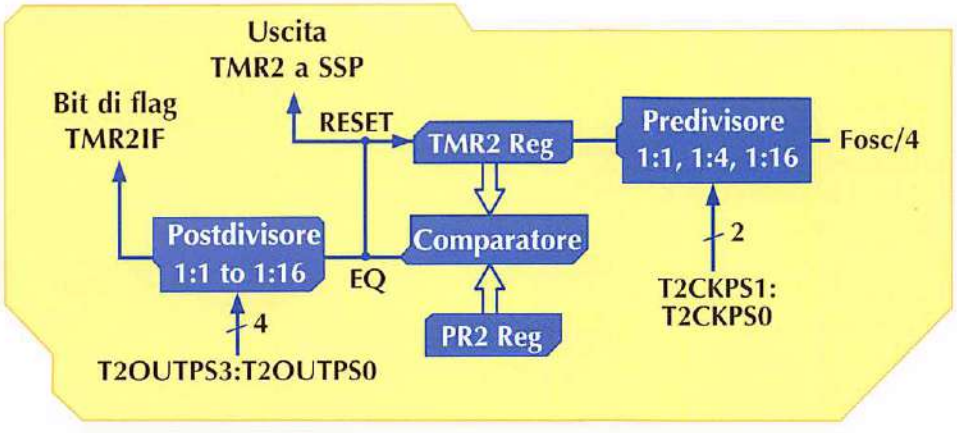
Quest'ultimo, come possiamo vedere nella figura, ha quattro range di lavoro.

T1CKPS1	T1CKPS0	Range del predivisore
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

Selezione del range sul predivisore del TMR1.

Indirizzo	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore in POR e BOR	Valore nel resto dei reset
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Registro che contiene il byte meno significativo del TMR1								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Registro che contiene il byte più significativo del TMR1								xxxx xxxx	uuuu uuuu
10h	T1CON	-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Registri relativi al TMR1.



Schema a blocchi del TMR2.

Altri registri relativi al TMR1

Quando si verifica un overflow del valore di conteggio (da FFFFh a 0000h) il TMR1 attiva il suo flag TMR1IF automaticamente. Questo è il bit meno significativo del registro PIR1. Per fare in modo che questo flag provochi un interrupt quando si attiva, il bit di abilitazione dell'interrupt del TMR1 deve essere attivato, TMR1IE = 1. Questo bit è quello meno significativo del registro PIE1.

Invece, quando il modulo di Capture/Compare/PWM (CCP) è configurato come comparatore con "attivazione speciale", se si produce questa attivazione (quando coincidono i valori che compara) il TMR1 si resetta. Per utilizzare tale caratteristica dobbiamo programmare il TMR1 per il funzionamento in modo temporizzatore o contatore sincrono.

Nella tabella della figura sono riportati i registri relativi al Timer 1. Sono indicati gli indirizzi del registro, il nome di quest'ultimo, la sua scomposizione in bit e lo stato che questi bit assumono dopo un reset. Il valore 'x' significa indeterminato, 'u' che rimane invariato e quelli non implementati '-' sono letti come 0.

Il temporizzatore TMR2

Il TMR2 è un temporizzatore a 8 bit con predivisore e postdivisore di frequenza. Occupa l'indirizzo 11h del banco 0 della memoria RAM e assume alcune funzioni speciali per implementare il funzionamento della Porta Seriale Sincrona (SSP) e per i moduli di Capture/Compare/PWM (CCP). Il registro del TMR2 può essere letto e scritto, e, mediante un reset, si inizializza il suo valore.

Il segnale di clock del TMR2 è quello interno del processore (Fosc/4) e presenta l'opzione di dividere la frequenza in range da 1:1, 1:4 e 1:16, in base a come avremo programmato i due bit meno significativi del registro T2CON. L'uscita del registro TMR2 passerà attraverso un postdivisore di frequenza con range consecutivi da 1:1 fino a 1:16. Nella figura possiamo vedere l'architettura interna del TMR2.

T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit7							bit0

bit7	Non implementato. Letto come 0
bit6-3	TOUTPS3:TOUTPS0: Bit di selezione del postdivisore del Timer 2
bit2	TMR2ON: Bit di attivazione del Timer 2 1= TMR2 attivato 0= TMR2 disattivato
bit1-0	T2CKPS1:T2CKPS0: Bit di selezione del predivisore di frequenza

Registro T2CON per il controllo del temporizzatore TMR2.

Registro di controllo T2CON

In base a come programmeremo il registro di controllo T2CON, il TMR2 si comporterà in modo diverso. Questo registro occupa l'indirizzo 12h del banco 0 della memoria RAM. Esiste un bit in questo registro, il TMR2ON, che permette di scollegare il temporizzatore e quindi, risparmiare consumo di energia nel microprocessore. I due bit meno significativi di questo registro selezionano il range del predivisore di frequenza in accordo con la tabella della figura. Il bit più significativo del registro non è implementato e i quattro successivi determinano il range con cui funzionerà il postdivisore di frequenza, come si può vedere nella tabella della figura.

T2CKPS1	T2CKPS0	Range del predivisore
0	0	1:1
0	1	1:4
1	X	1:16

Selezione del range del predivisore di frequenza.

TOUTPS3:TOUTPS0	Range del postdivisore
0000	1:1
0001	1:2
0010	1:3
....	....
1111	1:16

Selezione del range del postdivisore di frequenza.

Altri registri relativi al TMR2

L'uscita del postdivisore di frequenza controlla l'attivazione del flag TMR2IF, bit del registro PIR1 che si attiva automaticamente quando il temporizzatore va in overflow. In questo modo possiamo generare un interrupt interno al microcontroller con il TMR2.

Per permettere l'interrupt del TMR2 dobbiamo impostare a 1 il bit di abilitazione TMR2IE, però, dato che questo temporizzatore è un dispositivo ausiliario, è necessario attivare anche il bit di abilitazione per questo tipo di dispositivi, che è il PEIE.

Non dobbiamo dimenticare che anche il bit di abilitazione globale di interrupt è necessario che sia attivato. Quindi, se TMR2IE=PEIE=GIE=1, quando si attiva il flag (TMR2IF=1) si genera un interrupt.

Il TMR2 ha associato un registro di periodo PR2, che occupa l'indirizzo 92h della memoria dei dati RAM. Se il valore del TMR2 coincide con quello di PR2 si genera un impulso EQ e si resetta il TMR2. Il postdivisore può dividere tali impulsi in EQ prima dell'attivazione del bit di flag (TMR2IF).

I registri associati al TMR2 sono raccolti nella tabella, dove è indicato l'indirizzo che occupano nella memoria, i bit di cui sono composti e lo stato che questi bit assumono dopo un reset.

Conclusioni

Abbiamo visto i temporizzatori di cui dispone il microcontroller. Compreso il loro funzionamento, potremo utilizzarli in applicazioni future. Continueremo l'analisi dei dispositivi del PIC fino a iniziare lo studio del repertorio delle istruzioni. Ripassate i concetti studiati e teneteli presenti per i capitoli successivi.

Indirizzo	NOME	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore in POR e BOR	Valore nel resto dei reset
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
11h	TMR2	Registro del modulo TMR2								0000 0000	uuuu uuuu
92h	PR2	Registro del periodo del TMR2								1111 1111	1111 1111
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	--00 0000	--uu uuuu

Registri relativi al TMR2.

Simulazione e Break Points

Terminiamo le opzioni di simulazione analizzando i Break Points o punti di rottura. Questi elementi solitamente sono presenti nei simulatori di tutti i linguaggi di programmazione, dato che il loro impiego è molto utile nella fase di simulazione dell'applicazione.

Continuazione delle opzioni Debug → Run

Rimangono da analizzare le due opzioni di simulazione all'interno del menù Run che appartengono al campo Debug. Queste due opzioni non sono fra le più utilizzate, però è necessario sapere che esistono e che si possono utilizzare.

Update All Registers

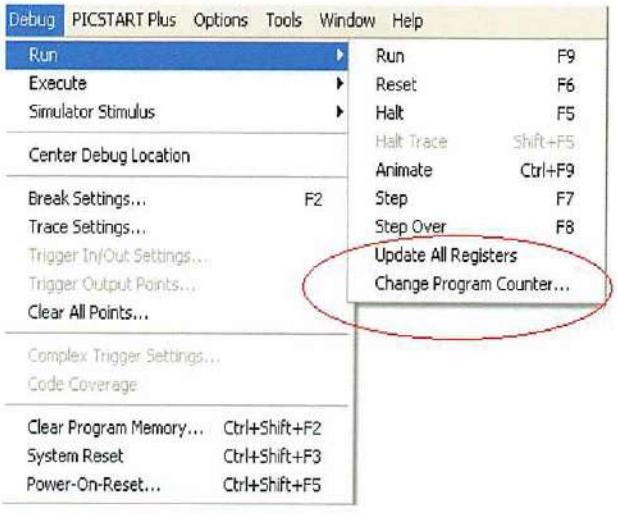
Questa opzione aggiorna il contenuto di tutti i registri dopo che è stata eseguita un'istruzione. Anche se i registri si aggiornano mano a mano che eseguiamo il programma, se vogliamo lasciare registrata una traccia del programma e teniamo aperta la finestra di memoria della traccia Window → Trace Memory al momento dell'esecuzione dell'istruzione di cui vogliamo registrare la traccia, dobbiamo selezionare questa opzione, dato che, diversamente, la traccia non viene aggiornata.

Change Program Counter

Selezionando Change Program Counter possiamo puntare con il PC a qualsiasi indirizzo di memoria o a qualsiasi etichetta contenuta nel nostro programma.

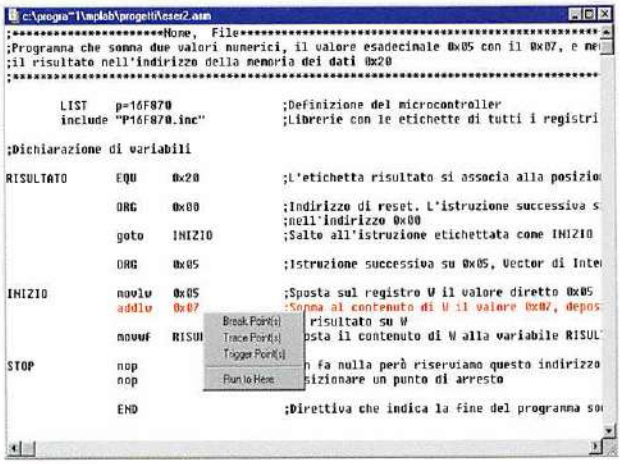
Apparirà la finestra della figura in basso e in essa indicheremo qualsiasi indirizzo, inserendola nel campo "PC". È possibile inserire direttamente oppure cliccando sulla freccia e selezionando una qualsiasi delle etichette. Indicato l'indirizzo selezioneremo "Change" e il PC punterà al nuovo indirizzo.

Grazie a questa opzione potremo saltare ad altri indirizzi ed evitare così di entrare in cicli di programma da cui potrebbe essere difficile uscire, perché dipendono da determinate condizioni o cicli che impiegano molto tempo a essere eseguiti, come ad esempio, le temporizzazioni. Questa opzione può essere utilizzata

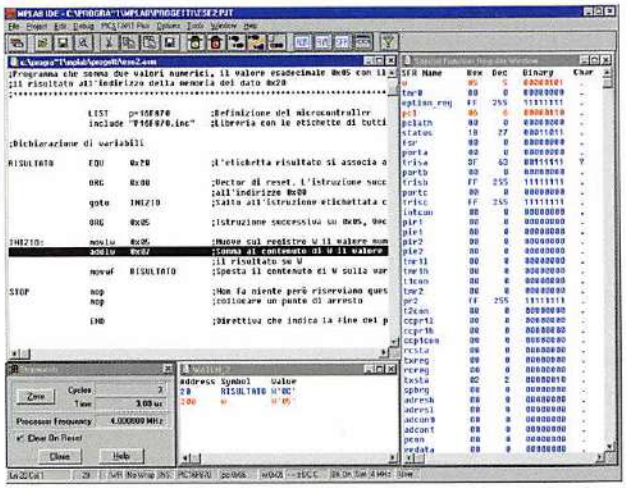


Opzioni del menù Run all'interno di Debug.

Finestra per modificare il valore del contatore di programma PC.



Quadro di dialogo che si apre cliccando il pulsante destro del mouse su una linea di codice.



Valori e stato di MPLAB quando, durante l'esecuzione del programma, trova un Break Point.

quante volte lo si desidera e la si attiva mediante un'icona della barra degli strumenti.

Break Points o Punti di Rottura

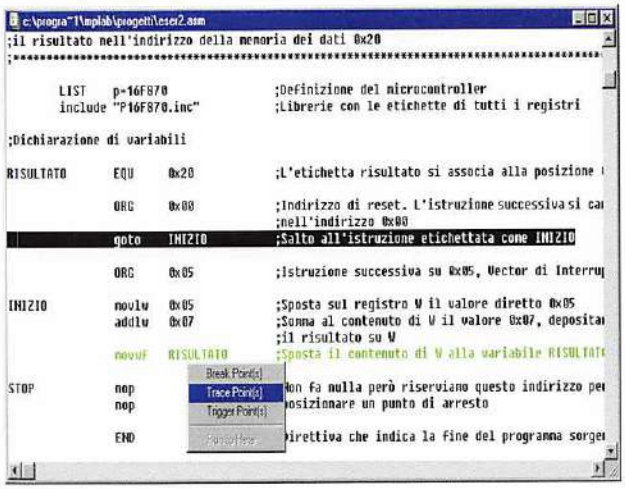
Quando vogliamo fermare l'esecuzione di un programma dopo aver eseguito una determinata istruzione e realizzare in questo modo un'analisi degli eventi fino a questo momento, utilizziamo i Break Points. Ci sono diversi modi di inserire un punto di rottura, però il modo più semplice è posizionarsi con il mouse sulla linea di programma dove vogliamo collocarlo e cliccare il pulsante destro. Fatto questo si aprirà un menù come quello della figura, in cui potremo selezionare l'opzione Break Point(s). Nella finestra rimarrà evidenziata in rosso la linea o le linee in cui vorremo che il programma si fermi durante l'esecuzione in modo Run o Animate.

Esempio: se carichiamo il progetto con cui abbiamo lavorato finora e posizioniamo un Break Point sulla linea della figura precedente (addlw 0x07) quando eseguiamo il programma possiamo vedere come si ferma l'esecuzione sulla linea selezionata. Nella figura possiamo vedere i valori che hanno assunto i registri e anche il tempo impiegato dal programma nell'esecuzione fino al punto di rottura.

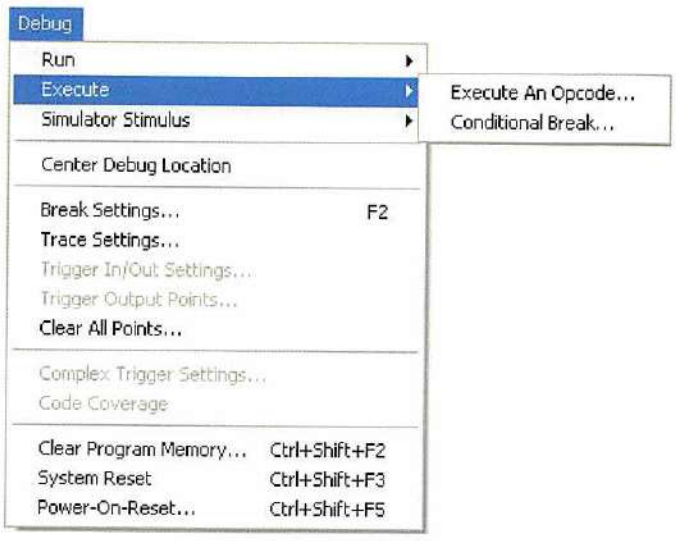
Trace Point o Traccia

Quando vogliamo vedere l'evoluzione del microcontroller durante l'esecuzione di una determinata istruzione dobbiamo creare una Traccia. Mediante questa si annota il valore che assumono i registri modificati dall'istruzione ogni volta che il programma passa su di essa. Se ci posizioniamo con il mouse sulla linea dove vogliamo ubicare il Trace Point e clicchiamo il pulsante destro, apparirà l'opzione Trace Point(s). La linea selezionata verrà evidenziata di colore verde, e per vedere il contenuto dei registri coinvolti in questa istruzione dobbiamo aprire la finestra Window→Memory Trace.

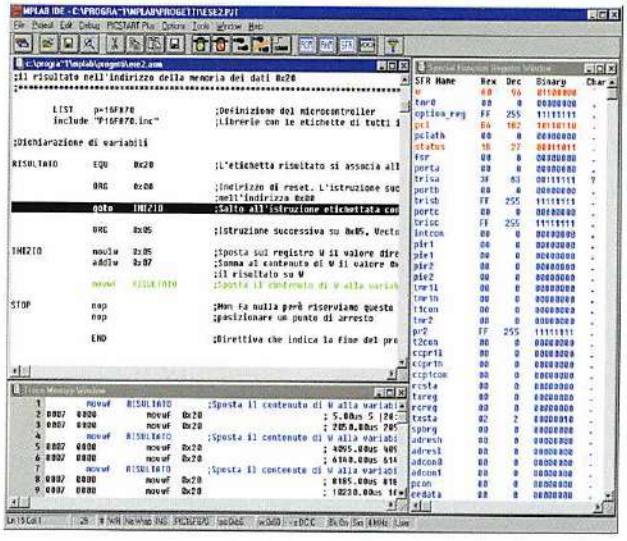
Esempio: togliamo il Break Point dell'esempio precedente (per fare questo dobbiamo tornare a posizionare il mouse sulla linea evidenziata in rosso e con il pulsante destro sele-



Ora selezioniamo Trace Point(s).



Apertura dell'opzione Execute.



Terminata l'esecuzione con un Halt nella finestra Memory Trace possiamo vedere la traccia selezionata.

zionare Break Point(s)) e sulla linea mostrata nella figura posizionare il nostro Trace Point (movwf RISULTATO). Faremo un reset del sistema (la nostra finestra Memory Trace sarà vuota) ed eseguiremo. Il programma si esegue normalmente, però quando selezioniamo Halt nella finestra di Memory Trace appaiono i valori cercati come si può vedere nella figura.

Execute

Questa opzione ci permette di fare dei cambi nell'esecuzione del programma. Quando la selezioniamo si aprono due nuove opzioni per il controllo dei registri durante l'esecuzione.

Execute an Opcode

Attivando questa opzione possiamo eseguire una o più istruzioni senza modificare il codice sorgente.

Dopo aver eseguito l'istruzione il programma continuerà normalmente e la memoria di programma partirà dallo stato in cui è rimasta dopo l'esecuzione dell'istruzione. Nel campo Opcode inseriremo l'istruzione che desideriamo eseguire e mediante il pulsante Execute la eseguiremo, senza modificare né il PC né i temporizzatori.

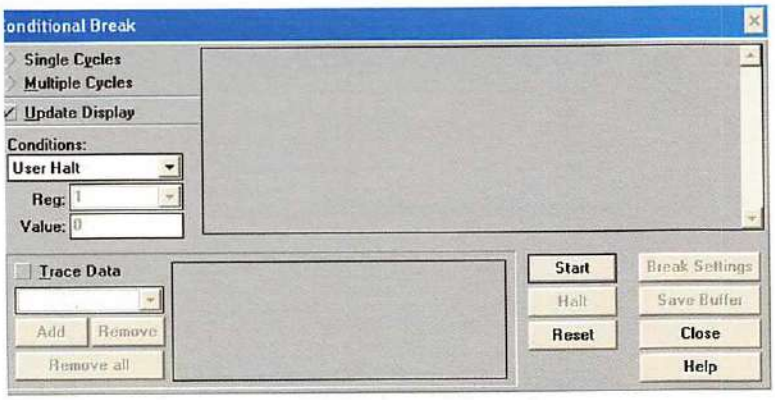
Conditional Break

Mediante questa opzione possiamo fissare un punto di rottura o Break Point condizionale, ovvero, un punto in cui MPLAB si fermerà se, e solo se, si compie una determinata condizione precedentemente fissata.

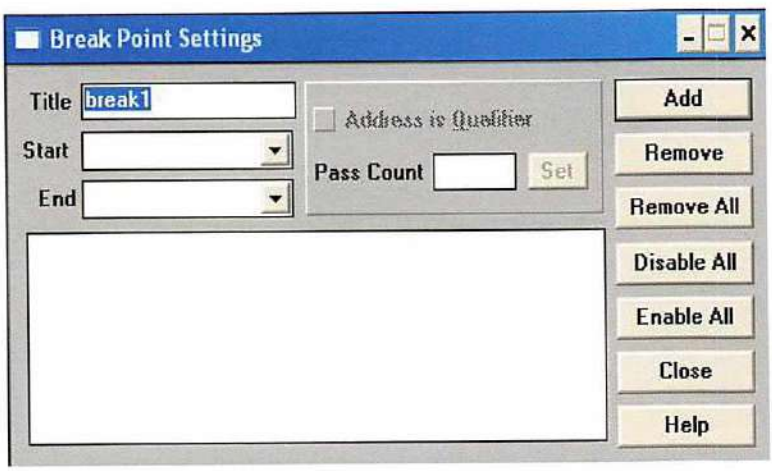
Nella figura possiamo vedere le videate che appaiono quando selezioniamo Conditional Break.

L'esecuzione inizia quando clicchiamo il pulsante Start e si ferma sul Break Point quando si compie la condizione specificata o clicchiamo Halt. Non approfondiremo questo ar-

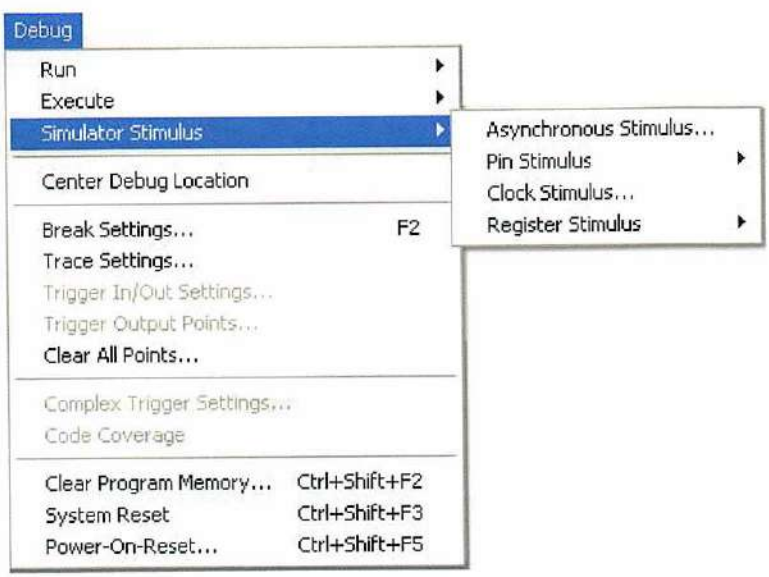
Opzione Execute an Opcode.



Quadro di dialogo del punto di rottura condizionale.



Finestra per modificare le caratteristiche dei Break e Trace Points.



Simulatore di stimoli esterni.

gomento perché non è molto utilizzato per la simulazione, però vi consigliamo di provare senza paura, sui vostri progetti, tutte le combinazioni e opzioni che presentano le funzioni studiate.

Setting (Break e Trace Points)

Nel menù Debug esiste l'opzione per configurare le caratteristiche dei punti di rottura e delle tracce mediante Break Setting e Trace Settings. Potrete verificare che solamente l'opzione Break Point dispone di un tasto di accesso rapido F2. Questo è così perché le tracce non sono molto utilizzate nella simulazione, invece i Break Points sono una delle tecniche più utilizzate dai programmatori in fase di messa a punto.

Possiamo indicare il numero di volte che vogliamo che si ripeta un'istruzione o la condizione che vogliamo compia il Break Point o la traccia.

Conclusioni

Per completare il corso che stiamo facendo del software MPLAB ci rimane solamente da studiare il simulatore di stimoli (Simulator Stimulus). Questa opzione ci permette di simulare gli ingressi che potremmo avere in un'applicazione reale, e poter quindi studiare la risposta del nostro programma.

Dopo aver visto il simulatore di stimoli potremo simulare qualsiasi progetto con MPLAB e potremo mettere a punto qualsiasi programma con questo strumento.

Per affinare le nostre conoscenze e acquisire una certa scioltezza con MPLAB ci eserciteremo con diversi esempi prima di iniziare con le applicazioni specifiche del Laboratorio, le quali comprendono tutte un capitolo di simulazione con MPLAB.

Watchdog timer, il cane da guardia

Il WDT è un contatore che genera un reset quando va in overflow. Molti microcontroller sono dotati di questo dispositivo, perché è molto utile per evitare che i nostri programmi rimangano bloccati o si perdano all'interno di qualche ciclo.

Funzionamento

Il Watchdog è un contatore ascendente da 8 bit che quando va in overflow provoca un reset sul processore. La sua funzione è quella di controllare la corretta elaborazione delle istruzioni del programma ed evitare di cadere in cicli infiniti o di restare in attesa di un segnale che non arriva mai. Il programmatore deve posizionare in modo strategico all'interno del programma le istruzioni che azzerano il WDT in modo da evitare che vada in overflow. In questo modo, se il programma segue uno sviluppo corretto, verranno eseguite le istruzioni e, prima che il WDT vada in overflow, ve ne sarà una che lo aggiornerà. Nel caso in cui il programma rimanga bloccato in un ciclo, il WDT non verrà reinizializzato e, quando giungerà al termine del suo conteggio, genererà un reset a tutto il sistema.

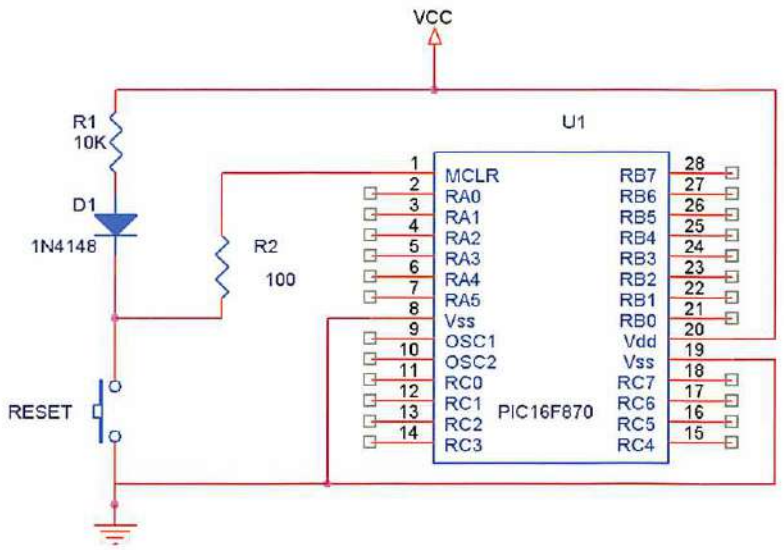
Tutti coloro che lavorano con un computer hanno dovuto qualche volta reinizializzare il PC con un reset, perché il sistema è rimasto bloccato. Con il microcontroller possiamo fare un reset manuale, mediante un apposito circuito collegato al piedino numero 1 del PIC ($\overline{\text{MCLR}}$), oppure lavorare con il WDT in modo che questo eviti automaticamente che si blocchi il programma.

L'istruzione CLRWDT

Per reinizializzare o aggiornare il valore del conteggio del WDT utilizzeremo l'istruzione CLRWDT (Clear Watch Dog Timer). Questa istruzione cancella il valore del WDT, reinizializzando il conteggio, e dovrà essere posizionata in punti strategici del programma per fare in modo che, se tutto va bene, il contatore non possa andare in overflow.

Anche l'istruzione SLEEP cancella il WDT, inoltre ferma tutto il sistema portandolo in un modo di lavoro in cui il consumo è minimo (modo riposo o di basso consumo).

Se non si disattiva il Watchdog entrando in modo riposo, quando giunge al termine del conteggio provocherà un reset e uscirà dal suddetto modo.



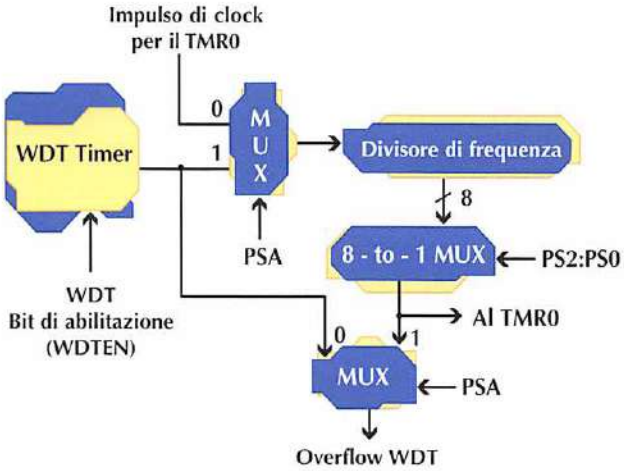
Reset manuale del PIC16F870.

Funzionamento del Watchdog.

Per disattivare il Watchdog imposteremo a 0 il bit 2 (WDTEN) della parola di configurazione.

Scrittura interna

Il WDT ha un controllo del tempo indipendente dalla rete principale, basato su un oscillatore RC (Resistenza-Condensatore) interno.



Schema interno del Watchdog.

PS2:PS0	Range del WDT	Range del TMR0
0 0 0	1:1	1:2
0 0 1	1:2	1:4
0 1 0	1:4	1:8
0 1 1	1:8	1:16
1 0 0	1:16	1:32
1 0 1	1:32	1:64
1 1 0	1:64	1:128
1 1 1	1:128	1:256

Range del postdivisore in funzione dei tre bit PS2:PS0.

Questo oscillatore è indipendente dall'oscillatore che può essere sul pin OSC1/CLKI, questo significa che il WDT continuerà a girare anche quando il clock dei pin OSC1/CLKI e OSC2/CLKO si sarà fermato, come succede con l'istruzione SLEEP.

La temporizzazione nominale con cui è programmato il Watchdog è di 18 ms, però la si può aumentare utilizzando il divisore di frequenza. Con il massimo valore del range si può raggiungere un tempo di aggiornamento di 2,3 secondi. Questo divisore è uguale a quello utilizzato dal TMR0 e può essere utilizzato solamente da uno dei dispositivi per volta, ovvero, o lo si utilizza con il WDT o con il TMR0.

Nella figura possiamo analizzare l'architettura interna del WDT e come interagisce con il divisore di frequenza. Gli impulsi del WDT possono passare tramite il postdivisore di frequenza che li divide per il range selezionato. Questo range si determina in base al valore dei tre bit del registro OPTION_REG (PS2:PS0) in base alla tabella della figura.

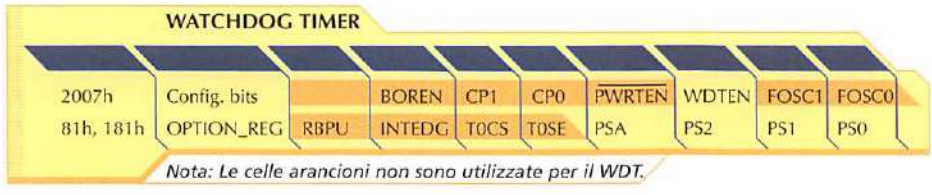
Per assegnare il divisore di frequenza al WDT è necessario impostare il bit PSA=1 sul registro OPTION_REG.

Quando il WDT va in overflow, il bit che ci indica che si è generato questo evento è il bit $\overline{TO}$ (STATUS<4>), che assumerà valore 0. In questo modo possiamo discriminare la causa che ha generato il reset.

Esempio

Il Watchdog di un PIC è programmato per andare in overflow dopo 18 ms. Se sappiamo che ogni istruzione impiega 1 µs per essere eseguita, calcolare ogni quante istruzioni, come massimo, bisogna aggiornare il Watchdog.

Registro della Parola di Configurazione o Configuration Word.



Registri che intervengono nel funzionamento del WDT.

$$\frac{18\ \text{ms}}{1\mu\text{s / istruzioni}} = \frac{18 \cdot 10^{-3}}{1 \cdot 10^{-6}} \quad 18000\ \text{istruzioni}$$

Istruzioni da eseguire prima che il WDT vada in overflow.

Se un'istruzione dura 1 µs e bisogna cancellare il WDT ogni 18 ms come massimo, sarà possibile eseguire le istruzioni riportate nella figura.

Esempio

Dato un lavoro critico all'interno del programma principale eseguito da un microcontroller PIC che lavora a 4 MHz, lo vogliamo sorvegliare con il Watchdog, in modo che, se si supera il tempo stabilito per la sua esecuzione, si reinizializza il processore.

Il lavoro consiste nel ripetere 100 volte un ciclo composto da 300 istruzioni normali, senza nessuna di salto.

All'inizio, davanti a ogni lavoro si colloca un'istruzione CLRWDT per reinizializzare il Watchdog. Al termine del lavoro posizioneremo un'altra istruzione CLRWDT per impedire il reset da overflow se il ciclo viene eseguito nel tempo stabilito. Quando questo tempo viene superato si genera un reset. Calcolare il tempo con cui vogliamo programmare il WDT.

Per calcolare questo tempo, come prima cosa, dobbiamo stimare il tempo impiegato a realizzare il lavoro normalmente, dopodiché programmarlo affinché vada in overflow nel caso duri di più. Nella figura in basso si possono vedere i passaggi seguiti per risolvere il nostro esempio.

Esempio

Vediamo ora un programma reale di tipo combinazionale. Si tratta di leggere il valore di una serie di segnali logici di ingresso (RA0-RA4) della porta A, realizzare un'operazione aritmetica con il suddetto valore (sommiamo la costante 5) e depositare il risultato su una porta di uscita (RB0-RB7 della porta B).

Nella figura riprodotta alla pagina successiva possiamo vedere il codice risultante del programma, con i relativi commenti che ci aiuteranno a capirlo. Come da abitudine, la prima parte o intestazione del programma è praticamente uguale a tutti gli altri: dichiarazione del PIC, del file che contiene le definizioni dei registri e l'organizzazione del programma in memoria.

Il programma in sé comincia a partire dall'etichetta 'Inizio', poi si configurano i dispositivi da utilizzare. In ultimo, nella terza parte, dopo l'etichetta 'Loop' si specifica ciò che vogliamo fare. Possiamo verificare che la prima istruzione di questo blocco è 'clrwdt', dato che siamo in un ciclo che si ripete costantemente e non vogliamo che per qualche motivo il programma si blocchi. In questo modo, se il microcontroller non può ricevere i segnali di ingresso o riportare il risultato sull'uscita, non continuerà a eseguire le istruzioni, quindi non eseguirà quella di aggiornamento del WDT, il che significa che si produrrà un Reset.

In tutti i programmi che contengono cicli interattivi si deve lavorare con il WDT e inserire istruzioni di refresh o aggiornamento del WDT collocate in punti strategici.

Copiate questo programma sull'editor di MPLAB, creando in precedenza un progetto, compilatelo ed eseguitelo. Osservate quindi come si modificano i registri ed esercitatevi con tutte le conoscenze acquisite.

Il ciclo di 300 istruzioni normali ha una durata:

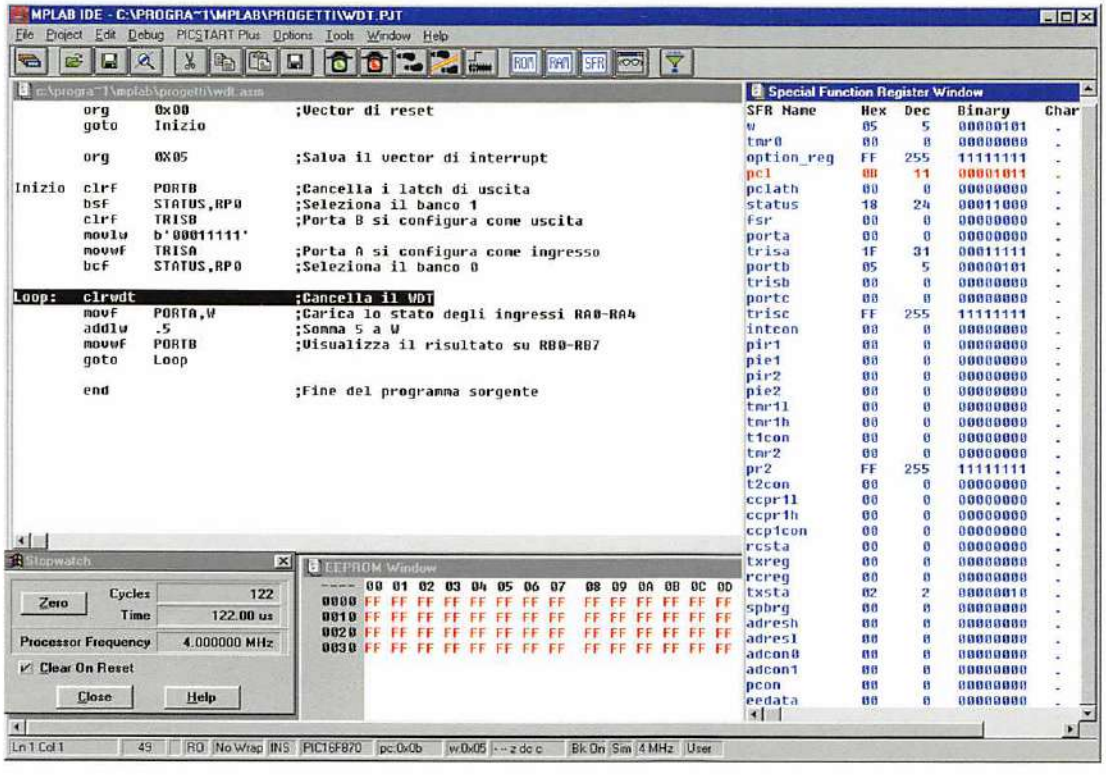
- Ciclo istruzione = 4·Tosc = 4· (1/4·10⁶) = 1·10⁻⁶ = 1 µs
- Tempo di ciclo = 300 · Ciclo istruzione = 300·1·10⁻⁶ = 300 µs
- Tempo Totale = 100 · Tempo di ciclo = 100·300·10⁻⁶ = 30·10⁻³= 30 ms

Il WDT si dovrà programmare e caricare con il valore adatto, in modo che vada in overflow alla fine di un periodo appena più lungo di 30 ms.

Soluzione al secondo esempio.

Programma risultante dall'esercizio di tipo combinazionale

```
                List        p=16F870        ;Tipo di processore
                include     "P16F870.INC"   ;Definizione dei registri interni

                org         0x00            ;Vector di Reset
                goto        Inizio

                org         0x05            ;Salva il vector di interrupt

Inizio    clrf  PORTB                       ;Cancella i latch di uscita
                bsf         STATUS,RP0      ;Seleziona banco 1
                clrf        TRISB           ;Porta B configurata come uscita
                movlw       b'00011111'
                movwf       TRISA           ;Porta A configurata come ingresso
                bcf         STATUS,RP0      ;Seleziona banco 0

Loop:           clrwdt                      ;Cancella il WDT
                movf        PORTA,W         ;Carica lo stato degli ingressi RA0-RA4
                addlw       .5              ;Somma 5
                movwf       PORTB           ;Visualizza il risultato su RB0-RB7
                goto        Loop

                end                         ;Fine del programma sorgente
```



Aspetto di MPLAB caricando il terzo esempio.

Le memorie EEPROM e FLASH

Le memorie del PIC16F870 danno a questo dispositivo un importante vantaggio rispetto ai suoi concorrenti. Abbiamo visto i concetti generali, però dobbiamo imparare a utilizzarli; a questo scopo dobbiamo entrare nel dettaglio, analizzare i registri che sono coinvolti e come si lavora con essi.

Promemoria

Come abbiamo già detto, il nostro microcontroller lavora con tre tipi di memoria: una memoria delle istruzioni FLASH e due memorie dei dati, una RAM e una EEPROM.

– Memoria di istruzioni FLASH. In essa si scrive il programma dell'applicazione, il codice del programma. È costruita con tecnologia FLASH.

– Memoria dei dati RAM. Contiene i dati del programma e i registri di controllo. È di tipo RAM, molto veloce e volatile.

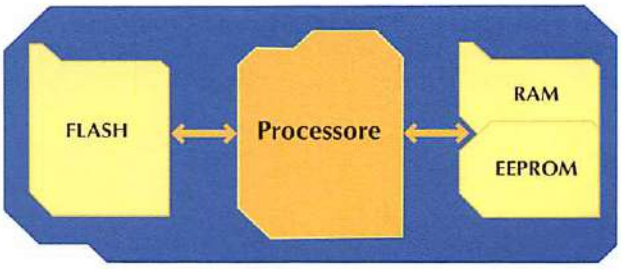
– Memoria dei dati EEPROM. Contiene i dati che devono rimanere in memoria anche togliendo l'alimentazione. È una memoria non volatile che si può leggere, scrivere, cancellare e riscrivere circa 1 milione di volte.

Modi di indirizzamento dei dati

Le istruzioni dei PIC16F870 possono specificare i dati e gli operandi mediante tre modi di indirizzamento:

– Immediato: il valore del dato è contenuto nel codice OP dell'istruzione e, nell'esecuzione, si carica nel registro W per la sua successiva elaborazione.

– Diretto: si utilizzano i 7 bit meno significativi del codice OP dell'istruzione per puntare a una qualsiasi delle 128 posizioni del banco 0. Per la scelta del banco si utilizzano i bit 6 e 5 del registro STATUS, che si chiamano RP1 e RP0.



Le tre memorie del PIC16F870.

PIC16F870		
MEMORIA DEL PROGRAMMA	2K x 14 words	FLASH
MEMORIA DEI DATI	128 x 8 byte	RAM
	64 x 8 byte	EEPROM

Capacità delle memorie del microcontroller.

– Indiretto: si utilizza come operando il registro INDF, che occupa l'indirizzo 0 in tutti i banchi. In questo caso si accede all'indirizzo puntato dal contenuto del registro FSR, situato all'indirizzo 4 dei banchi di memoria.

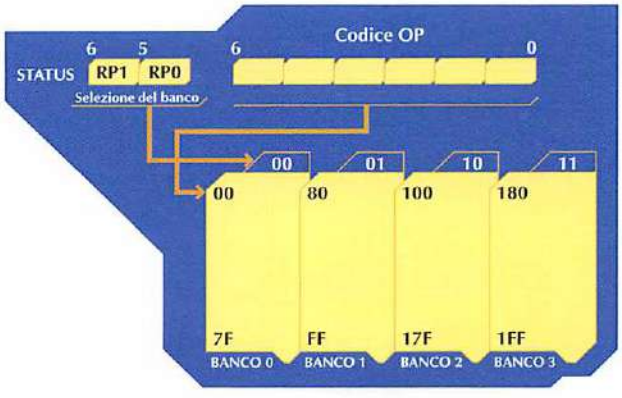
Il registro INDF non è implementato fisicamente, quindi ogni volta che si fa riferimento a esso si utilizza il contenuto del registro INDF per indirizzare l'operando.

La memoria EEPROM

Lo spazio della memoria EEPROM è uno spazio di memoria indipendente, che non è mappato all'interno della zona di memoria dei dati RAM, e il cui indirizzamento è particolare.

La memoria EEPROM è formata da 64 byte, posti dall'indirizzo 00h all'indirizzo 3Fh, che possono essere letti o scritti.

Per lavorare con questa memoria dobbiamo gestire i registri indicati nella tabella e che analizzeremo di seguito.



Indirizzamento diretto.

EEADR	10Dh	Indirizzo della EEPROM
EEADRH	10Fh	Parte alta dell'indirizzo di accesso alla memoria FLASH
EEDATA	10Ch	Dato contenuto nell'indirizzo della EEPROM
EEDATH	10Eh	Parte alta del contenuto quando si accede alla FLASH
EECON1	18Ch	Registro di controllo 1
EECON2	18Dh	Registro di controllo 2

Registri coinvolti nella gestione della memoria.

Struttura della EEPROM

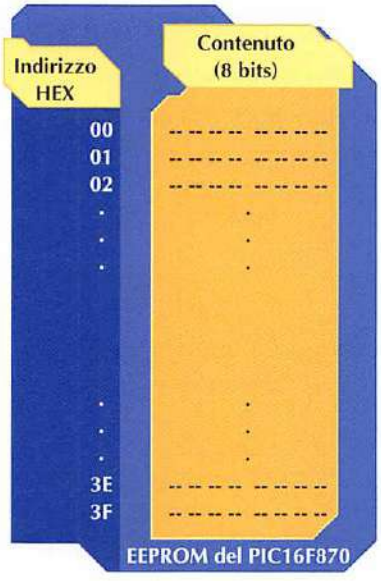
Il registro EEADR contiene gli indirizzi della memoria EEPROM a cui vogliamo accedere; i registri di controllo ci indicano le operazioni da realizzare su di essa e il registro EEDATA conterrà il dato situato nell'indirizzo in caso di lettura o il dato che si vuole memorizzare quando si scrive.

Dato che l'indirizzo più alto è 3Fh e il dato contenuto è da 8 bit, sono sufficienti i registri EEADR e EEDATA. Questo non basta invece per la memoria FLASH, poiché sia gli indirizzi sia i dati sono più grandi di 8 bit, quindi sono necessari due registri per indicare l'indirizzo e altri due per contenere il dato (utilizzeremo EEADR e EEADRH per l'indirizzo, EEDATA e EEDATH per il dato).

Registri di controllo per la gestione della EEPROM

Per lavorare con la memoria EEPROM dobbiamo specificare che azione desideriamo eseguire con la memoria. Per gestire il comportamento e l'operatività di quest'ultima utilizziamo i registri EECON1 e EECON2. il registro EECON2 non è implementato realmente, però si utilizza per assicurare la sequenza delle operazioni di scrittura nella EEPROM.

Il fattore tempo è molto critico nell'operazione di scrittura di questa memoria, in quanto può durare circa 2 millisecondi, tempo molto lungo per un PIC. Dato che dipende da fattori quali la tensione di alimentazione, la temperatura di lavoro, ecc., e che il tempo è elevato, si controlla il momento in cui termina questa operazione mediante il flag EEIF, che si attiverà al termine dell'operazione di scrittura.

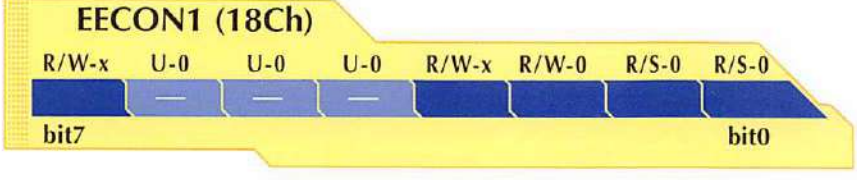


Struttura della EEPROM del PIC16F870.

Il registro EECON2, che non è implementato fisicamente, viene caricato con il valore 55h e dopo con il valore AAh, prima di iniziare un'operazione di scrittura, secondo le direttive dello stesso costruttore.

Nella tabella della figura possiamo vedere il registro EECON1 analizzato nei bit che lo compongono. Durante l'inizializzazione del controller viene disabilitata la scrittura dei dati nella EEPROM. Questa operazione dura 72 millisecondi e coinvolge nello stesso modo la scrittura della memoria FLASH. Il flag che controlla questa operazione è il WRT della Parola di Configurazione. Il bit WREN si utilizza per abilitare l'opzione di scrittura, evitando in questo modo scritture indesiderate. Dobbiamo abilitarlo per eseguire la scrittura.

Il bit WRERR indica, con il suo passaggio a 1, che si è generato un errore di scrittura, normale conseguenza della mancanza del tempo necessario per realizzare questa operazione a causa di un reset o dell'owerflow del Watchdog. Nella figura possiamo vedere un esempio



Struttura interna del registro di controllo EECON1.

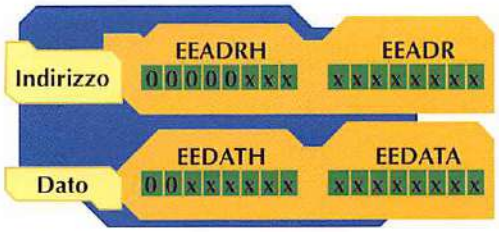
EECON1			
EEPGD	Bit 7	0	Accesso alla EEPROM
		1	Accesso alla FLASH
--------	Bit 6-4		Non implementati
WRERR	Bit 3	0	Non si è verificato errore nella scrittura
		1	C'è un errore nella scrittura, per mancanza del tempo necessario
WREN	Bit 2	0	Disabilita l'opzione di scrittura sulla EEPROM
		1	Bit di abilitazione di scrittura Lo dobbiamo impostare a 1 all'inizio di questa
WR	Bit 1	0	Valore che assume automaticamente al termine dell'operazione di scrittura
		1	Dobbiamo impostarlo a 1 per iniziare un'operazione di scrittura
RD	Bit 0	0	Non inizia un'operazione di lettura
		1	Dobbiamo impostarlo a 1 per iniziare un'operazione di lettura

Utilità dei bit del registro EECON1.

di programma in cui si esegue la scrittura nella memoria EEPROM.

L'interrupt della EEPROM

Per abilitare l'interrupt di fine scrittura nella EEPROM dobbiamo aver attivato il bit di abilitazione globale di interrupt GIE, il bit di abilitazione per i dispositivi ausiliari PEIE e il proprio bit di abilitazione di interrupt per questa azione, EEIE. Questo bit si trova sul registro PIE2, mentre nel registro PIR2 troviamo il bit di flag di fine scrittura nella EEPROM, bit EEIF.



Registri che contengono il dato e l'indirizzo della memoria FLASH.

La memoria FLASH

La memoria FLASH contiene il codice di un programma, però, se nel corso di un programma si generano dinamicamente delle informazioni che devono essere memorizzate nella FLASH, permette di farlo senza ricorrere a programmatori esterni. Questo offre due importanti vantaggi: poter riprogrammare l'applicazione in base alle condizioni esterne e ampliare la memoria dei dati non volatili occupando indirizzi della FLASH.

I dati che si scrivono in questa memoria han-

```
bsf     STATUS,RP1      ;Selezioniamo il banco 2 della memoria RAM
bcf     STATUS,RP0
movf    INDIRIZZO,W     ;Carichiamo l'indirizzo sul registro di lavoro
movwf   EEADR           ;Passiamo l'indirizzo al registro EEADR
movf    DATO,W          ;Carichiamo il dato sul registro di lavoro
movwf   EEDATA          ;Lo passiamo al registro EEDATA
bsf     STATUS,RP0      ;Selezioniamo il banco 3 della RAM
bcf     EECON1,EEPGD    ;Selezioniamo di accedere alla EEPROM
bsf     EECON1,WREN     ;Abilitiamo la scrittura nella EEPROM
bcf     INTCON,GIE      ;Proibiamo tutti gli interrupt
movlw   55h             ;Carichiamo questo valore sul registro W
movwf   EECON2          ;Lo passiamo su EECON2
movlw   AAh             ;Carichiamo questo valore sul registro W
movwf   EECON2          ;Lo passiamo su EECON2
bsf     EECON1,WR       ;Ordine di inizio della scrittura
bsf     INTCON,GIE      ;Abilitiamo gli interrupt (PEIE=EEIE=1)
sleep                   ;Entriamo in stato di riposo fino all'interrupt della
                         EEPROM
bcf     EECON1,EEIF     ;Cancelliamo il flag prodotto dall'interrupt
bcf     EECON1,WREN     ;Disabilitiamo la scrittura
```

Programma in cui si esegue una scrittura nella memoria

```
bsf     STATUS,RP1      ;Selezioniamo in banco 2 della memoria RAM
bcf     STATUS,RP0
movf    INDIRIZZO_H,W   ;Carichiamo la parte alta dell'indirizzo su W
movwf   EEADRH          ;Lo spostiamo dal registro W al registro EEADRH
movf    INDIRIZZO_L,W   ;Carichiamo la parte bassa dell'indirizzo su W
movwf   EEADR           ;Lo spostiamo dal registro W al registro EEADR
movf    DATO_H,W        ;Carichiamo la parte alta del dato sul registro di lavoro
movwf   EEDATH          ;La spostiamo sul registro EEDATH
movf    DATO_L,W        ;Carichiamo la parte bassa del dato sul registro di lavoro
movwf   EEDATA          ;La spostiamo sul registro EEDATA
bsf     STATUS,RP0      ;Selezioniamo il banco 3 della RAM
bsf     EECON1,EEPGD    ;Scegliamo di accedere alla FLASH
bsf     EECON1,WREN     ;Abilitiamo la scrittura nella FLASH
bcf     INTCON,GIE      ;Disabilitiamo tutti gli interrupt
movlw   55h             ;Carichiamo questo valore sul registro W
movwf   EECON2          ;Lo spostiamo su EECON2
movlw   AAh             ;Carichiamo questo valore sul registro W
movwf   EECON2          ;Lo spostiamo su EECON2
bsf     EECON1,WR       ;Ordine di inizio della scrittura (impiega 3 cicli)
nop                     ;Questa istruzione non esegue niente
nop
bsf     INTCON,GIE      ;Abilitiamo gli interrupt (PEIE=EEIE=1)
bcf     EECON1,WREN     ;Disabilitiamo le scritture
```

Programma in cui si scrive nella memoria FLASH.

no una lunghezza di 14 bit, quindi, oltre al registro EEDATA, possiamo utilizzare anche EEDATH, per contenere i bit più significativi del dato. La stessa cosa accade con gli indirizzi, dato che nel PIC16F870 abbiamo una capacità di memoria FLASH da 2Kx14, quindi per indirizzare correttamente sono necessari due registri, EEADR e EEADRH.

CP1	CP0	WRT	Indirizzi della FLASH	Lettura interna	Scrittura interna	Lettura ICSP	Scrittura ICSP
0	0	X	Tutta la memoria del programma	Sì	No	No	No
0	1	0	Aree non protette	Sì	No	Sì	No
0	1	0	Aree protette	Sì	No	No	No
0	1	1	Aree non protette	Sì	Sì	Sì	No
0	1	1	Aree protette	Sì	No	No	No
1	0	0	Aree non protette	Sì	No	Sí	No
1	0	0	Aree protette	Sì	No	No	No
1	0	1	Aree non protette	Sì	Sì	Sì	No
1	0	1	Aree protette	Sì	No	No	No
1	1	0	Tutta la memoria del programma	Sì	No	Sì	Sì
1	1	1	Tutta la memoria del programma	Sì	Sì	Sì	Sì

* ICSP è un modo di programmazione

Opzioni di protezione della memoria FLASH.

Lettura e scrittura della memoria FLASH

Per realizzare la gestione delle operazioni di scrittura-lettura in questa memoria utilizziamo gli stessi registri della EEPROM, EECON1 e EECON2, impostando, in questo caso, il registro EEPGD a 1. I processi di lettura e scrittura in questa memoria sono molto simili a quello della EEPROM, anche se, per evitare qualsiasi dubbio, nella figura è riportato un esempio di programma in cui si scrive nella memoria FLASH.

Protezione della memoria FLASH verso scrittura e lettura

Combinando i possibili valori del bit WRT e quelli dei bit CP1 e CP0 della Parola di Configurazione, otterremo diverse alternative per proteggere la memoria FLASH da operazioni di lettura e scrittura. Nella tabella a fianco sono riportate le diverse configurazioni.

Il repertorio delle istruzioni

È arrivato il momento di conoscere le istruzioni del microcontroller per iniziare a mettere in pratica tutte le conoscenze acquisite. Il corretto utilizzo delle istruzioni formerà complesse strutture di controllo, delle quali il linguaggio assembler non dispone direttamente, permettendoci di sviluppare qualsiasi applicazione. Dato che lavoriamo con un processore RISC, le istruzioni sottostanno a una serie di condizioni.

Caratteristiche generali

– Insieme ridotto: esistono solamente 35 istruzioni nel linguaggio assembler di questa gamma del PIC.
– Semplicità e rapidità: la maggioranza si esegue in un solo ciclo di istruzioni e solamente quelle di salto necessitano di due cicli. Il ciclo di istruzione è formato da quattro periodi del clock principale.
– Ortogonalità: la posizione degli operandi gestiti dalle istruzioni è molto flessibile. Qualsiasi oggetto del processore può diventare sorgente o destinazione.

Formato delle istruzioni.

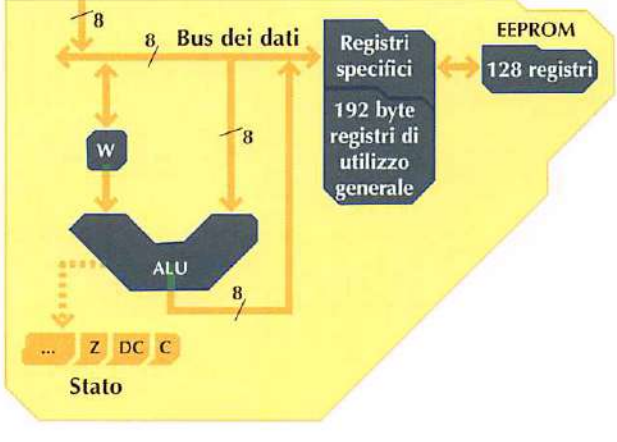
– Formato uniforme delle istruzioni: tutte le istruzioni di questo modello hanno una lunghezza fissa di 14 bit.

È necessario avere una visione chiara della struttura interna del PIC per utilizzare le istruzioni in modo corretto. La ALU o Unità Logico Aritmetica ha il compito di realizzare le operazioni aritmetiche e logiche. Uno dei suoi dati di ingresso arriverà dal registro di lavoro W, che non appartiene alla memoria dei dati, e l'altro può essere un valore letterale fornito come parametro con l'istruzione oppure arrivare dalla memoria dei dati.

Il risultato si può depositare nella memoria dei dati o sul registro W. Nella figura possiamo vedere il percorso che seguono i dati nel microcontroller PIC.

Per quanto riguarda l'indirizzamento nella memoria di codice, il flusso di controllo è stabilito dal Contatore di Programma (PC). Nella maggioranza delle istruzioni il PC si incrementa automaticamente per puntare all'istruzione successiva del programma.

Nelle istruzioni di salto, quando sono di tipo diretto, il valore che si carica sul PC provie-



Percorso dei dati nei microcontroller PIC.

Il PIC16F870 che funziona a 4 MHz esegue un programma di 2.000 istruzioni delle quali il 10% sono di salto. Quanto tempo impiega?

Dato che le istruzioni impiegano un ciclo di 4 istruzioni per essere eseguite (4 cicli di clock), eccetto quelle di salto che ne impiegano 2:

$$T_{osc} = \frac{1}{f} = \frac{1}{4 \cdot 10^6} = 250 \text{ ns}$$

$$T_{ciclo\ istruzione} = 4 \cdot T_{osc} = 1\mu s$$

Il 10% di 2.000 è 200, quindi 1.800 istruzioni verranno eseguite in 1 µs e le rimanenti 200 in 2 µs.

$$T_{TOTALE} = 1800 \cdot 1 + 200 \cdot 2 = 2200\ \mu s = 2{,}2 \text{ ms}$$

Esempio di calcolo del tempo di esecuzione di un programma.

Classificazione delle istruzioni secondo la funzione che realizzano.

ne da una parte dei bit del codice OP dell'istruzione stessa. Nei salti relativi, la ALU somma al valore attuale del PC quello del salto, scrivendo il risultato sul PC.

Classificazione delle istruzioni

Le istruzioni si possono raggruppare o classificare in base a diversi fattori. Delle 35 istruzioni che formano il repertorio del PIC è possibile fare una classificazione associandole in modo da discriminare quelle che operano con i bit da quelle che operano con i byte, in base alla funzione che svolgono, o semplicemente mettendole in ordine alfabetico.

Nel nostro caso analizzeremo le istruzioni in modo raggruppato, tenendo presente la funzione che realizzano. Potremo quindi trovare

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
MOVF f,d	Muove il contenuto del registro f verso una destinazione. Se d=0 la destinazione è W.	1	00 1000 dfff ffff	Z
MOVWF f	Muove il contenuto di W al registro f.	1	00 0000 1fff ffff	---
SWAPF f,d	Si scambiano i quattro bit meno significativi con i più significativi del registro f e si scrive il risultato nella destinazione (W se d=0).	1	00 1110 dfff ffff	---
MOVLW k	Muove il valore letterale k al registro W.	1	11 00xx kkkk kkkk	---

Istruzioni di trasferimeno

gruppi di istruzioni mostrati nella figura della pagina precedente.

Istruzioni di trasferimento

È il gruppo di istruzioni più utilizzato e il suo compito principale è quello di trasferire informazioni da e verso i registri dell'area dei dati. In tutti i programmi avremo a che fare con istruzioni di questo tipo quando dovremo trasferire delle informazioni. Anche quando vorremo configurare qualche risorsa interna del nostro PIC o lavorare internamente con i dati, dovremo far ricorso a queste istruzioni. Nella tabella della pagina precedente possiamo vedere le quattro istruzioni che fanno parte del gruppo di istruzioni di trasferimento.

```
Programma che configura le
quattro linee più significative della porta C come ingresso
e le quattro meno significative come uscita.
        MOVLW   b'11110000'
        MOVWF   TRISC
Cambiare le linee della porta C, gli ingressi con le uscite e
viceversa.
        SWAPF   TRISC,1
```

Esempio di utilizzo delle istruzioni di trasferimento.

MOVF f,d: (Move f) Il contenuto del registro 'f' viene trasferito alla destinazione in funzione del valore di 'd'. Se d = 0 la destinazione sarà il registro di lavoro W e se d = 1 la destinazione sarà lo stesso registro 'f'. Quest'ultima opzione si utilizza solamente per testare o verificare il valore del registro. Il flag Z del registro STATUS può essere attivato da questa istruzione.

MOVWF f: (Move W to f) Il contenuto del registro di lavoro W è trasferito sul registro indicato dall'operando dell'istruzione.

SWAPF f,d: (Swap Nibbles in f) Questa istruzione scambia i nibbles (insieme di quattro bit) del registro 'f'. Vengono scambiati di posizione i quattro bit più significativi con i quattro bit meno significativi. Se d = 0 il risultato verrà lasciato sul registro di lavoro W, invece se d = 1 il risultato verrà scritto sullo stesso registro 'f'.

MOVLW k: (Move Literal to W) Il valore let-

MNEMONICO E OPERANDI		DESCRIZIONE	CICLI	CODICE OP	FLAG
ADDLW	k	Somma a W il valore del letterale k e lascia il risultato in W	1	11 111x kkkk kkkk	C, DC, Z
ADDWF	f,d	Somma il contenuto di W al registro f. Se d = 0 il risultato si lascia su W	1	00 0111 dfff ffff	C, DC, Z
SUBLW	k	Sottrae al valore letterale k il valore contenuto nel registro W, lasciando il risultato in W	1	11 110x kkkk kkkk	C, DC, Z
SUBWF	f,d	Sottrae il valore del registro f al valore di W e lascia il risultato dove indica d	1	00 0010 dfff ffff	C, DC, Z
INCF	f,d	Incrementa f		00 1010 dfff ffff	Z
DECF	f,d	Decrementa f	1	00 0011 dfff ffff	Z
COMF	f,d	Complemento di f	1	00 1001 dfff ffff	Z
RLF	f,d	Ruota il registro f a sinistra tramite il carry (C)	1	00 1101 dfff ffff	C
RRF	f,d	Ruota il registro f a destra tramite il carry (C)	1	00 1100 dfff ffff	C

Istruzioni aritmetiche.

Calcolare il valore con il quale rimane caricato il registro W dopo aver eseguito il seguente programma:

```
MOVLW   F0h
INCF    W,0
ADDLW   0Ah
RRF     W,0
SOLUZIONE:  W = 0111 1101
```

Esempio 1 di utilizzo delle istruzioni aritmetiche.

terale 'k' da otto bit si trasferisce sul registro di lavoro W.

Nell'esempio alla pagina precedente possiamo vedere come si utilizzano queste istruzioni. Vi consigliamo di rivedere gli esempi eseguiti finora e analizzare ognuna delle istruzioni di trasferimento utilizzate.

Istruzioni aritmetiche

Le istruzioni di questo gruppo realizzano diverse operazioni aritmetiche tramite la ALU: somma, sottrazioni, complementi, incrementi, decrementi e rotazioni. In assembler non c'è nessuna istruzione né operando che ci permetta direttamente altre istruzioni tipiche come quelle della moltiplicazione o della divisione.

Le operazioni aritmetiche sono eseguite dalla ALU (Unità Logico Aritmetica) prendendo un dato dal registro di lavoro W e un altro dal registro della memoria RAM o da un valore letterale.

Se vogliamo sommare il dato di un registro con quello di un altro registro dobbiamo prima spostare il contenuto di uno dei due registri sul registro di lavoro e successivamente aggiungere il valore dell'altro registro. Lo stesso succede quando vogliamo scrivere il risultato in un registro che non è coinvolto nell'operazione. Dobbiamo passare il risultato del registro di lavoro al registro desiderato.

Calcolare il valore che contiene il registro W dopo aver eseguito il seguente programma:

```
MOVLW   B7h
SWAPF   W,0
DECF    W,0
SUBLW   FEh
SOLUZIONE:  W = 1000 01
```

Esempio 2 di utilizzo delle istruzioni aritmetiche.

ADDLW k: (Add Literal and W) Somma i valori del registro W e del letterale specificato. Il risultato si memorizza sul registro di lavoro W.

ADDWF f,d: (Add W and f) Somma il contenuto del registro di lavoro e il contenuto del registro 'f'. Se d = 0 il risultato si scriverà su W e se d = 1 si scriverà sullo stesso registro f.

SUBLW k: (Subtract W from Literal) Sottrae al valore letterale il valore contenuto nel registro W. Il risultato si scrive su W.

SUBLW f,d: (Subtract W from f) Sottrae al valore contenuto sul registro f il valore contenuto in W. Come per la somma, il risultato si scriverà in base al valore di 'd'.

Queste quattro istruzioni possono modificare il valore del registro di stato STATUS. Il bit Z passerà a 1 quando il risultato dell'operazione sarà uguale a 0. Il bit C (Carry o riporto) passerà a 1 nelle operazioni di somma, sempre che ci sia un riporto, restando a 0 quando il risultato è contenuto su un registro da 8 bit. Nelle operazioni di sottrazione, se sottraiamo A-B e A è maggiore di B, non ci sarà riporto, quindi il bit C passerà a 1, se A è minore di B, ci sarà riporto e il bit C passerà a 0 (al contrario della somma). Il bit DC funziona come C però per i numeri BCD.

Conclusioni

Continuiamo l'analisi del repertorio delle istruzioni, terminando di vedere le istruzioni aritmetiche e facendo pratica con i diversi esempi prima di passare a studiare il resto delle istruzioni. Vi consigliamo di esercitarvi con esempi simili a quelli presentati in precedenza, oltre che osservare i programmi contenuti nel CD e quelli realizzati finora, per comprendere meglio l'utilizzo e il significato delle istruzioni.

Il repertorio delle istruzioni (II)

Delle 35 istruzioni che formano il repertorio del nostro PIC abbiamo analizzato quelle di trasferimento e parte di quelle aritmetiche. Continueremo lo studio dei nuovi gruppi di istruzioni, inserendo degli esercizi pratici per consolidare le nostre conoscenze.

Istruzioni aritmetiche

Questo tipo di istruzioni realizza operazioni aritmetiche tramite la ALU. Nella tabella della figura sono riportate le istruzioni che formano questo gruppo. Abbiamo analizzato quelle di somma e sottrazione e ora continuiamo con il resto.

INCF f,d: (Increment f) Il valore del registro f si incrementa di una unità. Se d = 0 il risultato dell'operazione verrà scritto sul registro di lavoro W, se d = 1 il risultato sarà lo stesso registro.

DECF f,d: (Decrement f) Come l'istruzione precedente però, invece di incrementare, riduce il valore di una unità.

COMF f,d: (Complement f) Esegue il complemento del registro f, si cambiano i valori 1 con 0 e viceversa (negazione o complemento a uno). In funzione del valore di 'd' il risultato si scriverà sul registro W o nel proprio registro 'f'.

Questi tre registri modificano il bit Z del Registro di Stato, quindi, realizzando una di queste operazioni, se il risultato è 0 il bit Z passerà a valore 1.

RLF f,d: (Rotate Left f) Ruota I bit del registro 'f' di una posizione verso sinistra (la stessa cosa che moltiplicare per due il valore del registro). Il valore di 'd' determinerà dove si registra il risultato.

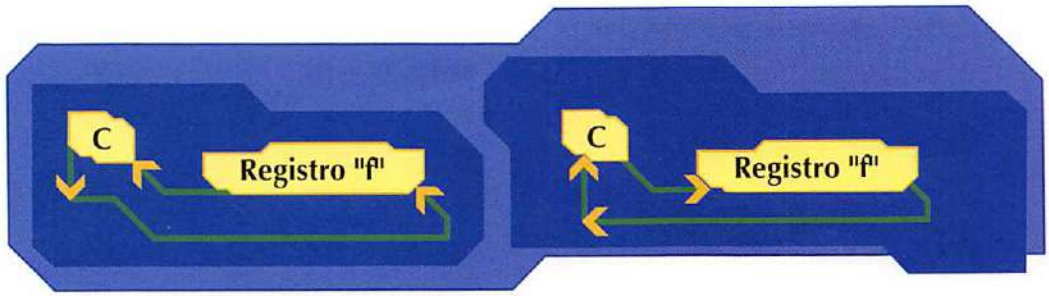
RRF f,d: (Rotate Right f) Come l'istruzione precedente, con la differenza che ruota i bit verso destra.

Nella figura possiamo vedere come funziona l'istruzione e come influenza il bit C del Registro di Stato.

Per operazioni complesse come la moltiplicazione e la divisione, dobbiamo scomporre queste in operazioni semplici. Qundi, una moltiplicazione altro non è che un ciclo dove si ripete la somma del dato del moltiplicando il numero di volte indicato dal moltiplicatore. Con la divisione avviene la stessa cosa, però si

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
ADDLW k	Somma a W il valore del letterale k e lascia il risultato su W	1	11 111x kkkk kkkk	C, DC, Z
ADDWF f,d	Somma il contenuto di W al registro f. Se d = 0 il risultato si scrive su W	1	00 0111 dfff ffff	C, DC, Z
SUBLW k	Sottrae al valore letterale k il valore contenuto sul registro W lasciando il risultato su W	1	11 110x kkkk kkkk	C, DC, Z
SUBWF f,d	Sottrae al valore del registro f il valore di W e lascia il risultato dove indica d	1	00 0010 dfff ffff	C, DC, Z
INCF f,d	Incrementa f		00 1010 dfff ffff	Z
DECF f,d	Decrementa f	1	00 0011 dfff ffff	Z
COMF f,d	Complemento di f	1	00 1001 dfff ffff	Z
RLF f,d	Ruota il registro f a sinistra utilizzando il riporto (C)	1	00 1101 dfff ffff	C
RRF f,d	Ruota il registro f a destra utilizzando il riporto (C)	1	00 1100 dfff ffff	C

Istruzioni aritmetiche.



Istruzioni aritmetiche di rotazione.

utilizza il resto invece della somma. Anche altre operazioni complesse dovranno essere scomposte nelle operazioni più semplici.

Esempio pratico

Abbiamo tre valori, A, B e C, scritti nell'indirizzo di memoria dei dati 0x20, 0x21 e 0x22, rispettivamente. Dobbiamo realizzare l'operazione aritmetica (A+B)-C e depositare il risultato all'indirizzo 0x23.

A questo punto dobbiamo saper risolvere questo esercizio. Apriremo l'editor di testo di MPLAB e inizieremo a scrivere il nostro programma. Ricordate che la prima cosa da fare è quella di definire nel programma quale microcontroller dobbiamo utilizzare e includere il file o la libreria dove si trovano tutte le etichette dei registri.

Fatto questo, è necessario definire le variabili da utilizzare. Per semplificare il programma, invece di lavorare con gli indirizzi, lavoriamo con etichette, quindi le dobbiamo associare ai loro indirizzi di memoria corrispondenti.

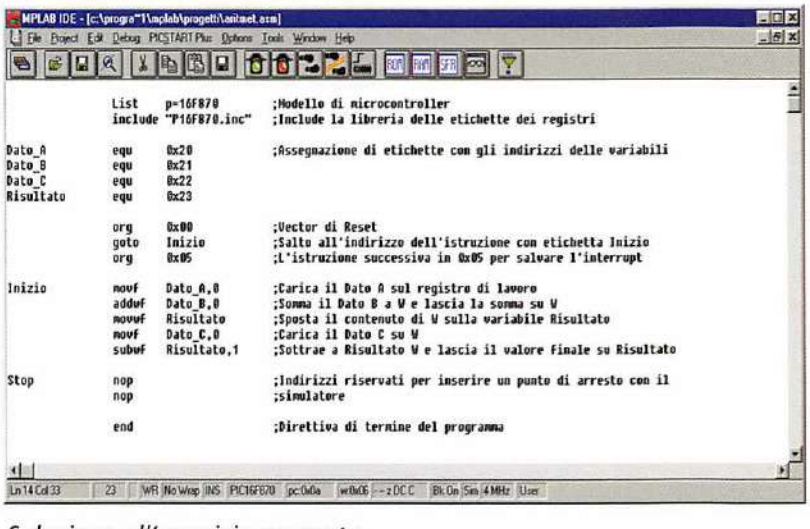
Il passo successivo consiste nel porre le direttive di inizio del programma (Vector di Reset e salto della posizione della routine di interrupt).

Di seguito risolveremo il problema mediante istruzioni di trasferimento e aritmetiche, concludendo il programma con istruzioni di non operazione e con la direttiva end.

Il programma risolto avrà l'aspetto che possiamo vedere nella figura. Come possiamo notare l'uso dei commenti semplifica molto la comprensione del programma.

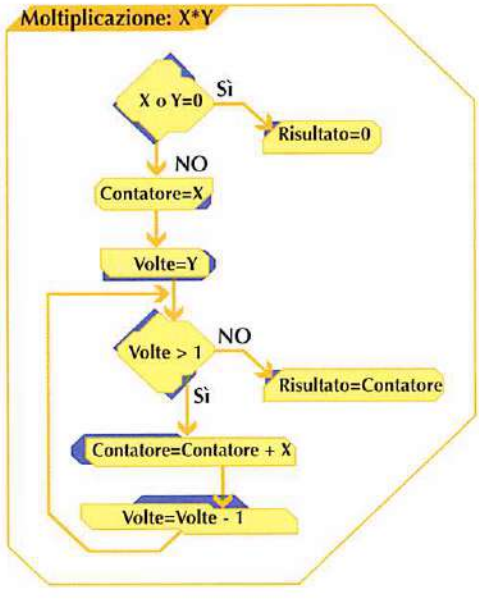
Per verificare che il programma faccia realmente ciò che è stato specificato nel risultato dobbiamo compilare il codice (prima dobbiamo associarlo a un progetto) e dopo aver verificato che non ci siano errori apriremo le finestre di simulazione.

In questo caso è stata presa in considerazione la finestra degli Special Function Register e una finestra dove visualizzare le variabili con cui vogliamo lavorare.



Soluzione all'esercizio proposto.

Algoritmo per risolvere una moltiplicazione mediante somme.



Per inserire valori sulle variabili utilizziamo la finestra Modify, come mostrato nella figura (inseriamo i valori in decimale).

Assegnando i valori alle variabili A, B e C (8, 5 e 6 rispettivamente) procederemo all'esecuzione passo a passo. In questo modo potremo osservare l'evoluzione dei registri e se abbiamo risolto bene il programma.

Istruzioni logiche

Questo gruppo di istruzioni comprende quelle che eseguono alcune operazioni di tipo logico. Anche se i parametri di queste istruzioni sono registri, le operazioni si realizzano bit a bit. Le operazioni logiche che potremo realizzare sono: AND, OR e XOR. Queste tre operazioni si realizzano con il registro di lavoro e un valore letterale o il valore contenuto su un registro. Per questa ragione ci sono due istruzioni per ogni operazione.

ANDLW k: (AND Literal with W) È realizzata l'operazione logica AND tra i due bit del registro di lavoro W e un valore letterale. Il risultato si deposita su W.

ANDWF f,d: (AND W with f) La stessa operazione logica però, questa volta, tra W e il valore contenuto sul registro 'f'. Il risultato si depositerà su W o su 'f' in base al valore di 'd'.

IORLW k: (Inclusive OR literal with W) Si realizza l'operazione logica OR tra i bit del Registro di Lavoro W e un valore letterale. Il risultato si deposita su W.

IORWF f,d: (Inclusive OR W with f) Somma logica OR tra W e il valore contenuto nel registro 'f'. Il risultato si depositerà in W o in 'f' in base al valore di 'd'.

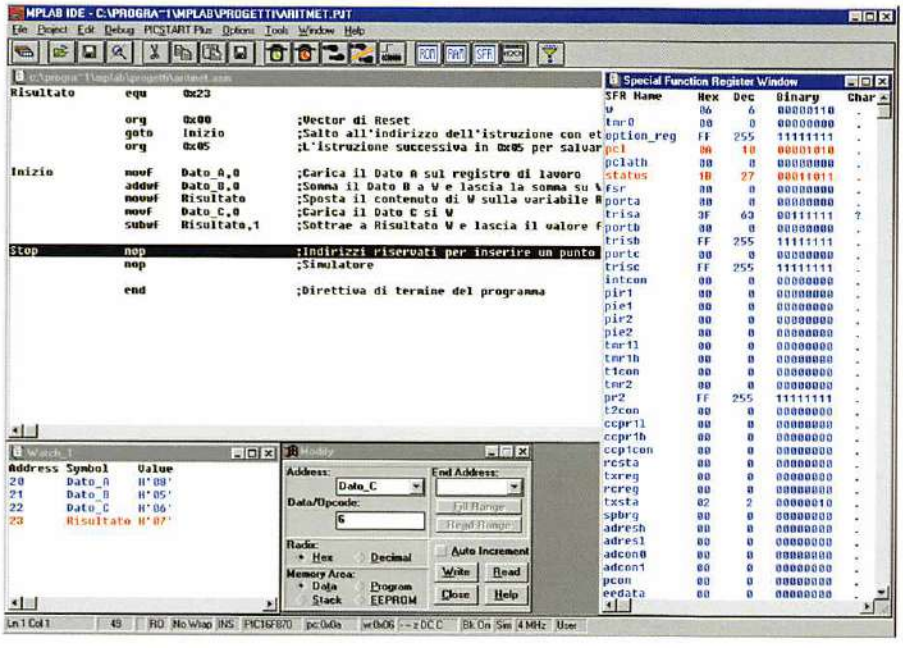
XORLW k: (Exclusive OR literal with W) OR esclusivo tra due bit del Registro di Lavoro W e un valore letterale. Il risultato si lascia su W.

XORWF f,d: (Exclusive OR W with f) Uguale al precedente però tra W e il valore contenuto nel registro 'f'. Il risultato si depositerà in W o in 'f', in funzione di 'd'.

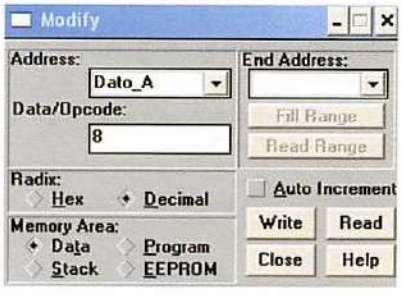
Nelle operazioni logiche i riporti non passano da un bit all'altro come succede con quelle aritmetiche. Le operazioni logiche si realizzano bit a bit, indipendentemente dai bit contigui.

L'unico bit del Registro di Stato STATUS che viene influenzato da queste istruzioni è il bit Z (Zero). Quando si esegue una di queste istruzioni se il risultato dell'operazione è uguale a 0, il bit Z passerà a 1.

Nella figura è presentato un semplice esempio di combinazione di istruzioni in cui si realizza un'operazione logica.



Verifichiamo come cambiano i registri W e la variabile. Risultato eseguendo passo a passo il codice del programma.



Finestra Modify per forzare valori sulle variabili.

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
ANDLW k	Moltiplicazione logica del valore di W e il valore del letterale k. Lascia il risultato su W	1	11 1001 kkkk kkkk	Z
ANDWF f,d	Moltiplicazione logica del valore del registro f e del valore di W. Lascia il risultato in funzione di 'd'	1	00 0101 dfff ffff	Z
IORLW k	Somma logica del valore W con il valore del letterale k. Lascia il risultato in W	1	11 1000 kkkk kkkk	Z
IORWF f,d	Somma logica del valore del registro f con il valore di W. Lascia il risultato dove indica d	1	00 0100 dfff ffff	Z
XORLW k	Somma esclusiva del valore di W con il valore del letterale k. Lascia il risultato su W		11 1010 kkkk kkkk	Z
XORWF f,d	Somma esclusiva del valore del registro f con il valore di W. Lascia il risultato dove indica d	1	00 0110 dfff ffff	Z

Istruzioni logiche.

Esempio pratico

Nella figura riprodotta qui sopra è riportato un nuovo esempio di programma con istruzioni logiche.

Copiate sull'editor di MPLAB l'esempio della figura. I passi da realizzare sono gli stessi visti nell'esempio delle istruzioni aritmetiche. È necessario creare un progetto, associare a questo il codice in assembler e compilarlo. Successivamente osserveremo come si modificano i registri coinvolti nel programma man mano che questo si esegue, in questo modo si capirà meglio il funzionamento delle istruzioni.

È raccomandabile fare pratica con diversi esercizi e familiarizzare sia con le istruzioni che con il software di simulazione MPLAB.

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

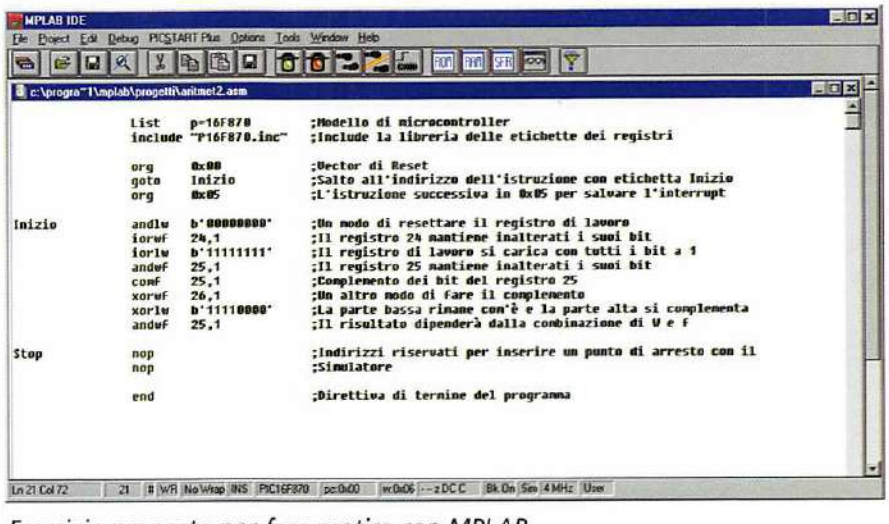
Operazioni logiche AND, OR e XOR.

Calcolare il valore con cui deve rimanere caricato il registro di lavoro W dopo aver eseguito il seguente programma:

```
MOVLW   F0h
INCF    W,0
XORLW   3Ah
RRF     W,0
```

SOLUZIONE: W = 0110 0101

Esempio di combinazione di istruzioni.



Esercizio proposto per fare pratica con MPLAB.

Il repertorio delle istruzioni

Continuiamo la presentazione delle istruzioni del microcontroller per fare in modo che il PIC cominci a interagire con l'hardware del nostro Laboratorio Digitale. In questa sezione vedremo le istruzioni di impostazione a zero e quelle di salto, facendo pratica con degli esempi.

Istruzioni di impostazione a zero

Questo gruppo di istruzioni si chiama così perché la funzione che esse svolgono è quella di impostazione a zero o cancellazione di un registro. Sono molto utili quando vogliamo resettare un valore, come ad esempio, quando eseguiamo un conteggio di tempo e trascorso il tempo desiderato, vogliamo reinizializzare il valore del conteggio. In questo gruppo sono comprese tre istruzioni:

- CLRF f: (Clear f) Il contenuto del registro f si cancella, impostiamo a zero tutti i bit del registro. Il bit Z del Registro di Stato passa automaticamente a 1.

- CLRW: (Clear W) Si cancella il contenuto del registro di lavoro W. Dato che questo registro è utilizzato in tutti i programmi è molto comodo che esista un'istruzione specifica per resettarlo. Come nell'istruzione precedente, il bit Z passa automaticamente a 1.

- CLRWDT f: (Clear Watchdog Timer) Questa istruzione resetta il valore del temporizzatore del Watchdog e il valore del suo prescaler o divisore di frequenza. Coinvolge il registro STATUS attivando (impostando a 1) i bit $\overline{TO}$ e $\overline{PD}$

Istruzioni di salto

Le istruzioni di salto rompono la normale frequenza del flusso di istruzioni del programma, provocando dei salti. Quindi queste istruzioni coinvolgono il contenuto del Contatore di Programma (PC).

In questo gruppo si possono includere le istruzioni BTFSC e BTFSS, dato che saltano un'istruzione se si compie una condizione, però abbiamo preferito inserirle nel gruppo successivo: istruzioni di gestione dei bit. Le istruzioni appartenenti al gruppo di salto sono:

- CALL k: (Call Subroutine) Chiamata a una subroutine. Prima si memorizza sullo stack l'indirizzo di ritorno, quello dell'istruzione successiva a questa (PC+1). Gli 11 bit dell'indirizzo di destinazione 'k' passano al PC <10:0>. I bit più significativi del PC si caricano mediante PCLATH. Il tempo di esecuzione di questa istruzione è di due cicli di istruzioni.

- RETLW k: (Return with Literal in W) Si carica il registro di lavoro W con il valore numerico k. Il PC acquisisce il valore dello stack (l'indirizzo di ritorno). Occupa due cicli di istruzione.

- RETFIE: (Return from Interrupt) Istruzione per ritornare da un interrupt. Si carica il PC con l'indirizzo della cima dello stack e si imposta a 1 il bit GIE di abilitazione globale degli interrupt.

- RETURN: (Return from Subroutine) Istruzione di due cicli di durata la cui funzione è quella di ritornare da una subroutine.

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
CLRF f	Resetta il valore del registro f	1	00 0001 1fff ffff	Z
CLRW	Resetta il valore del registro W	1	00 0001 0xxx xxxx	Z
CLRWDT	Resetta il valore del Watchdog Timer e del suo divisore di frequenza	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$

Istruzioni di impostazione a zero.

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
CALL k	Salto alla subroutine	2	10 0kkk kkkk kkkk	--
RETLW k	Ritorno da una subroutine e registro W caricato con il valore numerico K	2	11 01xx kkkk kkkk	--
RETFIE	Ritorno da un interrupt	2	00 0000 0000 1001	--
RETURN	Ritorno da una subroutine	2	00 0000 0000 1000	--
GOTO k	Salto incondizionale all'indirizzo indicato da k	2	10 1kkk kkkk kkkk	--
DECFSZ f,d	Decrementa il valore di f. Se il risultato è 0 salta l'istruzione successiva	1 o 2	00 1011 dfff ffff	--
INCFZ f,d	Incrementa il valore di f. Se il risultato è 0 salta l'istruzione successiva	1 o 2	00 1111 dfff ffff	--

Istruzioni di salto.

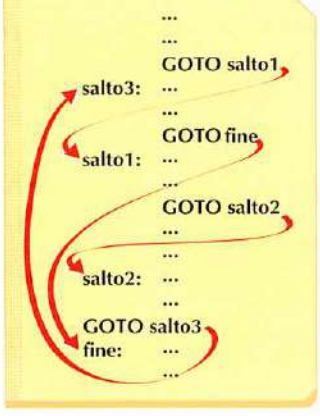
- GOTO k: Salto incondizionale. Il valore immediato di 11 bit si carica sul PC (bit <10:0>). I bit più significativi del PC si caricano tramite il registro PCLATH<4:3>. Questa istruzione risulta molto utile ed è assai utilizzata in assembler, però è un'istruzione che nel resto dei linguaggi di programmazione non si utilizza o se ne limita l'uso.

Quando eseguiamo un salto incondizionale rendiamo più complessa la struttura del programma, dato che saltiamo a un altro gruppo di istruzioni indicato mediante l'etichetta definita tramite il valore letterale, e lasciamo da eseguire le istruzioni successive a questa, a meno di non ritornare successivamente mediante un altro salto all'esecuzione sequenziale del programma. Questa istruzione non è adatta a un linguaggio di programmazione strutturato.

- DECFSZ f,d: Il contenuto del registro 'f' si riduce. In base al valore 'd' il risultato verrà scritto su W (d = 0) o sul registro 'f' (d = 1). Se il risultato è uguale a 0, si salta l'istruzione successiva e si continua con l'esecuzione.

- INCFSZ f,d: Il contenuto del registro 'f' si incrementa. Come nell'istruzione precedente, il risultato è scritto su W o sul registro 'f' in base al valore di 'd'. Se il risultato è uguale a 0, si salta l'istruzione successiva e si continua con l'esecuzione.

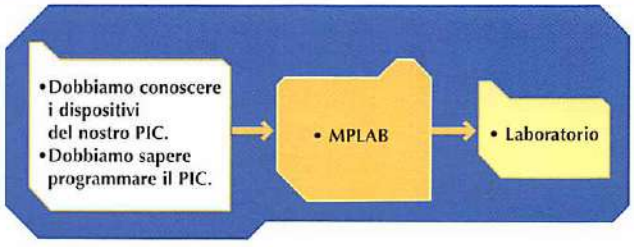
Programma con diversi salti incondizionali, il programma perde la sua struttura.



Calcolare il valore che contiene il registro VALORE dopo aver eseguito il seguente programma:

```
        MOVLW   02h
        MOVWF   VALORE
        DECF    VALORE,1
        DECFSZ  VALORE,1
        INCF    VALORE,1
        INCF    VALORE,1
SOLUZIONE:  VALORE = 0000 0001
```

Esempio di base dell'utilizzo delle istruzioni.



In un progetto realizzeremo il programma, la sua simulazione e il montaggio finale del circuito.

Esercizi con MPLAB

Normalmente le nozioni teoriche sono affiancate da esempi. In questo modo, oltre a mettere in pratica le nuove conoscenze acquisite, è possibile imparare a utilizzare il software MPLAB con maggiore disinvoltura. È consigliabile modificare gli esempi e giocare con i programmi, come anche provare le diverse opzioni del software di simulazione. Inoltre, poco a poco, inizieremo a interagire con l'hardware del laboratorio, fino a quando saremo in grado di realizzare senza difficoltà tutte le fasi di sviluppo di un progetto.

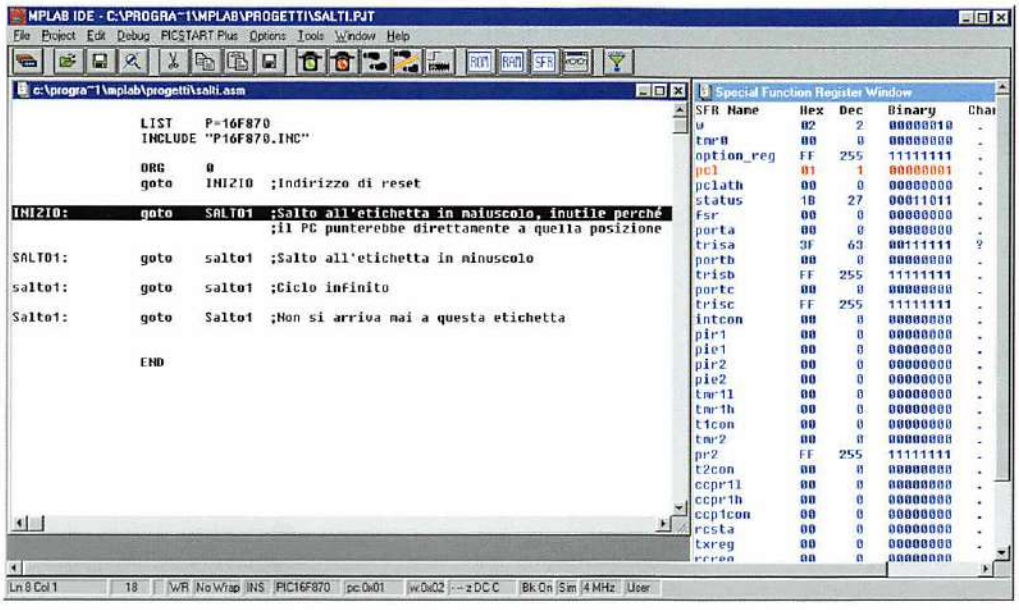
Il primo esempio con cui ci eserciteremo è una dimostrazione di come funziona l'istruzione GOTO. Eseguite MPLAB e create un progetto con il nome, ad esempio, "salti.pjt". Ora create una nuova finestra di scrittura, scrivete il programma riportato nella figura e salvatelo con il nome "salti.asm". Associate il codice

```
c:\progra~1\mplab\progetti\salti.asm

                LIST    P=16F870
                INCLUDE "P16F870.INC"

                ORG     0
                goto    INIZIO  ;Indirizzo di reset

INIZIO:         goto    SALTO1  ;Salto all'etichetta in maiuscolo, inutile perché
                                ;il PC punterebbe direttamente a quella posizione

SALTO1:         goto    salto1  ;Salto all'etichetta in minuscolo

salto1:         goto    salto1  ;Ciclo infinito

Salto1:         goto    Salto1  ;Non si arriva mai a questa etichetta

                END
```

Programma per fare pratica con i salti incondizionali.

al progetto Project → Edit Project → Add Node. Fatto questo, compilate e assemblate il programma, eseguendolo passo a passo per studiarne il funzionamento. Nella figura si può vedere come rimarrà la videata di MPLAB al momento di simulare. In questo esempio il simulatore è sensibile all'utilizzo di maiuscole o minuscole, però questo si può cambiare nelle opzioni del programma.

Il programma successivo con cui ci eserciteremo è un programma dedicato a riempire e gestire un insieme di indirizzi di memoria. Utilizzando l'indirizzamento indiretto (movwf INDF) si utilizza l'indirizzo contenuto sul registro FSR per puntare il dato da muovere. Si tratta di scrivere il valore 0x18 nelle 15 posizioni di dati contigue, iniziando dall'indirizzo 0x20.



Simulazione del programma precedente dei salti con GOTO.

```
c:\progra~1\mplab\progetti\memoria1.asm

                List    P=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizione dei registri interni

Contatore       equ     0x0c            ;Contatore interno (c(hex)=12(decimale))
Prima           equ     0x20            ;Posizione iniziale

                org     0x00            ;Vector di reset
                goto    Inizio

                org     0x05

Inizio          movlw   .15
                movwf   Contatore       ;Carica il contatore con il valore 15 in decimale
                movlw   Prima
                movwf   FSR             ;Inizializza il puntatore con l'indirizzo iniziale
                movlw   0x18            ;Carica il valore da scrivere

Ciclo           movwf   INDF            ;Scrive il valore all'indirizzo indicato da FSR
                incf    FSR,F           ;Incrementa il puntatore FSR
                decfsz  Contatore,F     ;Decrementa il contatore sino ad arrivare a 0
                goto    Ciclo           ;Esegue l'istruzione se il contatore non è a 0

Stop            nop
                nop

                end                     ;Fine del programma sorgente
```

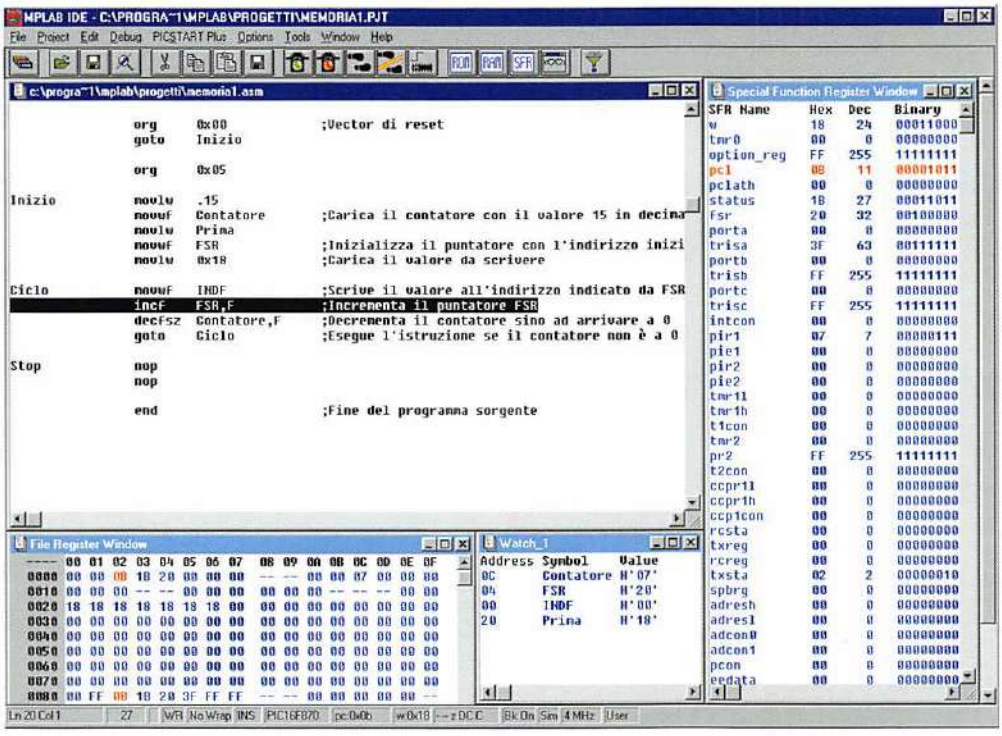
Esempio di gestione di memoria.

La procedura da seguire è la stessa dell'esempio precedente. Potete nominare il progetto "memoria1.pjt" e il codice in assembler con lo stesso nome ma con l'estensione ".asm". Nella figura è riportato il codice che risolve il problema.

Come vedrete, in questo programma gestiamo le istruzioni di salto goto e decfsz. Normalmente nelle istruzioni incrementa/decrementa e salta; la struttura da seguire è quella mostrata in questo esempio: l'istruzione di incremento/decremento seguita da una di salto incondizionale. Potete vedere come mediante l'utilizzo di semplici istruzioni è possibile creare strutture complesse di controllo e cicli condizionali. Anche se il suo utilizzo non è consigliato in nessun linguaggio di programmazione, l'istruzione goto è necessaria nel nostro caso per poter realizzare strutture di controllo.

Nella figura sono riportate le finestre che abbiamo aperto per realizzare la simulazione.

Osservate come si riempiono le posizioni di memoria con il valore desiderato e al riempimento di quelle richieste, l'istruzione decfsz agisce ed evita di entrare nuovamente in ciclo, saltando l'istruzione goto.



Videata di MPLAB.

È possibile modificare il programma per vedere cosa succede. Le modifiche si devono realizzare una a una per osservare meglio il comportamento del programma.

Di seguito vi diamo alcuni suggerimenti:

– definite le variabili nelle zone di memoria occupate, ad esempio Contatore in 0x0c;

– cambiate i dati nel programma;

– ampliate il programma e fate pratica con tutte le modifiche che servono, cercando di inserire al massimo le istruzioni che abbiamo viste finora.

Il repertorio delle istruzioni (IV)

Terminiamo di analizzare il repertorio delle istruzioni del nostro microcontroller con quelle di gestione dei bit e delle istruzioni speciali. Visto questo e facendo pratica con i diversi esempi, non dovremo più aver paura di programmare e saremo capaci di realizzare qualsiasi progetto per complicato che possa apparire.

Istruzioni per la gestione dei bit

Questo insieme di istruzioni ha il compito di verificare il valore di un bit particolare di un registro, di impostarlo a 1 oppure a 0, e anche di produrre un salto di un'istruzione, a seconda se si compie o meno la condizione stabilita.

Nella tabella possiamo vedere le quattro istruzioni che formano questo gruppo.

Impostazione a 0 oppure a 1 di un bit

Le prime due istruzioni servono per l'impostazione a 0 oppure a 1 di un bit di un registro. Un esempio di quanto utile possa risultare avere istruzioni specifiche per questo compito è quando dobbiamo cambiare banco di memoria per lavorare con un registro o scrivere dei dati, o se vogliamo definire un bit di una porta come ingresso o uscita.

BCF f,b: (Bit Clear f) Il bit 'b' contenuto nel registro f si cancella, ovvero si imposta a zero.

BSF f,b: (Bit Set f) Il bit 'b' contenuto nel registro f si setta, ovvero si imposta a uno.

Gruppi di istruzioni che formano il repertorio.

Salti dipendenti da un bit

Le due istruzioni successive sono salti dipendenti da un bit. Sono state spiegate nel fascicolo precedente e, anche se i loro nomi possono sembrare complicati da imparare, lo mnemonico stesso dell'istruzione ci indica il loro compito.

Quindi la "b" fa riferimento a un bit, la "t" significa test (verifica di un valore), la "f" è il nome dei registri in assembler, "s" significa salto

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
BCF f,b	Impostazione a 0 del bit b del registro f	1	01 00bb bfff ffff	— —
BSF f,b	Impostazione a 1 del bit b del registro f	1	01 01bb bfff ffff	— —
BTFSC f,b	Salta un'istruzione se il bit b del registro f ha valore 0	1 (2)	01 10bb bfff ffff	— —
BTFSS f,b	Salta un'istruzione se il bit b del registro f ha valore 1	1 (2)	01 11bb bfff ffff	— —

Istruzioni di gestione dei bit.

RP1	RP0	BANCO
0	0	0
0	1	1
1	0	2
1	1	3

Bit RP0 e RP1 del registro STATUS per selezionare il banco di memoria.

(skip), e la terminazione in "c" indicherà 0 (clear) e in "s" 1 (set). La traduzione sarà: "testa il valore di un bit e salta se è 1 (oppure 0)". Mediante queste due istruzioni potremo formare istruzioni di controllo e, grazie alla loro utilità, molto spesso le troveremo implementate nel linguaggio. Negli esempi con cui faremo pratica prossimamente potremo verificarne l'utilità, ma in questo caso, quello dell'allarme di un'automobile, ci serve come esempio veloce. Quando si rileva un'intrusione un bit cambia di stato, normalmente si imposta a 1, e attraverso questa istruzione si determinerà cosa fare nei due stati possibili, come si può vedere nella figura.

BTFSC f,b: (Bit Test, Skip if Clear) Se il bit 'b' del registro 'f' ha valore 1 si esegue l'istruzione successiva. In caso contrario, l'istruzione successiva è saltata e si esegue un'istruzione NOP, facendo quindi in modo che l'istruzione duri due cicli.

BTFSS f,b: (Bit Test, Skip if Set) Come l'istruzione precedente, saltando però l'istruzione successiva nel caso in cui il bit testato sia uguale a 1.

Istruzioni speciali

All'interno di questo gruppo troviamo due istruzioni le cui funzioni sono molto specifiche, quindi non si possono inserire in nessuno dei gruppi precedenti. Le possiamo vedere nella tabella della figura.

NOP: (No Operation) Questa istruzione non ha alcuna funzione. Serve unicamente a far passare un ciclo di istruzione in cui il microprocessore non esegue alcun lavoro.

SLEEP: Quando si esegue questa istruzione il microprocessore entra in modo riposo o di basso consumo.

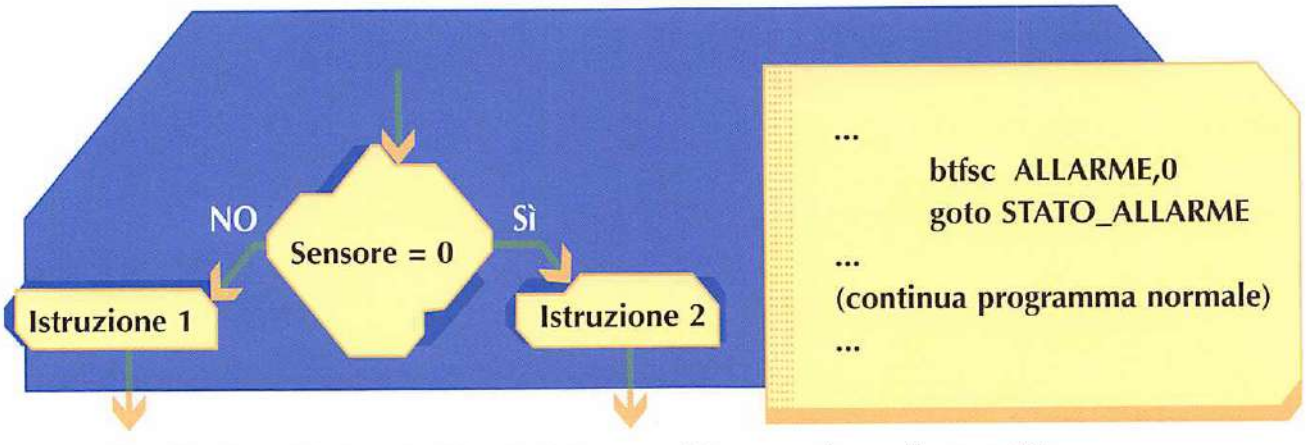
Il bit $\overline{PD}$ (Power-Down) del Registro di Stato si imposta a 0 e il bit $\overline{TO}$ (Time-Out) a 1. Anche il Watchdog e il suo predivisore di frequenza sono impostati a zero.

Facciamo pratica con gli esempi

Abbiamo terminato di vedere il repertorio delle istruzioni del nostro microcontroller, però il miglior sistema per assimilare i nuovi concetti teorici è metterli in pratica, eseguiamo quindi questi diversi esempi.

Esempio 1

Consiste nell'indicare il compito del programma della figura e il tempo di esecuzione impiegato in un PIC 16F870 a 4 MHz.



Esempio di applicazione delle istruzioni di salto in base a un bit con un allarme di automobile.

MNEMONICO E OPERANDI	DESCRIZIONE	CICLI	CODICE OP	FLAG
NOP	Non fa nulla	1	00 0000 0xx0 0000	— —
SLEEP	Porta il processore nel modo riposo o di basso consumo	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$

Istruzioni speciali.

Soluzione: La prima cosa da fare è resettare il registro di lavoro W, impostandolo a 0. Dopo faremo un'operazione OR con il valore binario 10101011 (AB in esadecimale), e lasciamo caricato W con questo valore. Dopodiché eseguiremo un test sul bit 3 del registro W e, dato che vale 1, saltiamo l'istruzione CLRW che avrebbe portato a 0 il registro W, e carichiamo su di esso il valore FFh (11111111 in binario).

Dato che sono cinque istruzioni e una di esse impiega due cicli per essere eseguita, il programma completo, per essere eseguito, impiegherà 6 µs dato che il ciclo di istruzioni è:

$$4 \cdot \frac{1}{4.10^6\ \text{Hz}} = 1\ \mu\text{s}$$

Esempio 2

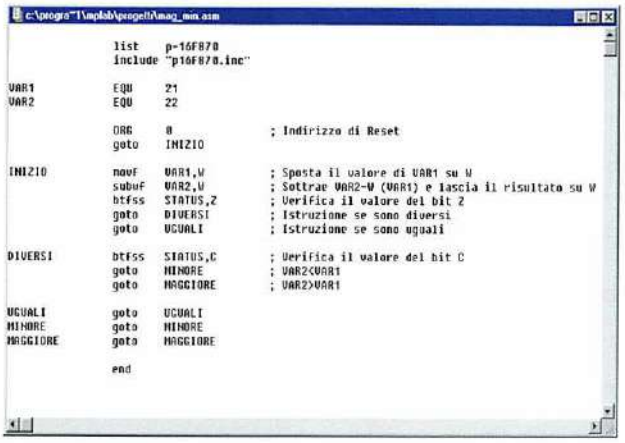
Eseguiremo in questo caso un tipico esempio di apprendimento in qualsiasi linguaggio di programmazione. Si tratta di identificare tra due numeri qual è il maggiore. Provate a trovare voi stessi la soluzione senza ricorrere alle figure. Ricordate che, come prima cosa, dovete fare un organigramma che risponda a come risolvere il problema per poi passare a scrivere il codice. Nella figura in basso a sinistra si può vedere una delle possibili soluzioni dell'enunciato proposto.

```
CLRW
IORLW   10101011b
BTFSS   W,3
CLRW
MOVLW   FFh
```

Codice dell'Esempio 1.

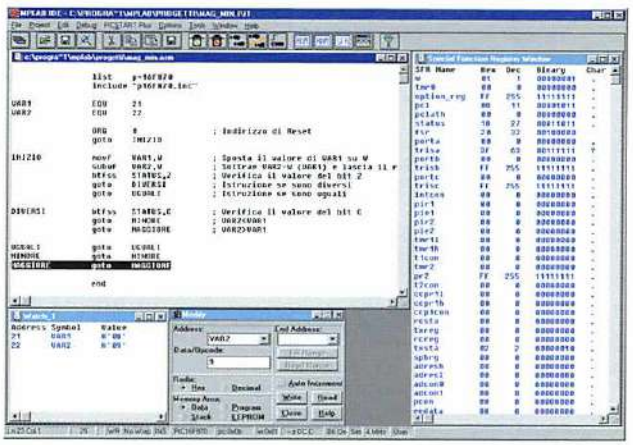
Per simulare il programma dobbiamo ricorrere alla finestra Modify e inserire valori nelle variabili da comparare. Se in VAR1 inseriremo un 9 e in VAR2 inseriremo un 8, eseguendo il programma vedremo che la sua esecuzione si ferma nel ciclo MINORE, dato che VAR2<VAR1. Se cambiamo il valore di VAR2 inserendo una A (10 in decimale), vedremo che il programma si ferma nel ciclo maggiore.

Nella figura della pagina precedente possiamo vedere la videata di simulazione utilizzata in questo programma.



```
c:\progra~1\mplab\progetti\mag_min.asm

            list    p=16F870
            include "p16F870.inc"

VAR1        EQU     21
VAR2        EQU     22

            ORG     0               ; Indirizzo di Reset
            goto    INIZIO

INIZIO      movf    VAR1,W          ; Sposta il valore di VAR1 su W
            subwf   VAR2,W          ; Sottrae VAR2-W (VAR1) e lascia il risultato su W
            btfss   STATUS,Z        ; Verifica il valore del bit 2
            goto    DIVERSI         ; Istruzione se sono diversi
            goto    UGUALI          ; Istruzione se sono uguali

DIVERSI     btfss   STATUS,C        ; Verifica il valore del bit C
            goto    MINORE          ; VAR2<VAR1
            goto    MAGGIORE        ; VAR2>VAR1

UGUALI      goto    UGUALI
MINORE      goto    MINORE
MAGGIORE    goto    MAGGIORE

            end
```

Codice dell'Esempio 2.



Simulazione dell'Esempio 2. Sarà necessario utilizzare la finestra Modify per inserire i valori da comparare.

RA1	RA0	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1

Sequenza di uscita della porta B nell'Esempio 3.

Esempio 3

In quest'ultimo esempio vogliamo imparare a utilizzare le strutture di controllo e iniziare ad avvicinare i nostri programmi ad applicazioni più reali. In questo esercizio di tipo combinazionale si tratta di fornire diverse combinazioni di uscita in funzione dello stato logico dei due segnali di ingresso.

In base allo stato dei segnali collegati ai pin RA0 e RA1, si attivano le uscite collegate alla porta B (RB0:RB7) conforme alla tabella della verità della figura.

Supponiamo di realizzare un semplice controllo di illuminazione mediante due interruttori.

Quando nessun interruttore è premuto tutte le luci rimangono spente, quando si preme uno di essi si accenderanno le zone gestite da questo interruttore, e quando li attiveremo tutti e due insieme, tutte le luci si accenderanno. Questa è una delle possibili applicazioni che si possono fare con questo esercizio.

Quando simuleremo questo esempio potremo verificare solamente una combinazione RA0=RA1=0, in quanto questi pin non potranno essere forzati su valori diversi. Quello che faremo sarà caricare sul microcontroller il programma, dopo aver imparato a programmarlo, e lo verificheremo col suo hardware corrispondente (due interruttori o, in mancanza, due fili, e otto diodi LED con le loro resistenze di limitazione collegate alla porta B).

c:\programmi\mplab\progetti\ed27_3.asm

```
        list    p=16F870        ;Tipo di processore
        include "P16F870.INC"   ;Definizione dei registri interni

        org     0x00            ;Vector di reset
        goto    Inizio

        org     0x05            ;Salva il vector di interrupt

Inizio  clrf    PORTB           ;Cancella il latch di uscita
        bsf     STATUS,RP0      ;Seleziona il banco 1
        clrf    TRISB           ;Porta B configurata come uscita
        movlw   b'00011111'
        movwf   TRISA           ;Porta A configurata come ingresso
        bcf     STATUS,RP0      ;Seleziona il banco 0

Ciclo   clrwdt                  ;Aggiorna il WDT
        btfsc   PORTA,0         ;Verifica lo stato di RA0
        goto    RA0_è_1         ;Se è 1 salta a questa etichetta
        btfsc   PORTA,1         ;RA0=0 e testa lo stato di RA1
        goto    Sono_10         ;Salta a questa etichetta se RA1=1
        clrf    PORTB           ;Sono a 00, uscite tutte a zero
        goto    Ciclo

Sono_10 movlw   b'10101010'
        movwf   PORTB           ;Uscita per la combinazione 10
        goto    Ciclo

RA0_è_1 btfsc   PORTA,1         ;Verifica lo stato di RA1
        goto    Sono_11         ;Salta se RA1=1
        movlw   b'01010101'
        movwf   PORTB           ;Uscita per la combinazione 01
        goto    Ciclo

Sono_11 movlw   b'11111111'
        movwf   PORTB           ;Uscita per la combinazione 11
        goto    Ciclo

        end                     ;Fine del programma sorgente
```

Codice che risolve l'Esempio 3.

Programma per i primi esperimenti con il PIC

Anche se abbiamo già provato il funzionamento del PIC con il programma scritto, la procedura corretta è realizzare il progetto passo a passo, iniziando col definire ciò che vogliamo fare e stabilendo l'hardware e il software necessari, e infine realizzare la verifica pratica.

Primi esperimenti

Di seguito inizieremo un progetto che comprende diversi esperimenti e lo descriveremo in modo molto breve:

– Generazione di un numero casuale tra 0 e 9 con presentazione in binario.

– Generazione di un numero casuale tra 0 e 9 con presentazione su display a 7 segmenti tramite un driver BCD/7 segmenti.

– Generazione di un numero casuale tra 0 e 9 con collegamento diretto al display senza utilizzare il driver precedentemente citato.

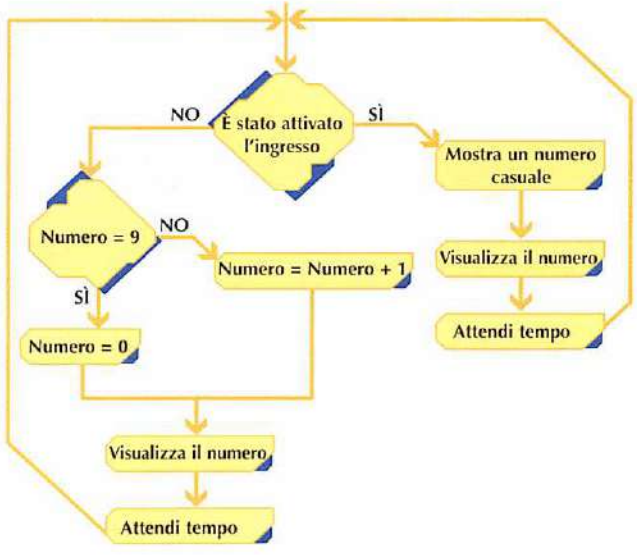
Anche se questo programma è già scritto nel PIC, lo spiegheremo nel dettaglio.

Analisi dell'enunciato

Si vuole generare un numero casuale tra 1 e 9 e visualizzarlo su quattro LED, che rappresentano il codice binario, e su un display a 7 segmenti.

Si dispone di un pulsante per attivare un ingresso del circuito che ferma il conteggio per visualizzare il numero casuale per tre secondi, tornando poi a ripetere, trascorso questo tempo, la visualizzazione sequenziale.

Per prima cosa bisogna capire in modo chiaro l'enunciato e portarlo in un organigramma.

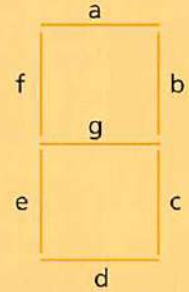


Organigramma dell'applicazione.

L'organigramma non ha lo scopo di entrare nel dettaglio, ma ci aiuterà quando inizieremo a sviluppare il codice.

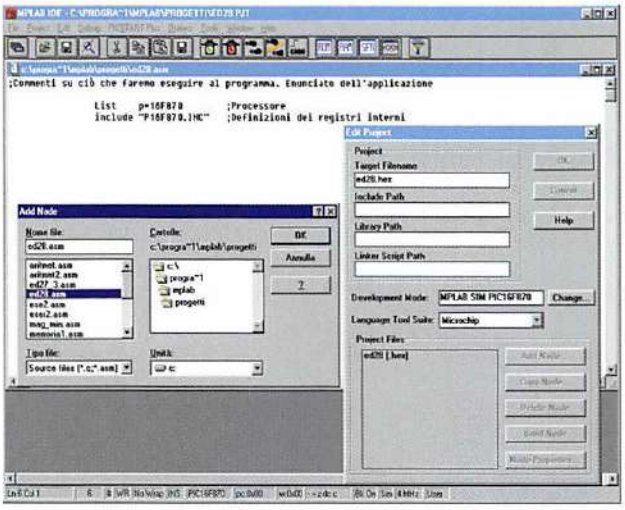
Fatto questo, il passo successivo consiste nell'elencare i dispositivi di cui abbiamo bisogno:

– Uscite: dato che il numero da rappresentare è compreso tra 1 e 9, con 4 bit, 4 uscite saranno sufficienti nel caso di attivare 4 LED o utilizzare un driver per convertire il codice binario a 7 segmenti. Tuttavia, se vogliamo attivare direttamente il display senza utilizzare i driver, oltre alle 4 uscite che vogliamo utilizzare per rappresentare il numero in binario, avremo bisogno di 8 uscite in più per ottenere direttamente dal PIC la conversione a 7 segmenti. Utilizzeremo 4 terminali della porta C (RC3:RC0) per ottenere il numero in binario, e 8 terminali della porta B (RB7:RB0) per ottenere il numero in codice a 7 segmenti.



	DIGIT	BINARIO	7 SEGMENTI b'Ogfedcba'
	0	0000	00111111
	1	0001	00000110
	2	0010	01011011
	3	0011	01001111
	4	0100	01100110
	5	0101	01101101
	6	0110	01111101
	7	0111	00000111
	8	1000	01111111
	9	1001	01100111

Conversione di un digit a codice a 7 segmenti.



Creiamo un progetto, lo editiamo e iniziamo a generare il codice.

Configuriamo i dispositivi del PIC, in questo caso le porte di I/O.

```
Loop        clrwdt                  ;Aggiorna il WDT
            btfss   PORTA,0         ;RA0 è stato attivato?
            goto    Loop            ;Non ancora
            movf    TMR0,W          ;Adesso sì
            movwf   Numero          ;Acquisisce il valore del TMR0 (N° casuale)
```

È stato premuto l'ingresso?

```
Dividi:     movlw   d'9'
            subwf   Numero,W        ;Sottrae 9 al numero casuale
            movwf   Numero          ;Lo memorizza
            sublw   d'8'
            btfss   STATUS,C        ;Controlla se è minore di 8
            goto    Dividi          ;NO
            incf    Numero,F        ;Il numero è compreso fra 1 e 9
```

Adattamento del valore acquisito per ottenere un numero fra 0 e 9.

– Ingressi: abbiamo bisogno solamente di un ingresso digitale, dato che in base a esso presenteremo il numero casuale o i numeri dall'1 al 9 in modo sequenziale. Configureremo RA0 della porta A come ingresso digitale.

– Dispositivi interni: dato che abbiamo due

```
Tabella:    addwf   PCL,F           ;Spostamento sulla tabella
            retlw   b'00111111'     ;Numero 0
            retlw   b'00000110'     ;Numero 1
            retlw   b'01011011'     ;Numero 2
            retlw   b'01001111'     ;Numero 3
            retlw   b'01100110'     ;Numero 4
            retlw   b'01101101'     ;Numero 5
            retlw   b'01111101'     ;Numero 6
            retlw   b'00000111'     ;Numero 7
            retlw   b'01111111'     ;Numero 8
            retlw   b'01100111'     ;Numero 9
```

Routine di conversione in codice a 7 segmenti.

```
Delay_20_ms:    bcf     INTCON,T0IF     ;Azzera il flag di overflow
                movlw   0xb1            ;Complemento hex. di 78
                movwf   TMR0            ;Carico il TMR0
Delay_20_ms_1   clrwdt                  ;Aggiorna il WDT
                btfss   INTCON,T0IF     ;Overflow del TMR0?
                goto    Delay_20_ms_1   ;Non ancora
                return
```

Routine di temporizzazione per eliminare i rimbalzi.

```
Delay_var:      bcf     INTCON,T0IF     ;Azzera il flag di overflow
                movlw   0x3c            ;Complemento hex. di 195
                movwf   TMR0            ;Carico il TMR0
Intervallo      clrwdt                  ;Aggiorna il WDT
                btfss   INTCON,T0IF     ;Overflow del TMR0?
                goto    Intervallo      ;Non ancora
                decfsz  Delay_cont,F    ;Decrementa il contatore di intervalli
                goto    Delay_var       ;Ripete l'intervallo di 50 ms
                return
```

Routine di temporizzazione di un tempo determinato multiplo di 50 ms in funzione di una variabile.

temporizzazioni, una per la rappresentazione del numero casuale e l'altra per visualizzare sequenzialmente i numeri, è necessario utilizzare i temporizzatori del PIC. Sarà sufficiente utilizzare il TMR0, anche se potremmo utilizzare un temporizzatore diverso a ogni temporizzazione.

Codice del programma

Possiamo iniziare a creare il codice che risolverà l'applicazione. A questo scopo apriremo MPLAB e creeremo un progetto, apriremo l'editor, daremo un nome al file in assembler e, mediante Edit Project, assegneremo questo file al progetto.

Di seguito presenteremo il nostro programma, ma voi potrete farne uno diverso che risolva comunque l'intendimento iniziale. Ogni progettista può presentare una soluzione differente, anche se, in linea generale, devono avere le stesse caratteristiche.

Configurazione

La prima cosa da fare quando iniziamo a programmare è configurare il nostro microcontroller, definendo il funzionamento delle porte di ingresso e uscita e dei dispositivi interni,

```
Dato:           movlw   d'9'
                movwf   Temporale       ;Inizializza il contatore del dato
RA0_1           clrwdt                  ;Aggiorna il WDT
                btfss   PORTA,0         ;Controlla se RA0 è a 1
                goto    Uscita          ;No, visualizza il numero casuale
                movf    Temporale,W     ;Numero da visualizzare
                movwf   PORTC           ;Visualizza sui LED
                call    Tabella         ;Conversione in BCD
                movwf   PORTB           ;Visualizza sul display
                movlw   d'1'
                movwf   Delay_Cont      ;Variabile di temporizzazione
                call    Delay_var       ;Temporizza 0,05"
                decfsz  Temporale,F     ;Numero successivo
                goto    RA0_1
                goto    Dato

                call    Delay_20_ms     ;Elimina rimbalzi

Uscita:         movf    Numero,W        ;Acquisisce il numero casuale
                movwf   PORTC           ;Porta il numero casuale in binario sui LED
                call    Tabella         ;Lo converte in 7 segmenti
                movwf   PORTB           ;Uscita sul display
                movlw   d'60'
                movwf   Delay_Cont      ;Inizializza variabile di temporizzazione
                call    Delay_var       ;Temporizza 3"
                clrf    PORTB           ;Scollega l'uscita
                clrf    PORTC           ;Scollega l'uscita sui LED

                goto    Loop

                end                     ;Fine del programma sorgente
```

Codice che risolve la visualizzazione.

se si lavora con essi. Il codice, quindi, dovrebbe avere la forma della figura.

Lettura dell'ingresso

Configurato il micro potremo già entrare nel programma in sé. Seguendo l'organigramma possiamo vedere che la prima cosa da fare è verificare la condizione dell'ingresso, per sapere se il pulsante è stato premuto oppure no. Se l'ingresso è attivo, RA0=1, inizieremo il programma, acquisendo il valore casuale che si otterrà dal WDT e scrivendo questo valore sulla variabile "Numero". Questa variabile dovrà essere definita all'inizio del programma. Nelle linee di codice dell'immagine si può vedere quanto spiegato in precedenza.

Adattamento del numero casuale

Il numero casuale è, mediante sottrazioni consecutive, diviso per 9. In questo modo l'ultimo resto sarà compreso tra 0 e 8, che verrà incrementato di una unità per ottenere alla fine un numero tra 1 e 9.

Subroutine

– Conversione. Per poter presentare i numeri direttamente sul display a 7 segmenti li dobbiamo convertire in questo codice (se si utilizza il driver per il display sarà sufficiente mandare il codice binario). A questo scopo creiamo una subroutine che verrà richiamata quando avremo bisogno di questo valore. Dato che questa subroutine contiene solamente una tabella di conversione, la chiameremo "Tabella".

```
                List    p=16F870        ;Processore
                include "P16F870.INC"   ;Definizioni dei registri interni

Numero          equ     0x20            ;Numero casuale
Delay_Cont      equ     0x21            ;Contatore di intervalli
Temporale       equ     0x22            ;Variabile temporale

                org     0x00            ;Vector di Reset
                goto    Inizio
                org     0x05            ;Salva il vector di interrupt
;-----------------------------------------------------------------------
Tabella:        addwf   PCL,F           ;Spostamento sulla tabella
                retlw   b'00111111'     ;Numero 0
                retlw   b'00000110'     ;Numero 1
                retlw   b'01011011'     ;Numero 2
                retlw   b'01001111'     ;Numero 3
                retlw   b'01100110'     ;Numero 4
                retlw   b'01101101'     ;Numero 5
                retlw   b'01111101'     ;Numero 6
                retlw   b'00000111'     ;Numero 7
                retlw   b'01111111'     ;Numero 8
                retlw   b'01100111'     ;Numero 9
;-----------------------------------------------------------------------
Delay_20_ms:    bcf     INTCON,T0IF     ;Azzera il flag di overflow
                movlw   0xb1            ;Complemento hex. di 78
                movwf   TMR0            ;Carico il TMR0
Delay_20_ms_1   clrwdt                  ;Aggiorna il WDT
                btfss   INTCON,T0IF     ;Overflow del TMR0?
                goto    Delay_20_ms_1   ;Non ancora
                return
;-----------------------------------------------------------------------
Delay_var:      bcf     INTCON,T0IF     ;Azzera il flag di overflow
                movlw   0x3c            ;Complemento hex. di 195
                movwf   TMR0            ;Carico il TMR0
Intervallo      clrwdt                  ;Aggiorna il WDT
                btfss   INTCON,T0IF     ;Overflow del TMR0?
                goto    Intervallo      ;Non ancora
                decfsz  Delay_Cont,F    ;Decrementa il contatore di intervalli
                goto    Delay_var       ;Ripete l'intervallo di 50 ms
                return
;-----------------------------------------------------------------------
Inizio          clrf    PORTB           ;Cancella i latch di uscita
                clrf    PORTC           ;Cancella i latch di uscita
                bsf     STATUS,RP0      ;Seleziona banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                clrf    TRISC           ;Porta C si configura come uscita
                movlw   0x07
                movwf   ADCON1          ;Si configura la Porta A come I/O digitali
                movlw   0xFF
                movwf   TRISA           ;Porta A si configura cone ingresso
```

Programma completo che risolve l'enunciato 1.

```
                movlw   b'00000111'
                movwf   OPTION_REG      ;Prescaler da 256 per il TMR0
                bcf     STATUS,RP0      ;Seleziona il banco 0

Loop            clrwdt                  ;Aggiorna il WDT
                btfss   PORTA,0         ;RA0 è stato attivato?
                goto    Loop            ;Non ancora
                movf    TMR0,W          ;Adesso sì
                movwf   Numero          ;Acquisisce il valore del TMR0 (N° casuale)

                call    Delay_20_ms     ;Elimina rimbalzi
        ;-----------------------------------------------------------------------
Dividi:         movlw   d'9'
                subwf   Numero,W        ;Sottrae 9 al numero casuale
                movwf   Numero          ;Lo memorizza
                sublw   d'8'
                btfss   STATUS,C        ;Controlla se è minore di 8
                goto    Dividi          ;NO
                incf    Numero,F        ;Il numero è compreso fra 1 e 9
        ;-----------------------------------------------------------------------
Dato:           movlw   d'9'
                movwf   Temporale       ;Inizializza il contatore del dato
RA0_1           clrwdt                  ;Aggiorna il WDT
                btfss   PORTA,0         ;Controlla se RA0 è a 1
                goto    Uscita          ;No, visualizza il numero casuale
                movf    Temporale,W     ;Numero da visualizzare
                movwf   PORTC           ;Visualizza sui LED
                call    Tabella         ;Conversione in BCD
                movwf   PORTB           ;Visualizza sul display
                movlw   d'1'
                movwf   Delay_Cont      ;Variabile di temporizzazione
                call    Delay_var       ;Temporizza 0,05"
                decfsz  Temporale,F     ;Numero successivo
                goto    RA0_1
                goto    Dato

                call    Delay_20_ms     ;Elimina rimbalzi

Uscita:         movf    Numero,W        ;Acquisisce il numero casuale
                movwf   PORTC           ;Porta il numero casuale in binario sui LED
                call    Tabella         ;Lo converte in 7 segmenti
                movwf   PORTB           ;Uscita sul display
                movlw   d'60'
                movwf   Delay_Cont      ;Inizializza variabile di temporizzazione
                call    Delay_var       ;Temporizza 3"
                clrf    PORTB           ;Scollega l'uscita
                clrf    PORTC           ;Scollega l'uscita sui LED
                goto    Loop

                end                     ;Fine del programma sorgente
```

Programma completo che risolve l'enunciato 2.

Il funzionamento di questa tabella è molto semplice, dato che si tratta di sommare al contatore di programma il valore che vogliamo tradurre, in questo modo il contatore di programma va sulla posizione dove si trova il valore tradotto e ritorna dalla subroutine.

– Temporizzazioni. Nel nostro esempio abbiamo creato due temporizzazioni. Una ha lo scopo di eliminare i rimbalzi o i valori errati che possono generare gli elementi elettromeccanici. Ha una durata fissa di 20 ms e questo valore deriva dal fatto che il PIC lavora a una frequenza di 4 MHz e il TMR0 evolve ogni microsecondo, quindi se vogliamo temporizzare 20.000_s (20 ms) con un predivisore di 256, il TMR0 dovrà contare 78 eventi (78x256). Il valore 78 equivale a 0x4e h e dato che il TMR0 è ascendente, dovremo caricare il suo complemento a 1 (0xb1 hex.).

Mediante una nuova routine di temporizzazione programmeremo il tempo da attendere nella rappresentazione sequenziale o nella visualizzazione del numero casuale. Il ragionamento per la nuova temporizzazione è simile al precedente: la velocità di lavoro è di 4 MHz , quindi il TMR0 si incrementa ogni microsecondo. In questo modo il TMR0 deve contare 195 eventi che, con un predivisore di 128, producono un intervallo totale di 50.000 µs - 50 ms (195x256). Il valore 195 deve essere espresso in esadecimale (c3) e, dato che TMR0 è ascendente, bisognerà caricare il suo complemento (3c h). Questo intervallo di 50 ms si ripete tante volte quante indicate dalla variabile "Delay_cont"; per questa ragione il ritardo (in inglese delay) minimo è di 50 ms ("Delay_cont=1) e quello massimo di 12,8" (Delay_cont=255). Nel nostro caso la temporizzazione di 3 secondi nella visualizzazione del numero casuale la realizzeremo con Delay_cont=60.

Abbiamo quindi una nuova variabile che deve essere dichiarata all'inizio del programma.

Con questi esempi si può verificare come sia semplice fare una temporizzazione.

Presentazione dei numeri

Manca poco per avere il nostro primo grande programma risolto, ma manca ancora da fare la parte che può essere considerata la più difficile, risolvere l'enunciato scrivendo il codice in assembler. Abbiamo fatto la configurazione dei dispositivi, l'acquisizione del numero casuale, creato le temporizzazioni, la conversione a 7 segmenti, ma ora dobbiamo assemblarlo in modo più pratico per risolvere l'enunciato iniziale.

Nelle illustrazioni si può vedere come è stato strutturato il programma in modo da risolvere in modo semplice il progetto. Il programma è debitamente commentato per far risultare più semplice la sua comprensione.

Potete verificare come sia stata creata una nuova variabile che servirà per contenere i valori in modo temporale e come vengono chiamate le subroutines o vengono eseguiti i salti alle altre parti del programma commentate in precedenza.

Programma per i primi esperimenti con il PIC (II)

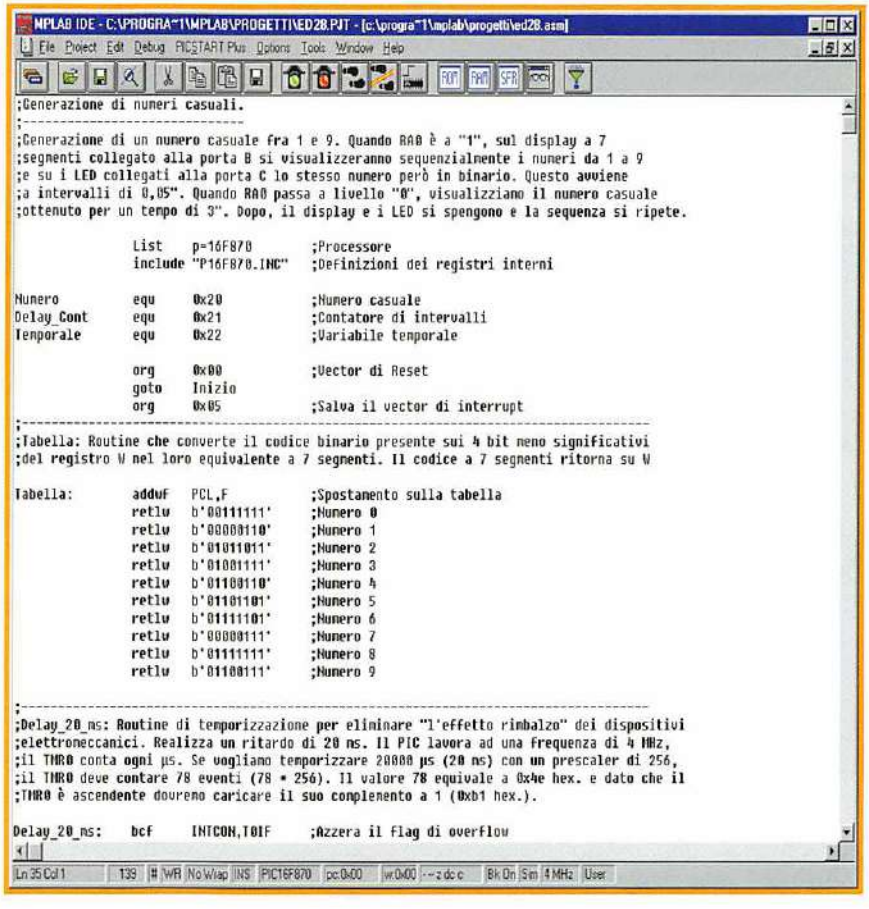
Continuiamo con il programma per il nostro progetto, seguendo però la logica sequenza di sviluppo di un progetto.

Dopo aver realizzato il codice, dobbiamo verificare che questo venga compilato senza errori e che risponda all'enunciato mediante la simulazione.

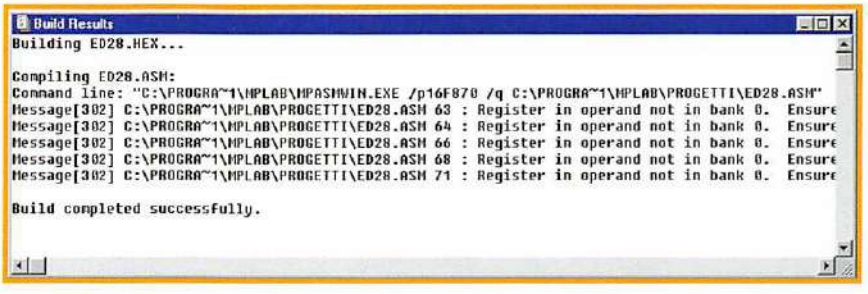
Preparazione del codice

Lo strumento da utilizzare per assemblare e compilare il nostro codice è MPLAB, però prima di compilare dobbiamo "preparare" il nostro programma. Con il termine preparare intendiamo fare riferimento al lavoro di messa a punto e depurazione per rendere il nostro programma veramente pronto. A questo scopo dobbiamo ordinare il codice, rivederlo e aggiungere i commenti per rendere più semplice la sua comprensione. È bene quindi abituarci a inserire delle intestazioni in cui vengano spiegate le funzioni realizzate da ogni blocco differenziato del programma. Possiamo intestare il programma con l'enunciato iniziale dell'applicazione, ogni subroutine con la funzione che realizza, e perché no, il modo in cui è stata risolta. Possiamo inoltre separare – come abbiamo fatto quando abbiamo spiegato il codice – le parti di programma più caratteristiche o che realizzano funzioni specifiche. Nelle illustrazioni viene evidenziato come un progettista lascia il suo codice totalmente "sistemato".

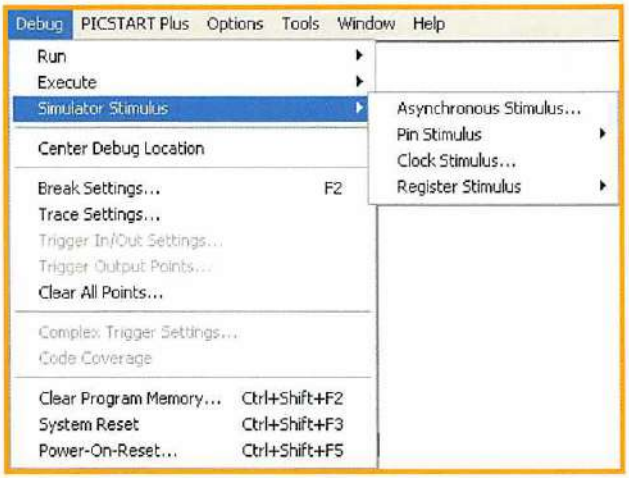
Fatto questo, ordinato e rivisto tutto il codice, possiamo procedere alla sua compilazione.



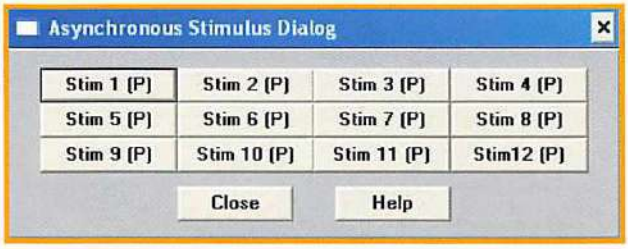
Il codice sta assumendo la sua forma man mano che si avanza nella preparazione.



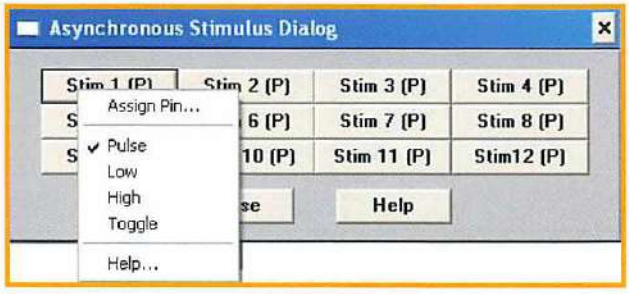
Risultato della compilazione e assemblaggio.



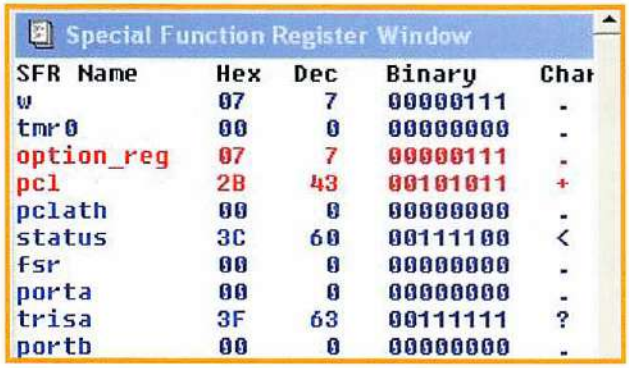
Menù per accedere alla simulazione degli ingressi.



Finestra per stimolare stimoli asincroni.



Per assegnare un ingresso clicchiamo con il pulsante destro del mouse sul comando.



I registri che si modificano nell'esecuzione vengono evidenziati con il colore rosso.

Compilazione

Se selezioniamo l'opzione di compilazione, Project → Build All, o clicchiamo il pulsante della barra degli strumenti, vedremo che il programma che abbiamo presentato come soluzione, si assembla senza errore. Appariranno cinque messaggi per avvertirci di verificare che i registri indicati siano nei banchi di memoria definiti nel programma. Questi messaggi non indicano degli errori, semplicemente avvisano di una situazione che al compilatore risulta strana.

Nel caso abbiate realizzato un programma diverso da quello presentato e il processo di compilazione produca degli errori, sarà necessario correggerli linea per linea, fino ad ottenere un programma libero da errori.

Simulazione

Il fatto che un programma non dia errori di compilazione è un passo importante verso la soluzione finale, però non è quello definitivo, infatti ci potrebbero essere degli errori per quanto riguarda il funzionamento che ci aspettiamo. Per evitare questo MPLAB ci offre la possibilità di simulare il programma ed è esattamente quello che faremo.

Simulazione di un ingresso

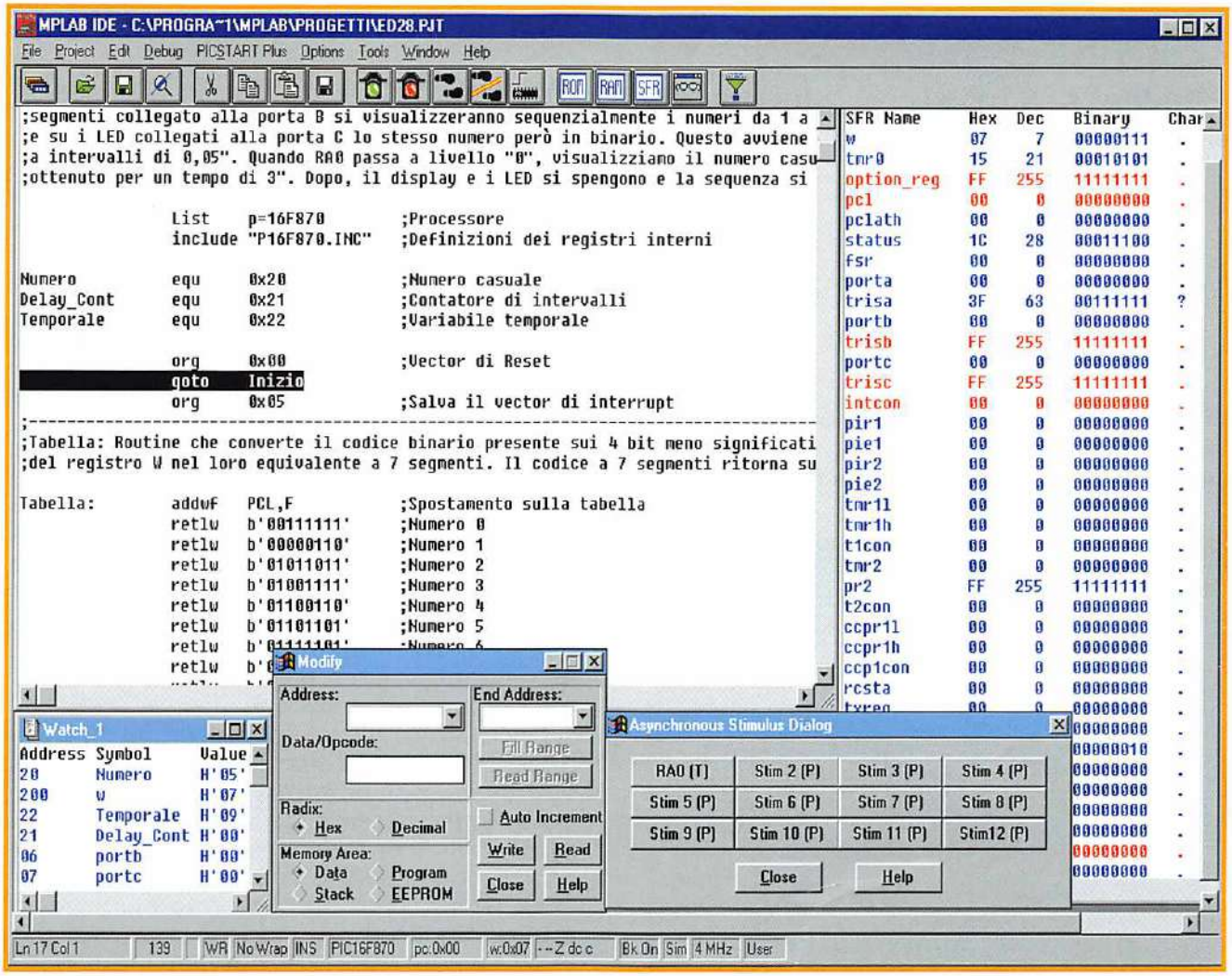
Quando abbiamo studiato MPLAB abbiamo visto tutte le possibilità della simulazione, eccetto quella di simulare uno o più ingressi. In questo programma dipendiamo dal valore di un ingresso (RA0) per poter verificare la risposta del sistema, però questo ingresso si può simulare ed è ciò che impareremo a fare. Selezionando Debug → Simulator Stimulus si apre un menù come quello della figura dove sono presentate quattro opzioni:

– Asynchronous Stimulus (Stimolo asincrono): apre un quadro di dialogo interattivo per controllare i segnali di ingresso sui pin di ingresso del microcontroller.

– Stimulus Pin File (File di stimoli sui terminali): mediante un file di testo si descrivono i segnali di ingresso sui pin.

– Stimulus Register File (File di registro degli stimoli): si utilizza il contenuto di un file di testo per porre direttamente a 1 oppure a 0 ognuno degli 8 bit di un registro.

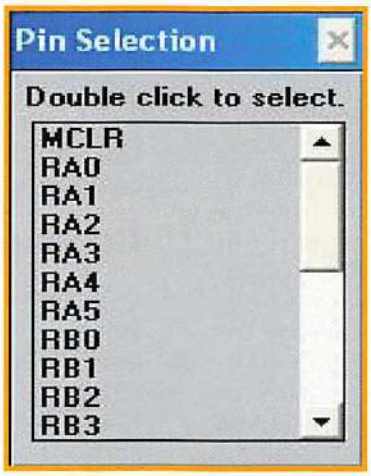
– Clock Stimulus (Segnale di clock): genera un'onda quadra programmabile.



Ecco come appare la videata dopo aver creato le finestre di simulazione.

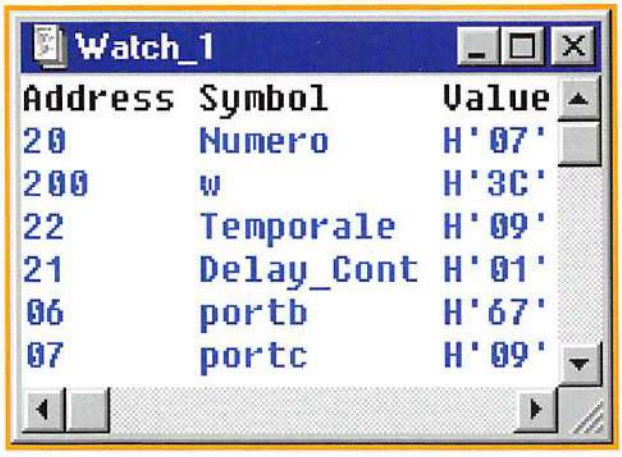
Quando selezioniamo la prima opzione, stimolo asincrono, appare la finestra della figura. La finestra ha dodici pulsanti e a ognuno di essi possiamo assegnare una linea di ingresso, quindi potremo simulare dodici ingressi. Per assegnare un ingresso a un pulsante dobbiamo cliccare il pulsante destro del mouse sul pulsante desiderato.

Così facendo si apre un menù come quello della figura della pagina precedente. Mediante questo tipo di menù possiamo scegliere il tipo di stimolo, ovvero la forma che avrà il segnale di ingresso ogni volta che attiveremo il pulsante: un solo impulso (Pulse), livello basso o 0 (Low), livello alto o 1 (High), o scambiare fra di loro i livelli quando si clicca il pulsante (Toggle). Per assegnare l'ingresso al pulsante selezioneremo "Assign Pin" aprendo così un nuovo menù in cui troviamo la lista di tutti i pin di ingresso del microcontroller.

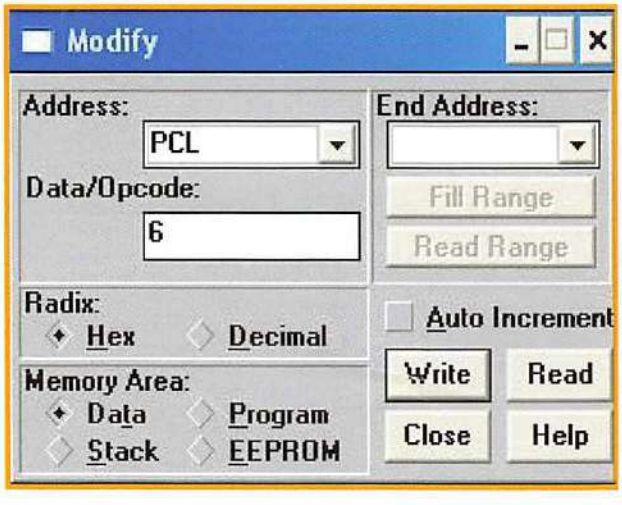


Lista dei pin a cui è possibile assegnare il pulsante.

Selezioneremo RA0 e il modo di stimolo come alternato o Toggle, perciò quando cliccheremo il pulsante per la prima volta l'ingresso passerà a livello alto sino a quando non attiveremo di nuovo il pulsante.



Risultato della prima sequenza con i valori sulle porte di uscita.



Forziamo un valore sul registro contatore di programma.

Finestre di simulazione

Per eseguire la simulazione del programma conviene aprire la finestra dei Registri delle Funzioni Speciali (Special Function Registers). Vi consigliamo inoltre di creare una finestra per poter visualizzare lo stato dei registri più importanti: variabili, registro di lavoro e porte di uscita. Nello stesso tempo avremo una finestra di creazione di stimoli per forzare gli ingressi, e infine dovremo disporre della finestra Modify per poter forzare il valore di un registro ed evitare che il programma rimanga bloccato in cicli infiniti; come capiterebbe ad esempio nella routine di temporizzazione. Nella figura della pagina precedente possiamo vedere che aspetto avrà il video.

Simulazione del programma

Per iniziare la simulazione passo a passo dovremo cliccare l'icona o premere il tasto F7. In questo modo vedremo come il programma inizia la sua esecuzione configurando come prima cosa i dispositivi del PIC. Potremo vedere nella finestra dei registri delle funzioni speciali, come si modificano i valori dei registri lungo il corso dell'esecuzione. Quando si modifica un registro, questo verrà evidenziato in rosso.

Il programma si fermerà in un ciclo (Loop) aspettando di ricevere l'ordine di inizio, cioè che si attivi l'ingresso, quindi per simulare questo dovremo ricorrere al nostro generatore di stimoli e cliccare il pulsante che abbiamo assegnato a RA0. Così facendo, sarà come dire al simulatore che è arrivato un valore 1 (questo valore non cambierà fino a quando non cliccheremo nuovamente il pulsante di stimolo) salteremo l'istruzione "goto Loop" continuando con il programma. Abbiamo acquisito un valore casuale e passiamo alla routine di temporizzazione.

Qui capiterà la stessa cosa di prima, rimarremo in attesa fino a quando il flag di overflow del Timer 0 non passerà a 1. Per evitare un ciclo infinito useremo la finestra Modify selezionando il flag T0IF e scrivendo come valore un 1. Facendo questo salteremo alla tabella di conversione, però non potremo muoverci su questa tabella se non avremo un valore corretto sul contatore di programma, quindi gli assegniamo un valore, ad esempio 6, come mostrato nell'immagine.

Il programma esce dalla tabella e adatta il numero acquisito, il quale, mediante sottrazioni consecutive avrà un valore fra 1 e 9. Ripeteremo il ciclo fino a ottenere tale risultato e a partire da questo punto verranno presentati i numeri in modo sequenziale o il numero casuale in funzione dello stato di RA0.

Conclusioni

Questo esercizio completo non ha nessuna funzionalità se non si impara a giocare con esso. Modificando il programma, provate tutto ciò che è possibile, verificate i vostri dubbi e imparate a gestire il simulatore; riassumendo, è molto importante non avere alcun timore e dedicare il tempo necessario a imparare e prendere confidenza con quanto appreso.

Software di scrittura IC-PROG

Abbiamo già completato la scheda di scrittura del PIC, quindi ora dobbiamo solamente sapere come scriverlo, cioè come passare un programma al microcontroller. Esistono diversi programmi il cui scopo è scrivere il microcontroller, però il più semplice e utilizzato è IC-Prog. Questo è un programma di libera distribuzione che oltre al PIC ci permetterà di scrivere le SmartCard.

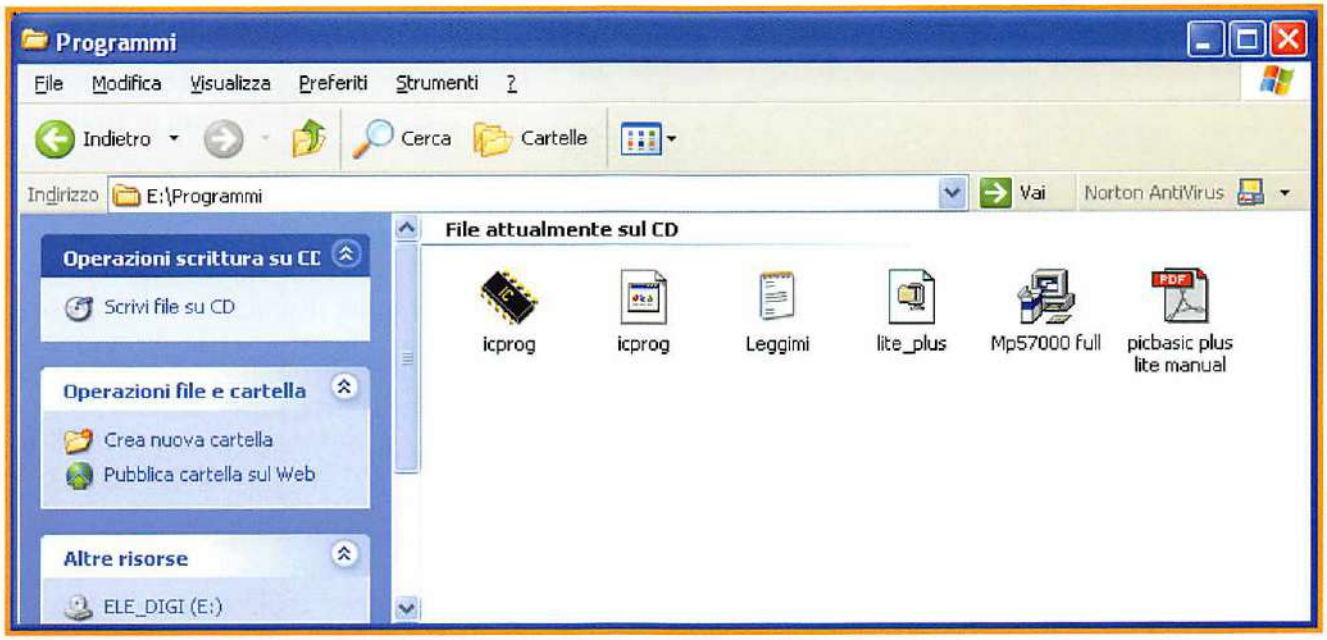
Installazione

Il programma si trova sul CD-ROM dell'opera, nella cartella "Programmi". All'interno di questa cartella i file che ci interessano sono: "icprog.exe" e "icprog.sys".

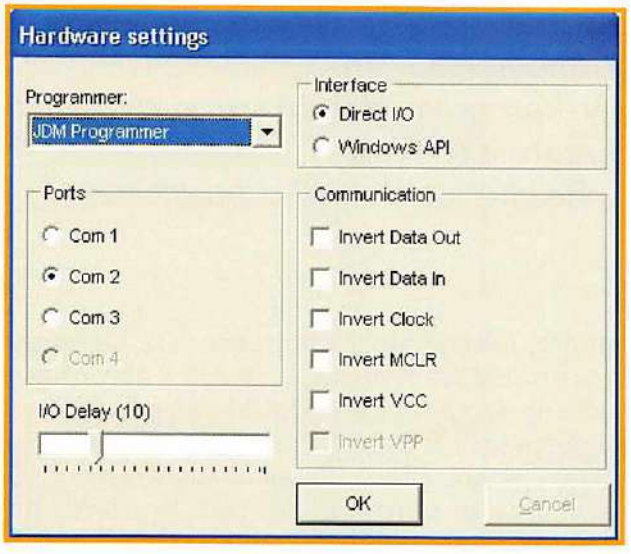
Dobbiamo creare una directory sul nostro PC e copiare in essa i due file, per eseguire in seguito il programma da quel punto.

Copiati i file eseguiremo il programma che ha una forma di microcontroller, a questo punto apparirà una finestra come quella mostrata nella figura. In questa finestra ci viene comunicato che è la prima volta che eseguiamo il programma e che lo dobbiamo configurare per farlo funzionare correttamente. Cliccheremo il pulsante "OK" e apparirà una nuova finestra, la finestra di configurazione dell'hardware. Dobbiamo selezionare la porta seriale del PC che vogliamo utilizzare per collegare il cavo che all'altro estremo avrà la scheda di scrittura. Nel caso avessimo un mouse collegato sulla COM1 selezioneremo la COM2, se fosse libera. Solitamente si dispone di un mouse PS/2, quindi dovremo avere le

Finestra che appare la prima volta che si esegue il file icprog.exe.



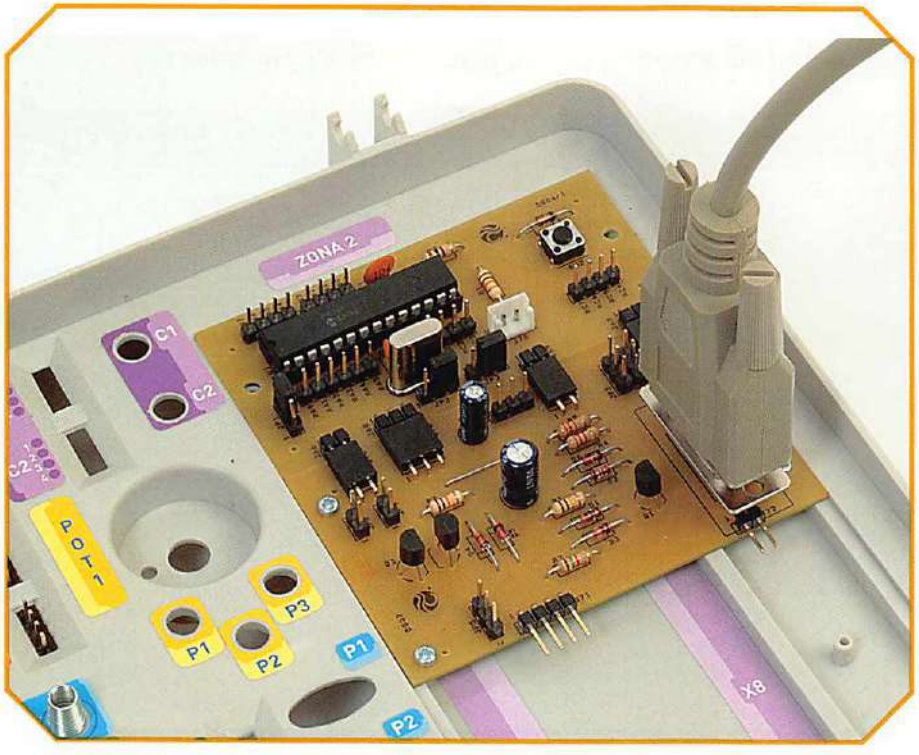
Nel CD-ROM che vi è stato fornito potete trovare il programma IC-Prog.



Configurazione dell'hardware.

Cambiamo la lingua del programma.

porte seriali libere e potremo scegliere la COM1. Se il sistema operativo è Windows 95, 98 o ME selezioneremo Interface Direct I/O. Se abbiamo Windows NT, 2000 o XP sceglieremo Interface Windows API. Il resto dei parametri di configurazione deve essere quello mostrato nella figura.



Collegamento del programmatore alla porta seriale del PC.

Le porte seriali di un PC hanno un connettore DB9 (nove pin) maschio, a questo connettore verrà collegato un cavo che a uno degli estremi avrà un DB9 femmina per collegarsi al PC e all'altro capo un DB9 maschio per collegarsi alla scheda di scrittura del Laboratorio.

Questo cavo sarà del tipo "pin-to-pin", cioè il pin 1 di un estremo sarà collegato al pin 1 dell'altro estremo e così via con tutti gli altri pin. Il cavo potete costruirlo voi stessi o comprarlo in qualche negozio di informatica o elettronica o ai grandi magazzini.

Configurazione

Se nella finestra precedente premiamo OK, si apre direttamente la videata principale di IC-Prog. Nella parte superiore destra troviamo un menù a tendina in cui potremo scegliere il dispositivo con cui lavorare. Il programma dispone di un menù di controllo per realizzare il lavoro di scrittura, di configurazione e anche di un insieme di

All'inizio chiediamo conferma prima di eseguire qualsiasi processo.

Sceglieremo "Verifica durante la programmazione".

Campi misti che devono essere modificati.

Menù di "settaggi".

pulsanti per l'accesso rapido alle funzioni più comuni.

La lingua di default è l'inglese, però la potremo cambiare in italiano facendo accesso a Setting → Options e si aprirà una nuova finestra in cui sceglieremo la scheda Lingua, selezionando quindi la lingua desiderata. A partire da questo momento, quando apriremo IC-Prog, ci verranno presentate tutte le sue opzioni nella lingua selezionata.

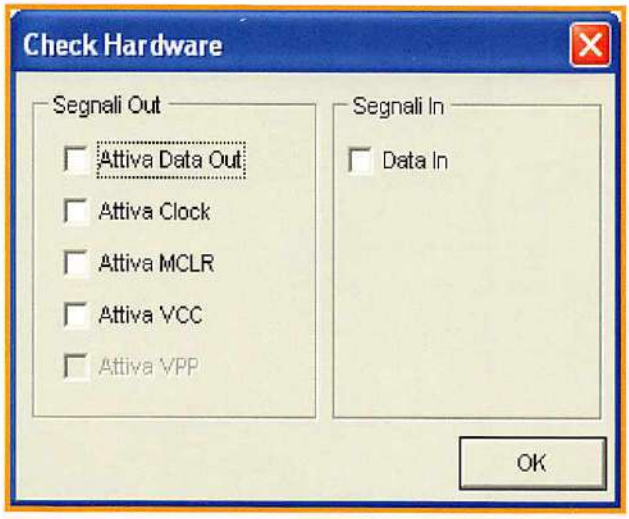
Approfittando dell'apertura di questa finestra, configuriamo il resto delle opzioni, per adeguare il software all'applicazione a cui lo vogliamo dedicare. Vedremo ogni scheda configurando campo per campo:

– Conferme: in questa scheda configureremo quando il software dovrà richiedere una conferma prima di eseguire un processo. Anche se non è necessario possiamo scegliere delle opzioni, almeno per ora, visto che stiamo iniziando a usare il programma, come misura precauzionale.

– Notifiche: sia il programma che il software ci avvisano nel caso in cui si verifichino le condizioni presentate. Anche in questa scheda non è indispensabile scegliere qualcosa.

– I²C: per il momento non toccheremo niente in questa scheda, lasciando i valori di default.

– Programmazione: in questa scheda sceglieremo l'opzione "Verifica durante la pro-



Opzioni per verificare che il dispositivo selezionato comunichi con il software.

Conferma per cancellare tutti i settaggi eseguiti.

grammazione", questo è un modo per verificare che il processo di scrittura sia stato realizzato correttamente.

– Shortcuts: si configurano combinazioni di tasti per l'accesso rapido, in modo che premendo i tasti indicati potremo cambiare automaticamente dispositivo. Noi non abbiamo bisogno di questa caratteristica, quindi lasceremo questa scheda così com'è di default.

– Drag & Drop: configurando questa possibilità ne potremo beneficiare utilizzando il mouse. Non abiliteremo l'opzione.

– SmartCard - Shell: opzioni di configurazioni che momentaneamente non cambieremo, lasciando quindi i valori di default.

– Lingua: opzione commentata in precedenza per cambiare la lingua del software.

– Misc: si utilizza la velocità di processo in "realtime", perché è quella che funziona nella maniera ottimale per Windows 2000 e XP, per gli altri sistemi come W98 si può utilizzare la velocità di elaborazione "realtime" o "normale", in quest'ultimo caso il driver per 2000/XP non si attiverà.

Nell'immagine possiamo vedere come rimane configurata questa opzione.

Il menù di "Settaggi"

Configurate tutte le opzioni continuiamo a vedere gli altri menù, nello specifico il menù dei settaggi. Se apriamo il menù a tendina vediamo le opzioni mostrate nell'immagine della pagina precedente in basso.

Il primo campo del menù si chiama "Chip". Aprendo questo menù potremo selezionare il dispositivo con cui vogliamo lavorare. Ricordate che nella parte superiore destra della barra degli strumenti, possiamo selezionare il dispositivo con cui lavorare. Per il momento non selezioneremo alcun dispositivo

Il campo successivo "Chip Recenti" memorizza gli ultimi dispositivi con cui abbiamo lavorato, avendo così a disposizione un accesso rapido nel caso volessimo cambiare dispositivo.

Mediante "Hardware" torneremo alla finestra di configurazione iniziale in cui selezioniamo il programmatore, la porta, l'interfaccia e la comunicazione. L'opzione successiva "Controllo Hardware", ci permetterà di provare che la comunicazione precedente è corretta e se comunica con il dispositivo selezionato. Questa prova la utilizzeremo raramente, dato che lavoreremo unicamente con due dispositivi, sapendo già se la configurazione e la comunicazione sono state fatte correttamente.

Il campo "Opzioni" è stato commentato in precedenza e il campo successivo "SmartCard (Phoenix)" lo selezioneremo nel caso disponessimo di questo dispositivo.

Infine, mediante "Ripristina Settaggi", potremo ristabilire i valori di default e cancellare tutte le variazioni eseguite in questa sezione del menù. Dato che risulta un'operazione critica ci verrà chiesta conferma prima di eseguire questa opzione, come possiamo vedere nella figura.

Software di scrittura: IC-PROG (II)

Configurato il software dobbiamo spiegare i passaggi necessari per procedere alla scrittura del dispositivo. Vedrete quanto risulta semplice gestire il programma e capirete perché è uno dei più utilizzati nel mondo dei microcontroller.

Selezione del dispositivo

Tutto il processo di scrittura inizia dal selezionare il dispositivo che si vuole programmare. Abbiamo visto che esistono due modi di selezionare quest'ultimo, uno nel menù "Settaggi", e l'altro di accesso rapido che sarà quello che utilizzeremo. Se apriamo la finestra che si trova nella parte superiore destra del programma, vedremo tutti i dispositivi con cui possiamo lavorare.

Selezioniamo il PIC 16F870, dato che al momento (lavoreremo con SmartCard) scriveremo solo il microcontroller. L'aspetto della videata cambia selezionando uno o l'altro dispositivo. Quindi, con il micro selezionato apparirà la finestra di configurazione del chip e nella parte sinistra la sua memoria di programma e dei dati EEPROM.

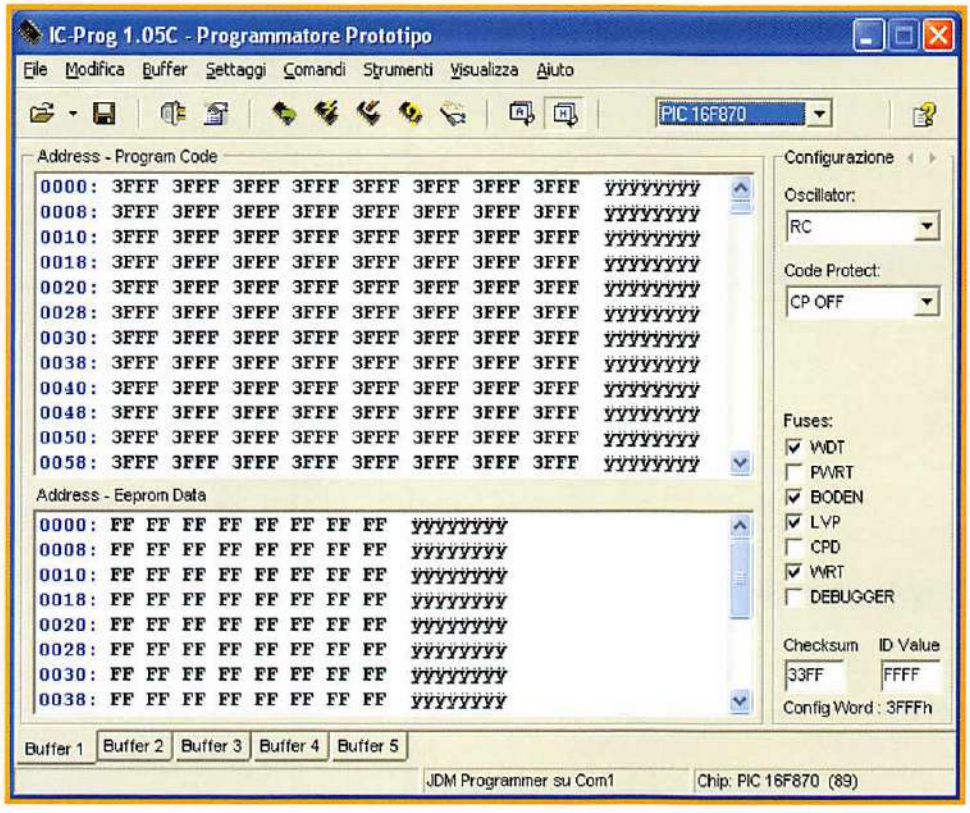
Quando desideriamo scrivere programmi o dati sulla SmartCard, la prima cosa che dobbiamo fare è selezionare il tipo di memoria di cui dispone la scheda, nel nostro caso la 24C16. Selezionando la memoria la videata assumerà l'aspetto mostrato in figura nella pagina successiva.

Selezione del file

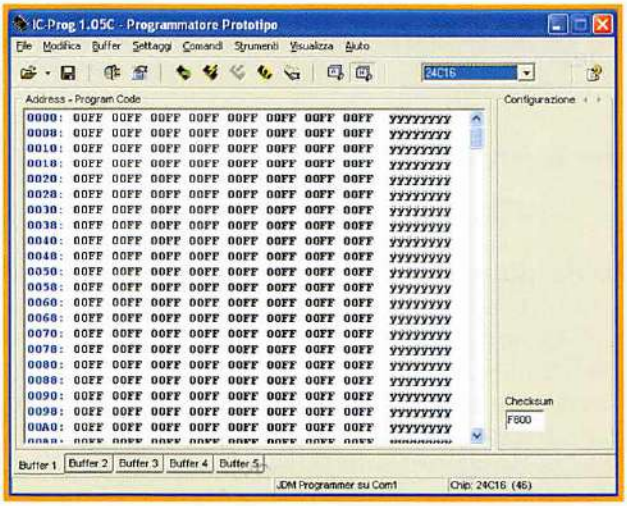
Dopo aver scelto il dispositivo, il passo successivo consiste nel selezionare il file che vogliamo scrivere sul dispositivo. A questo scopo apriremo il menù "file" e all'interno di esso potremo vedere le seguenti opzioni:

– Apri: mediante questa opzione possiamo caricare sul software il file che vogliamo scrivere sul dispositivo. Scriveremo sempre file del tipo "IHX8 file" che avranno estensione ".hex", sia per il PIC che per la SmartCard. Nel campo "Tipo file" potremo selezionare di vedere unicamente questo tipo di file, dato che per default appariranno tutti i file con qualsiasi estensione. Continueremo seguendo il file compilato con l'estensione specificata in precedenza che vogliamo scrivere sul dispositivo e cliccheremo sul pulsante "Apri".

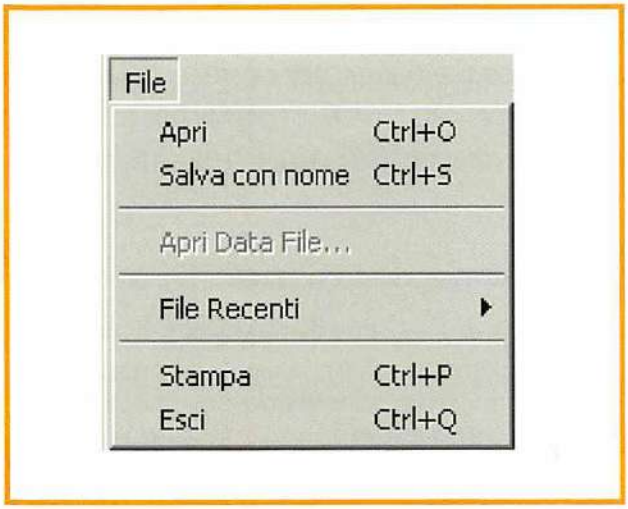
Ricordate che ogni volta che si assembla un file con MPLAB, au-



Presentazione della videata con il PIC 16F870 selezionato.



Videata per lavorare con le SmartCard, le quali non sono altro che una memoria di tipo 24C16.



Menù "File".

tomaticamente si crea un file con estensione ".hex", quindi se puntiamo alla directory dove abbiamo lavorato con MPLAB, potremo trovare diversi di questi file.

– Salva con nome: mediante questa opzione possiamo salvare i cambiamenti eseguiti in un progetto aperto in precedenza.

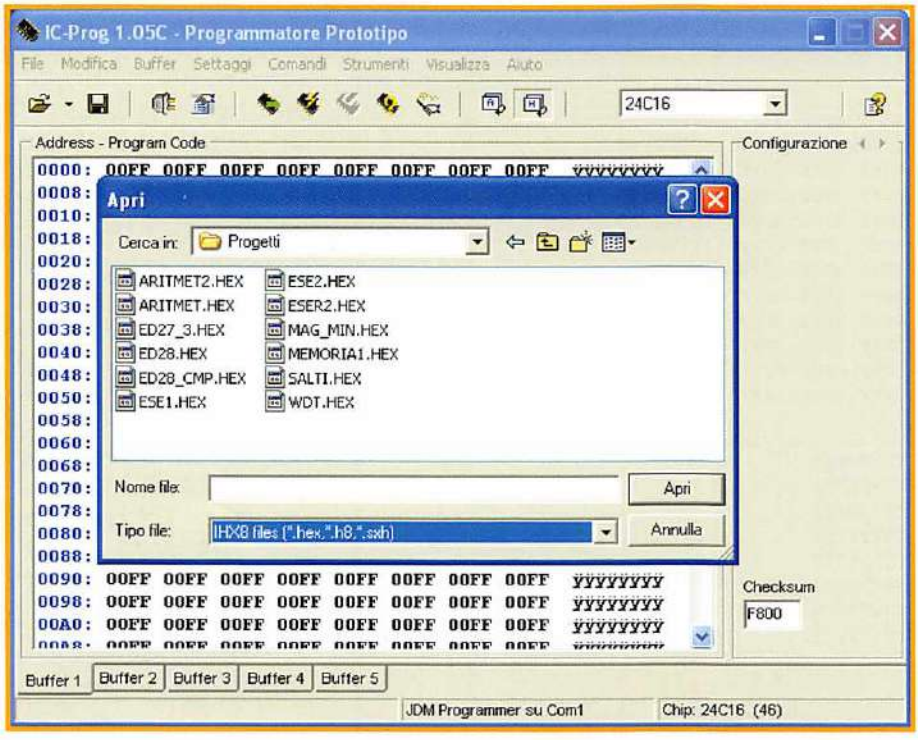
Non consideriamo necessario analizzare il resto delle opzioni di questo menù, data la loro semplicità e il loro utilizzo poco frequente.

Presentazione e informazione menù "Visualizza"

Quando carichiamo un file qualsiasi la presentazione che ne risulta non è pratica, perché è fatta in esadecimale. In questo modo non possiamo analizzare il programma, dato che si trova tradotto in questo formato e non lo sappiamo decodificare.

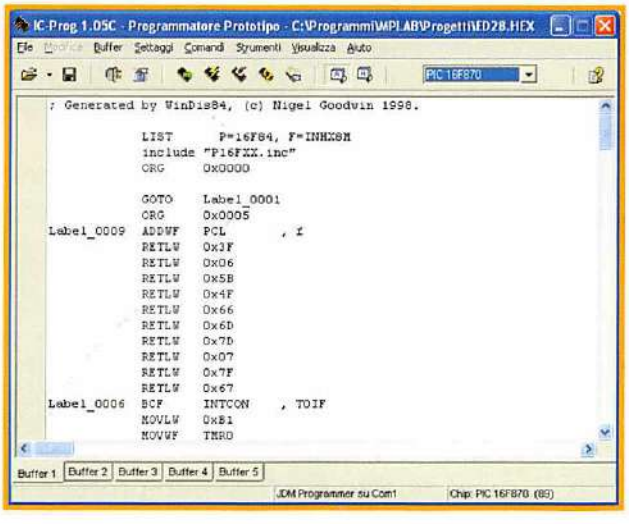
Nel menù "Visualizza" sono presentate due opzioni di visualizzazione: una è l'opzione di default che in "Hex" – esadecimale – e l'altra è l'opzione "Assembler".

Mediante quest'ultima opzione possiamo vedere il file che desideriamo scrivere o leggere dal dispositivo in formato assembler, formato che abbiamo imparato e sappiamo interpretare. In questo modo possiamo eseguire cambiamenti nel file senza la necessità di ricorrere a un software dif-



Selezioniamo "Apri file" per caricare il file che vogliamo scrivere sul dispositivo.

Menù "Visualizza" per le diverse presentazioni.



Presentazione in assembler del file letto o caricato.

Menù "Comandi".

ferente, come ad esempio MPLAB. In questo menù sono riportate anche due opzioni di informazione, che sono: "Verso di Inserimento", che rappresenta visualmente la posizione del programmatore e del dispositivo e "Informazioni sul Chip" che riporta le caratteristiche del dispositivo selezionato.

L'opzione "Aggiorna", aggiorna il monitor e si utilizza raramente.

Operazioni

Stiamo orientando l'utilizzo di questo software esclusivamente per scrivere, facendo accesso al menù "Comandi" possiamo vedere che permette anche altre opzioni che commenteremo di seguito:

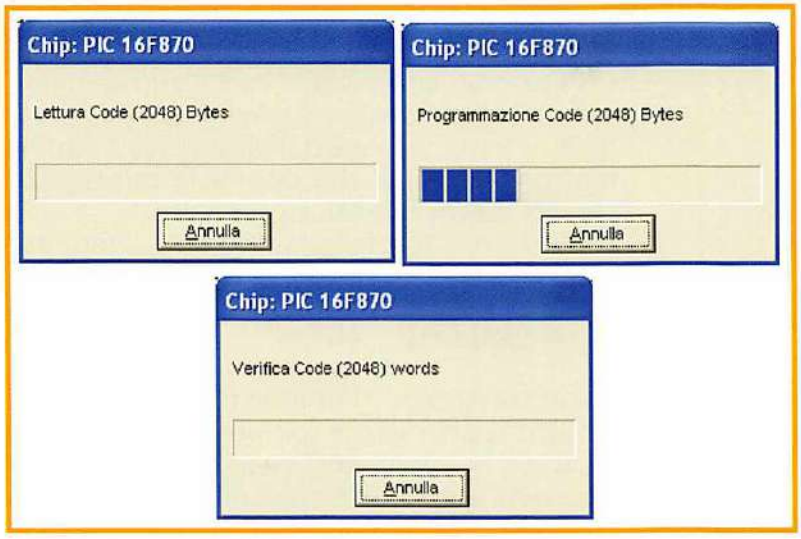
– Leggi Tutto: mediante questa opzione possiamo leggere il programma che si trova scritto sul microcontroller. Anche se a prima vista non sembra molto pratico dato che la presentazione è fatta in esadecimale, cambiando in assembler possiamo analizzare il programma contenuto dal dispositivo, nel caso in cui quest'ultimo non abbia codici di protezione attivati.

– Programma Tutto: opzione che permette di scrivere sul dispositivo il file selezionato. Nelle schede SmartCard si potrà utilizzare questa opzione direttamente, ma lavorando con il PIC prima di questa operazione dovremo cancellare precedentemente la memoria del PIC, e selezionare i bit della parola di configurazione. Fatto questo procederemo alla scrittura completa del microcontroller. Questa operazione solitamente richiede un'ulteriore conferma.

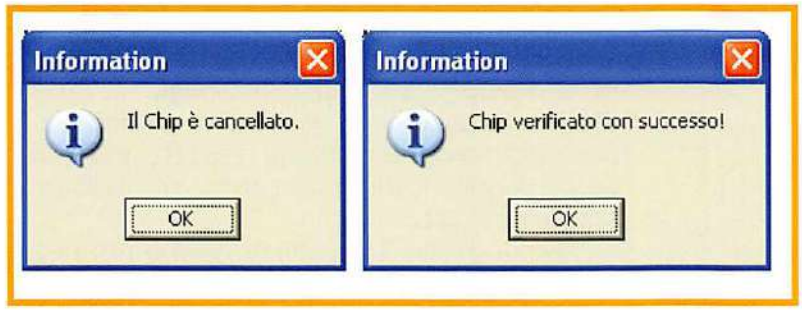
– Programma Fuses: quando vogliamo scrivere solamente la parola di configurazione e non vogliamo scrivere un programma, selezioneremo questa opzione. Si utilizza raramente.

– Cancella Tutto: passo precedente alla programmazione, che deve essere realizzata per assicurarci che il dispositivo non contenga dati residuali, posizioni di memoria che non saranno sovrascritte con il nuovo programma e che potrebbero causare problemi durante l'esecuzione dello stesso. Dato il rischio di questa operazione, normalmente il software chiede conferma dell'operazione.

– Verifica Blank: verificare che la can-



Stati delle operazioni da realizzare sul dispositivo.



Messaggi che informano dei risultati delle operazioni.

cellazione sia stata effettuata con successo.

– Verifica: verificare che la scrittura sia stata effettuata correttamente.

– SmartCard Wizard: passaggi e aiuti per lavorare con questo dispositivo.

In tutte le operazioni una barra di stato ci informerà dello stato del processo, come possiamo vedere nelle figure in alto.

La barra degli strumenti

Come nella maggior parte dei programmi IC-Prog presenta una barra con delle icone, per ottenere un accesso rapido alle operazioni più frequenti. Nella figura possiamo vedere le funzioni di ogni pulsante, che si possono anche osservare posizionando il mouse sul pulsante e attendendo un istante per la nota.

Riassumendo

Quando vogliamo scrivere un programma sul microcontroller la prima cosa che dobbiamo fare è verificare che la scheda di scrittura sia configurata correttamente. Sarà necessario collegare a questa il PC e far partire il programma IC-Prog.

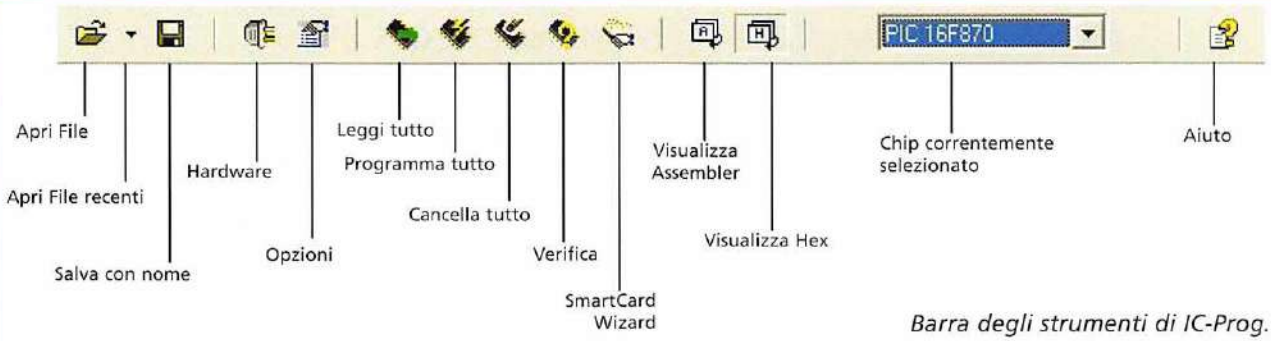
Continueremo cancellando il contenuto del PIC e vi consigliamo di verificare sempre la cancellazione, o con l'opzione apposita, o leggendo il contenuto del dispositivo e verificando il risultato dell'operazione.

Bisogna aprire il file che si desidera caricare e verificare che i bit della parola di configurazione siano selezionati in modo corretto.

A questo punto si può procedere a programmare tutto, verificando questo processo e leggendo in seguito il contenuto del dispositivo per vedere se concorda con ciò che desideravamo scrivere.

Mediante delle finestre il software ci informerà del risultato delle operazioni eseguite.

IC-Prog è un software pratico, semplice e di libera distribuzione.



Barra degli strumenti di IC-Prog.

IC-PROG: leggere e salvare

Abbiamo visto il software IC-PROG; ora faremo un po' di pratica con alcuni esempi molto semplici. Affineremo le nostre conoscenze esercitandoci con un esempio reale in cui utilizzeremo le opzioni più comuni del programma. Per poter realizzare gli esempi presentati qui di seguito, sarà necessario disporre del cavo di collegamento e del laboratorio.

Vediamo ora la procedura di lettura del programma che si trova scritto sul microprocessore.

Configurazione dell'hardware

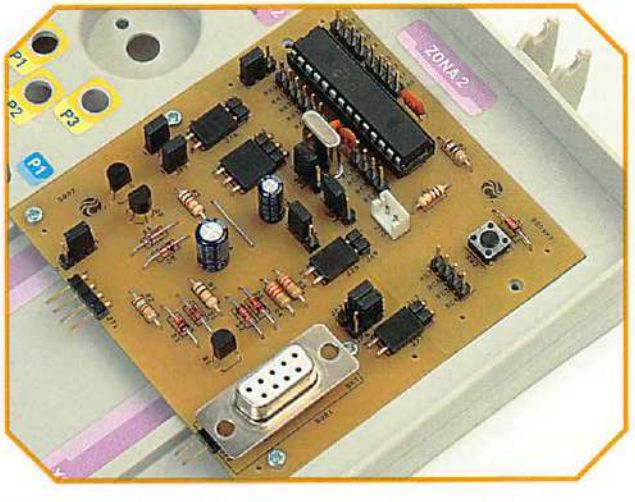
Dobbiamo collocare il PIC nel suo zoccolo all'interno del laboratorio, se non era già stato montato, e configurare i ponticelli per eseguire la lettura.

Per primi configureremo i ponticelli che si trovano sulla scheda DG06 contenente il PIC. JP1 discrimina tra i due possibili ingressi al pin 1 del PIC, quello di reset, JP2 sceglie tra i diversi collegamenti della massa e JP3 tra le diverse prese di alimentazione. Quando lavoriamo con la porta seriale del computer, è la porta stessa che ha sui terminali una tensione di riferimento che viene utilizzata per alimentare il circuito. Queste tensioni arrivano ai ponticelli citati in precedenza e, attraverso questi, selezioneremo il tipo di alimentazione per il PIC. Per utilizzare l'alimentazione che arriva dal cavo del PC i ponticelli dovranno essere situati sui terminali 1 e 2 di JP1, JP2 e JP3.

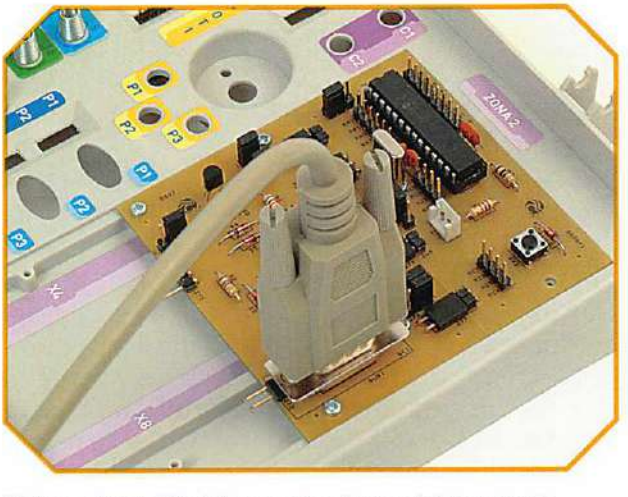
La scheda di scrittura (DG07) dispone anche di 4 ponticelli che vi verranno spiegati più avanti e che si utilizzano per lavorare con le schede SmartCard, quindi è indifferente la configurazione che hanno (JP4, JP5, JP6 e JP7). Sarà necessario impostare i ponticelli JP8 e JP9, dato che ci permettono di accedere al PIC tramite i terminali RB6 e RB7.

Ora colleghiamo il cavo alla porta seriale del computer, assicurandoci di collegarlo alla

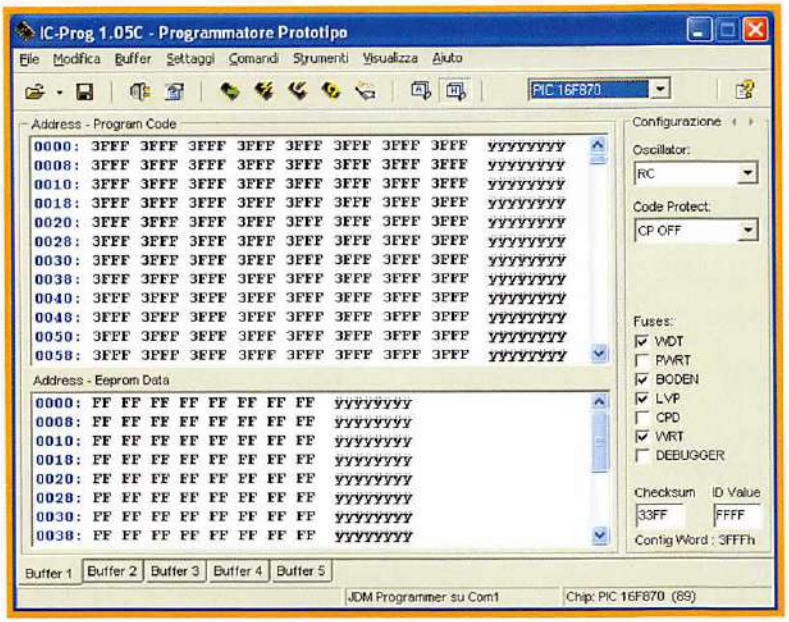
Colleghiamo il cavo al computer.



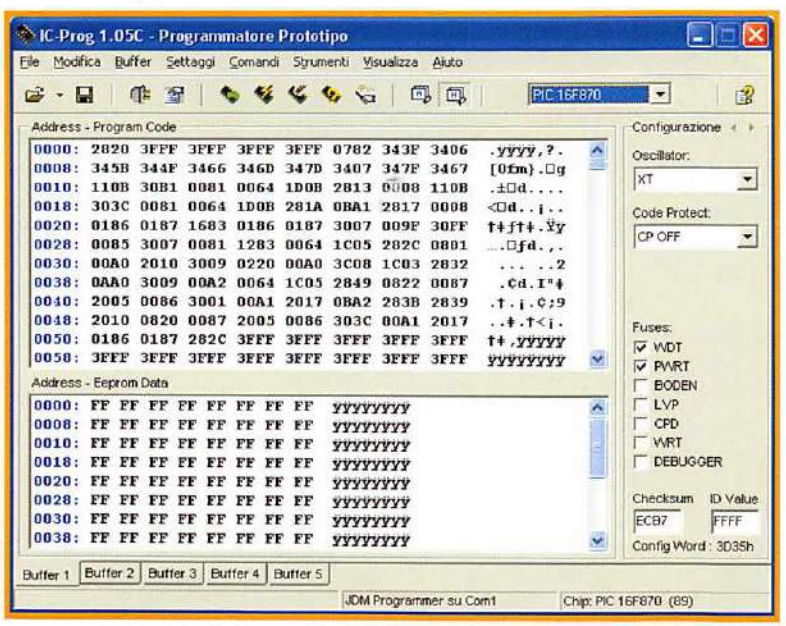
Configurazione dei ponticelli del laboratorio.



L'altra estremità del cavo si collega al laboratorio.



Videata iniziale del programma, in cui potremo verificare le opzioni configurate.



Aspetto di IC-PROG dopo aver terminato il processo di lettura.

porta che abbiamo configurato su IC-PROG (normalmente la COM1). La porta seriale del computer è un connettore DB9 maschio, quindi l'estremo del cavo da collegare sarà quello che ha un connettore DB9 femmina. Fatto questo, dobbiamo collegare l'altro estremo del cavo al laboratorio, come possiamo vedere nella figura.

Configurazione del software

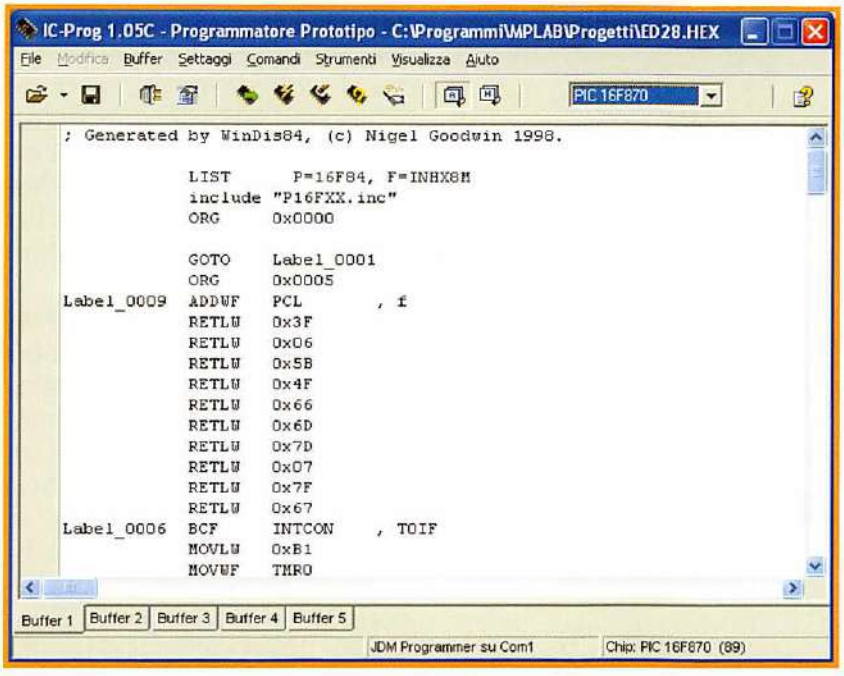
Se abbiamo eseguito correttamente i passaggi precedenti, possiamo far partire il software di scrittura. Verifichiamo che sulla linea inferiore del programma si trovino selezionati sia il PIC16F870 che il programmatore e la porta di comunicazione su cui abbiamo collegato il cavo (JDM Programmer su COM1). Questa linea è puramente informativa, dato che se abbiamo bisogno di cambiare qualche parametro dobbiamo accedere al menù come abbiamo visto nel fascicolo precedente.

Se selezioniamo l'opzione Leggi tutto della barra degli strumenti, premendo F8, o sul menù Comandi, sarà come dire al programma di andare a leggere sul microcontroller il programma che si trova scritto all'interno. Il software provvederà a leggere il dispositivo indicando, mediante una barra di stato, come avanza il processo di lettura. Nella figura possiamo vedere la finestra che indica lo stato di questo processo.

Risultato del processo di lettura

Quando termina il processo di lettura vedremo che la finestra contenente il codice del programma varia alcuni valori e otterremo, come risultato, una videata simile a quella

Il software sta leggendo il microcontroller.



Possiamo vedere il contenuto del PIC in linguaggio assembler.

```
Uscita:         movf    Numero,W        ;Acquisisce il numero casuale
                movwf   PORTC           ;Porta il numero casuale in binario sui LED
                call    Tabella         ;Lo converte in 7 segmenti
                movwf   PORTB           ;Uscita sul display
                movlw   d'60'
                movwf   Delay_Cont      ;Inizializza variabile di temporizzazione
                call    Delay_Var       ;Temporizza 3"
                clrf    PORTB           ;Scollega l'uscita
                clrf    PORTC           ;Scollega l'uscita sui LED
                goto    Loop
```

```
Label_0008  MOVF    0x20      , W
            MOVWF   0x07
            CALL    Label_0009
            MOVWF   PORTB
            MOVLW   0x3C
            MOVWF   0x21
            CALL    Label_0004
            CLRF    PORTB
            CLRF    0x07
            GOTO    Label_0005
```

Differenze da codice creato dal programmatore e quello recuperato leggendo il PIC.

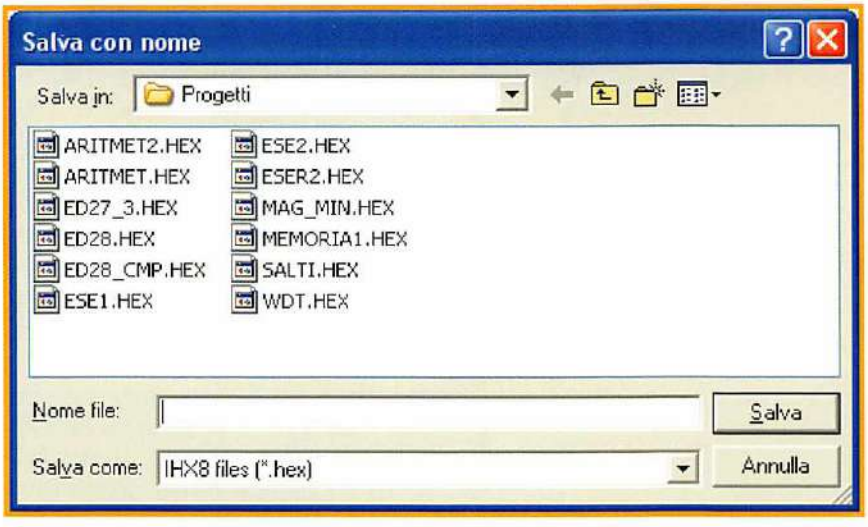
presentata nella figura della pagina precedente.

Apparentemente non abbiamo ottenuto nulla che ci possa servire per sapere se ciò che è stato letto nel PIC corrisponde al programma che era scritto, dato che vediamo solamente una serie di numeri e lettere che non sappiamo identificare. Quella che abbiamo davanti a noi è una rappresentazione in esadecimale del contenuto degli indirizzi di memoria del programma del microcontroller. Visto che i dati sono rappresentati in esadecimale non potremo capire il programma, però questo semplice software presenta l'opzione di poter vedere gli stessi dati codificati in linguaggio assembler. Per poter vedere il contenuto del chip in linguaggio assembler possiamo accedere al menù Vedi, o selezionare il pulsante della barra degli strumenti con la lettera "A".

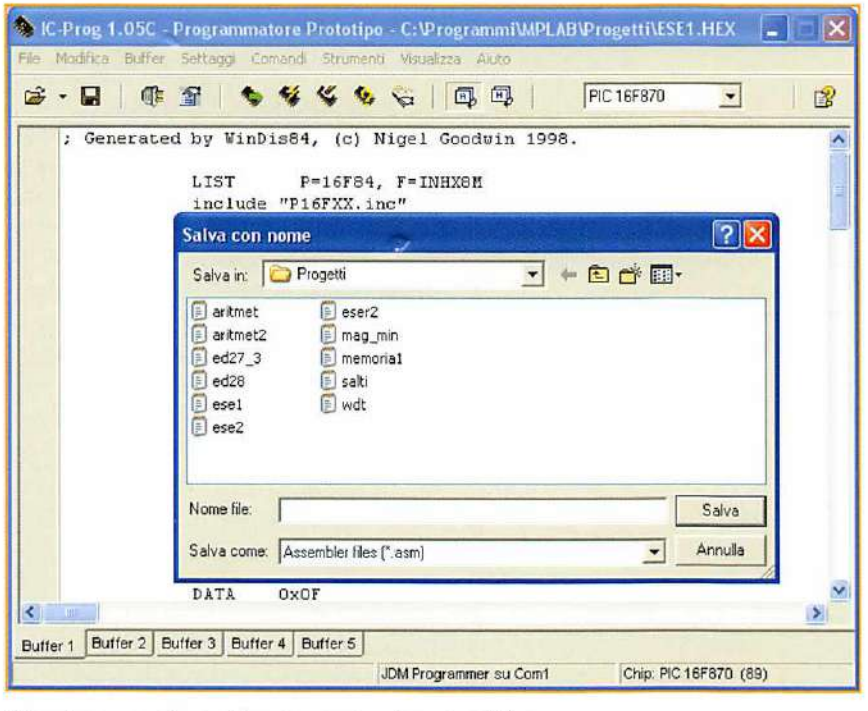
La videata di IC-PROG cambierà stato e ci presenterà il codice in linguaggio assembler, come mostrato dalla figura.

Spostandoci sul programma potremo identificare le istruzioni, però noteremo anche le molte differenze rispetto al programma che sappiamo di aver caricato sul PIC. Per prima cosa vediamo che sul programma letto non esistono commenti. I commenti sono un riferimento per il programmatore, in quanto lo aiuteranno a capire il codice se avrà bisogno di lavorare nuovamente con esso in futuro, però non vengono tradotti in codice macchina, perché non si scrivono sul microcontroller. Noteremo anche che i nomi delle variabili e delle etichette non appaiono dalla lettura del dispositivo, in quanto i dati sono elaborati in formato esadecimale e per le etichette viene utilizzata una numerazione propria. Troveremo anche altre differenze, ma non sono significative per la comprensione del programma.

Nella figura della pagina precedente possiamo vedere un esempio delle differenze tra una parte del codice creato per il programmatore e scritto sul PIC (superiore) e la stessa parte di codice letta direttamente sul microcontroller.



Finestra che appare quando scegliamo di selezionare il programma con la rappresentazione in esadecimale.



Finestra per salvare il programma in assembler.

Salvare il programma

IC-PROG ci permette di salvare il programma letto sul microcontroller grazie al quale si possono recuperare o copiare programmi per lavorare successivamente con essi. Per poter utilizzare questa interessante opzione dobbiamo accedere al menù File e selezionare "Salva con nome", o cliccare sul pulsante con il simbolo di un dischetto che si trova sulla barra degli strumenti.

Ci sono due modi per salvare il programma letto: in codice esadecimale o in assembler. Se selezioniamo l'opzione di salvare quando sulla videata del programma abbiamo la rappresentazione in esadecimale, ci apparirà la finestra che vediamo nella figura. In questa finestra dobbiamo inserire la posizione dove vogliamo salvare il file (è consigliabile che sia quella creata per salvare tutti i nostri progetti) e il nome che vogliamo assegnare a esso. Possiamo vedere che l'estensione del file che stiamo salvando è predeterminata come ".hex", ovvero salveremo il file in codice macchina. Un file con questa estensione non è interpretabile da un programmatore ma è adatto a essere scritto su un altro chip.

Se selezioniamo la presentazione in assembler vista in precedenza, e selezioniamo l'opzione salva, la videata che apparirà è quella mostrata nella figura in basso.

Questa finestra è simile alla precedente con la differenza che ora il file ha un'estensione predeterminata ".asm". Il file verrà salvato in codice assembler e il programmatore potrà, quindi, aprirlo con qualsiasi editor di testo, studiarlo o modificarlo. Per tornare a scrivere il programma sul chip sarà necessario assemblarlo per ottenere il file in codice macchina.

IC-PROG: cancellazione e programmazione

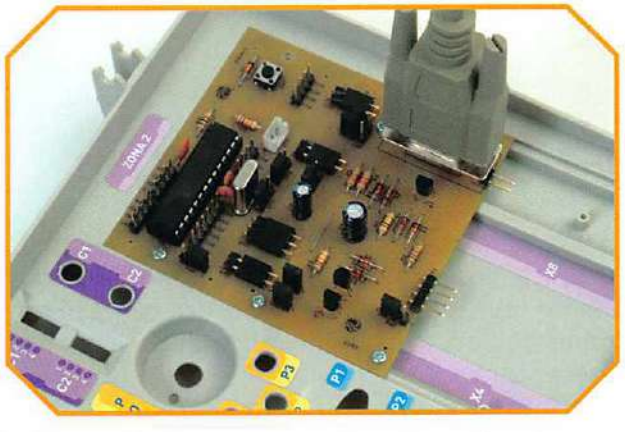
Abbiamo imparato una delle principali funzioni di IC-PROG, dobbiamo anche studiare come cancellare e programmare il dispositivo. Con questo completiamo l'analisi del programma e siamo pronti per eseguire qualsiasi operazione sul microcontroller.

Configurazione hardware per la cancellazione e la scrittura

Come abbiamo spiegato in precedenza il processo di scrittura e cancellazione del PIC 16F870, si esegue mediante segnali elettrici. Il laboratorio è stato progettato per fare in modo che non sia necessario togliere il chip dal suo alloggiamento per eseguire qualsiasi azione, e questo vale sia per le procedure sopracitate che per quelle di funzionamento.

La configurazione hardware e software per eseguire la cancellazione del PIC è la stessa che abbiamo applicato per la lettura, quindi non dobbiamo cambiare nulla. I jumpers JP1, JP2 e JP3 si alimentano in modo che l'alimentazione al PIC arrivi tramite il cavo di collegamento al PC. I ponticelli quindi saranno impostati sui pin 1 e 2 dei connettori.

I ponticelli su JP8 e JP9 devono essere inseriti, e in ultimo dobbiamo collegare il cavo al laboratorio e al PC. Nell'immagine della figura possiamo vedere l'aspetto del nostro laboratorio dopo l'avvenuta preparazione per lavorare con IC-PROG.

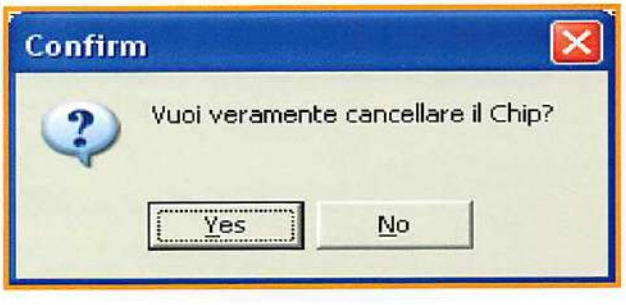


Il laboratorio deve essere configurato per la scrittura e collegato con il cavo al PC.

Cancellazione del dispositivo

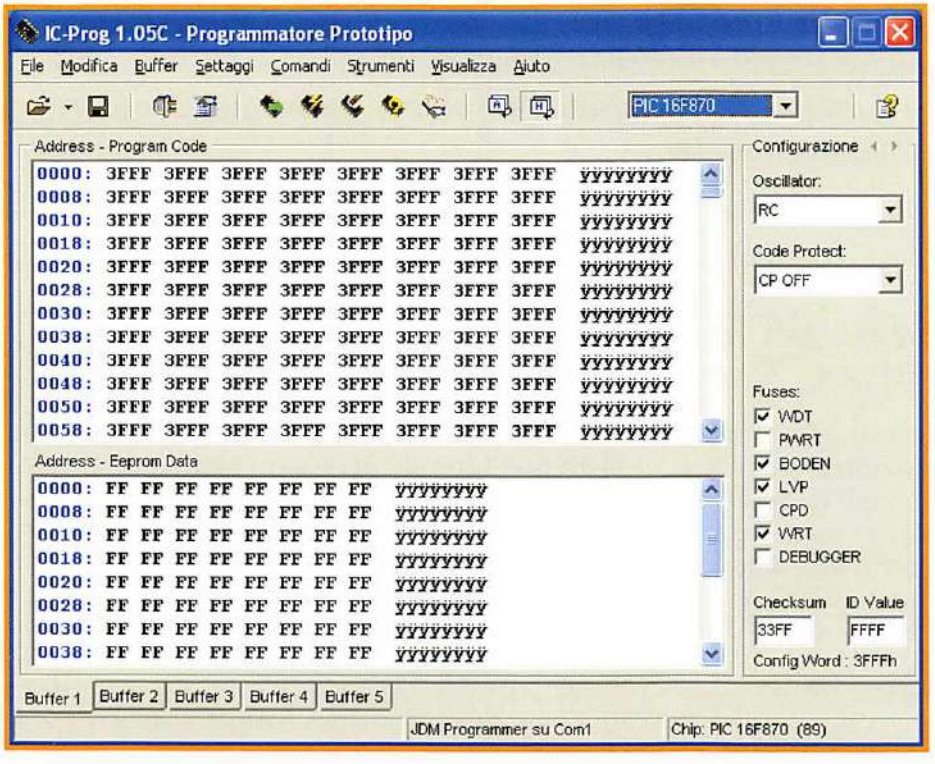
La cancellazione del contenuto del dispositivo si esegue facendo accesso al menù Comando e selezionando Cancella tutto o mediante il pulsante predisposto a questo scopo sulla barra degli strumenti. Prima di cancellare il dispositivo è consigliabile salvare il programma o i dati che contiene nel caso siano presenti, nella directory creata per contenere i nostri progetti. Selezionando l'opzione di cancellazione, dato che è un processo critico e pericoloso il software ci fa una domanda per confermare se realmente si desidera eseguire l'operazione. Per cancellare il PIC dobbiamo rispondere "Yes" e al termine di questo processo ci informerà della fine dell'operazione mediante una finestra come quella mostrata nella figura in basso.

Per verificare che il processo sia stato eseguito con successo, oltre alle opzioni di verifica che

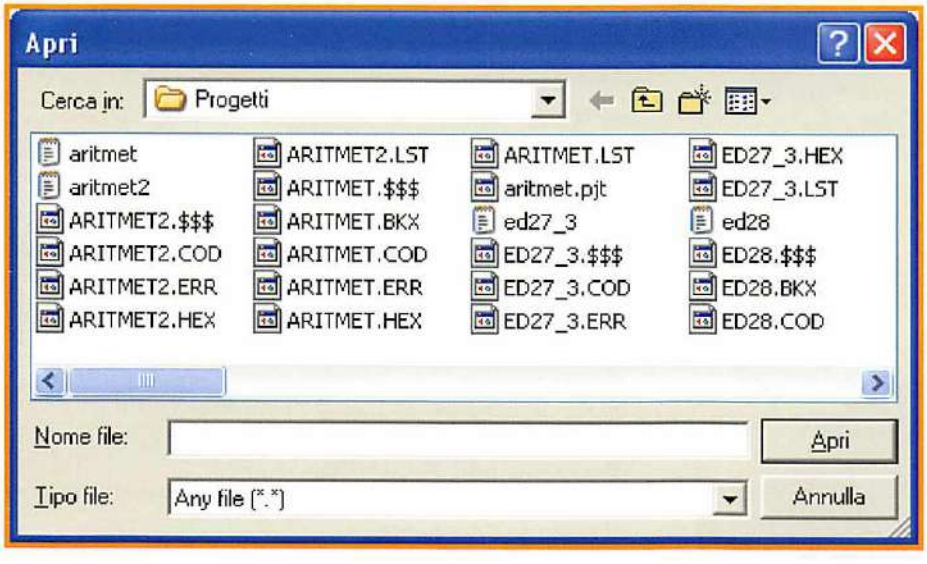


Richiesta di conferma per procedere alla cancellazione del chip.

Messaggio che ci informa che il processo di cancellazione è stato eseguito con successo.



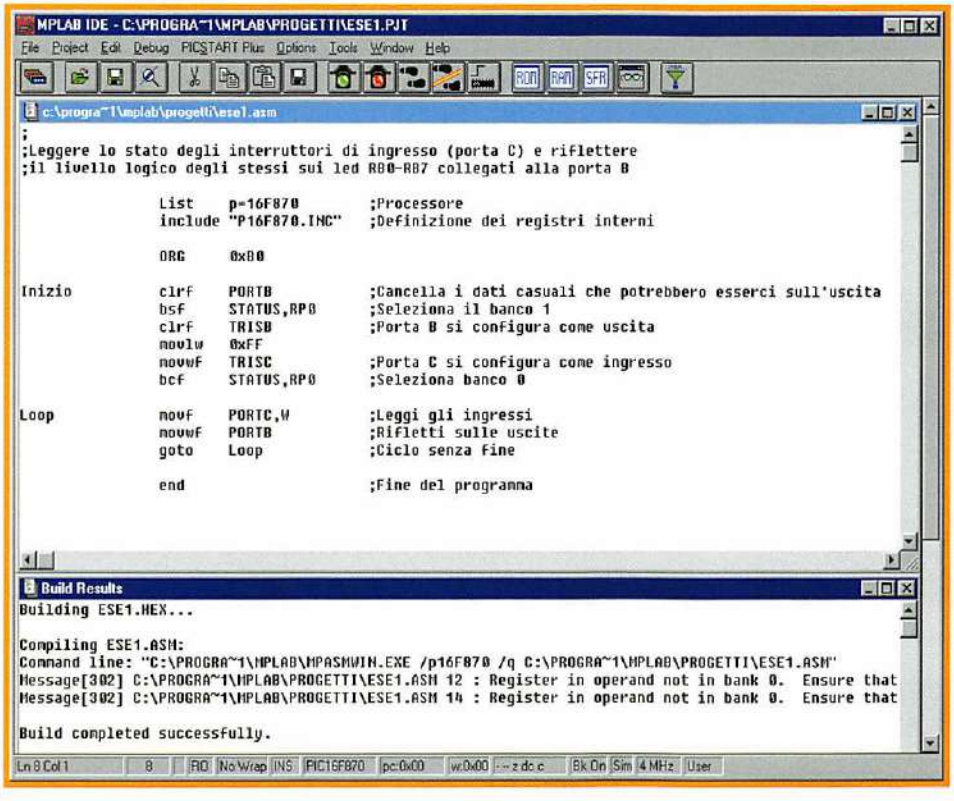
Videata di IC-PROG dopo la lettura di un dispositivo che non contiene niente, cioè che è stato cancellato.



Carichiamo un programma per la scrittura successiva.

Finestra che appare quando cerchiamo di caricare un file che non corrisponde.

presenta il programma stesso, è consigliabile procedere a una lettura del dispositivo. Se facciamo questo, possiamo verificare come al termine dell'operazione il codice di programma, presenterà solo dei valori 3FFF in tutte le sue posizioni di memoria, quindi potremo verificare che il dispositivo è stato cancellato correttamente.

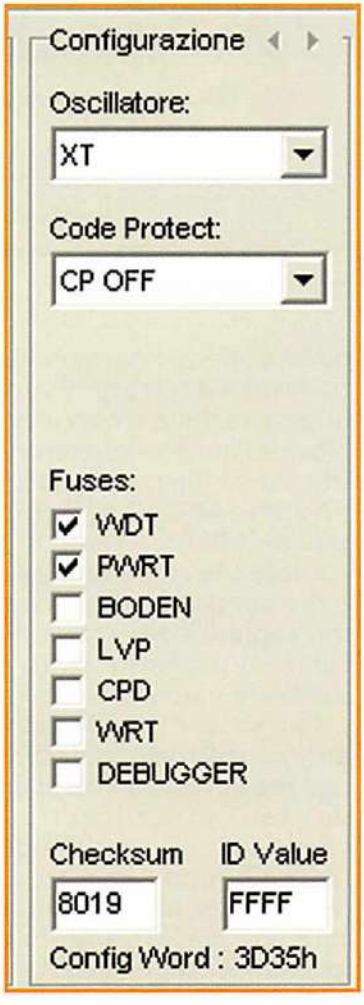


Risultato della compilazione con MPLAB.

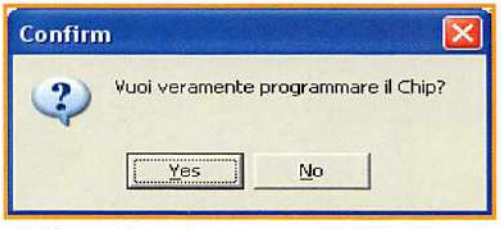
Se passiamo nuovamente in assembler possiamo vedere come si presenta in questo linguaggio il contenuto del PIC, che in questo caso si trova vuoto.

Scrittura del dispositivo

Per scrivere un programma sul PIC per prima cosa dobbiamo selezionare quale programma vogliamo scrivere. A questo scopo puntiamo sul menù File e selezioniamo apri File, oppure clicchiamo il pulsante a forma di cartella della barra degli strumenti. Ci apparirà per default l'indirizzo in cui stiamo lavorando, e l'estensione dei file fra cui possiamo scegliere, in funzione del dispositivo che vogliamo scegliere. Nel caso che stiamo trattando, il dispositivo è il PIC 16F870 su cui possiamo scrivere solamente un file in codice macchina, in altre parole con estensione ".hex". Se cercassimo di caricare un file con un'estensione differente, ad esempio ".asm" o ".cod" apparirà una finestra come quella della pagina accanto, in cui il software ci chiede se deve cambiare i byte per convertire il file. Il software convertirà il file nel suo formato, però questo non sarà valido per la scrittura. Per far pratica con un esempio lavoriamo con il primo degli esercizi inclusi nel CD allegato a



Configurazione per la scrittura.



Richiesta di conferma per scrivere.

Stato del processo di scrittura.

Errore di programmazione.

questa opera. Inseriamo il Cd nel lettore e puntiamo alla directory "Esercizi e applicazioni". In questa cartella troveremo 6 esercizi in assembler, dei quali sceglieremo il primo "ese1.asm". Dato che i file non sono compilati sarà necessario aprire MPLAB e assemblare il codice per ottenere il file risultante in codice macchina. Ricordate che sarà necessario creare un progetto e che per poter inserire un file in esso lo dovremo copiare nella stessa directory dove è stato creato il progetto. Fatto questo possiamo assemblare e generare il file ".hex".

Per scrivere il file generato lo dobbiamo aprire così come vi abbiamo spiegato in precedenza e, dopo averlo aperto, verificare che sia stato caricato correttamente. A questo scopo potremo vedere gli indirizzi di memoria che hanno cambiato i loro valori, o passare alla presentazione in esadecimale e verificare così che il programma caricato sia quello desiderato.

Prima di scrivere il programma sul PIC dobbiamo configurare la scrittura. Nella parte destra del programma si trovano i parametri che dobbiamo configurare al momento di scrivere, come possiamo vedere nella figura della pagina precedente.

La prima cosa che dobbiamo configurare è l'oscillatore. Esistono quattro possibilità: LP, XT, HS e RC. Avendo a disposizione sul laboratorio un cristallo di quarzo per ottenere la frequenza dobbiamo configurare l'oscillatore come XT. Il parametro successivo da configurare è la protezione del codice, questo è molto importante, infatti selezionando la protezione del codice non potremo riscrivere nuovamente il nostro PIC, verrà protetta la zona determinata per fare in modo che non possa essere sovrascritta. Quindi noi selezioneremo CP OFF, cioè il codice di programma non è protetto.

Infine, dobbiamo configurare la Parola di Configurazione. Se andassimo a rivedere la teoria del PIC, quando abbiamo studiato la Parola di Configurazione, potremmo identificare ognuno dei bit da configurare con IC-PROG. In questo caso lasceremo selezionato WDT e PWRT.

Configurata correttamente la scrittura caricheremo il programma sul microcontroller e a questo scopo selezioneremo programma tutto all'interno del menù Comando, cliccheremo F5 o selezioneremo l'apposito pulsante della barra degli strumenti.

È sempre consigliabile cancellare il dispositivo prima di programmarlo, per evitare che rimanga qualche posizione di memoria con dei valori residui. Nel nostro caso la cancellazione è già stata eseguita in precedenza, quindi possiamo scrivere direttamente.

Quando selezioniamo scrivi il software, si presenta una richiesta di conferma, alla quale risponderemo "yes" inizierà quindi il processo di scrittura. Durante la scrittura, una barra di stato ci indicherà come si sta sviluppando il processo, se si verifica un errore nella programmazione apparirà un messaggio come quello in figura. Quando la scrittura sarà terminata, potremo verificare se è stato programmato correttamente, facendo una lettura del contenuto del dispositivo.

Non dimentichiamo che quando lavoreremo con le SmartCard i passaggi saranno identici, tranne per il dispositivo e alcuni parametri che sono differenti; ma i processi di lettura, cancellazione e scrittura sono molto simili.

Esercizio 1: ingressi e uscite del programma

Faremo alcuni esercizi lavorando con essi come se fossero dei veri progetti. Partiremo da zero e attraverso le conoscenze e con l'aiuto dei programmi che abbiamo imparato a utilizzare, arriveremo fino a provare il lavoro eseguito in modo pratico e reale sul laboratorio, verificandone il corretto funzionamento.

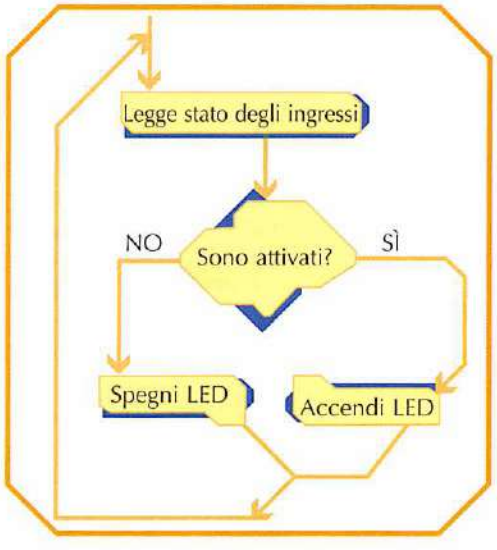
Enunciato

Si tratta di realizzare un esercizio in cui si legge lo stato logico degli elementi collegati alla porta C, riflettendo questo stato su diodi LED collegati alla porta B.

Questo semplice esercizio lavora con gli ingressi/uscite del nostro microcontroller, però ci servirà soprattutto per far pratica con quanto abbiamo visto fino a ora acquisendo così maggior metodo e scioltezza nella realizzazione di un progetto.

Organigramma

Se ricordiamo i passaggi da seguire per affrontare un progetto, sappiamo che il primo di essi è l'analisi dell'enunciato, e che la forma migliore per effettuare questo lavoro è l'utilizzo di un organigramma. Se osserviamo l'organigramma della figura vediamo che con la sola lettura del valore dell'ingresso e con il passaggio di quest'ultimo sull'uscita il problema sarebbe risolto.



Organigramma dell'applicazione.

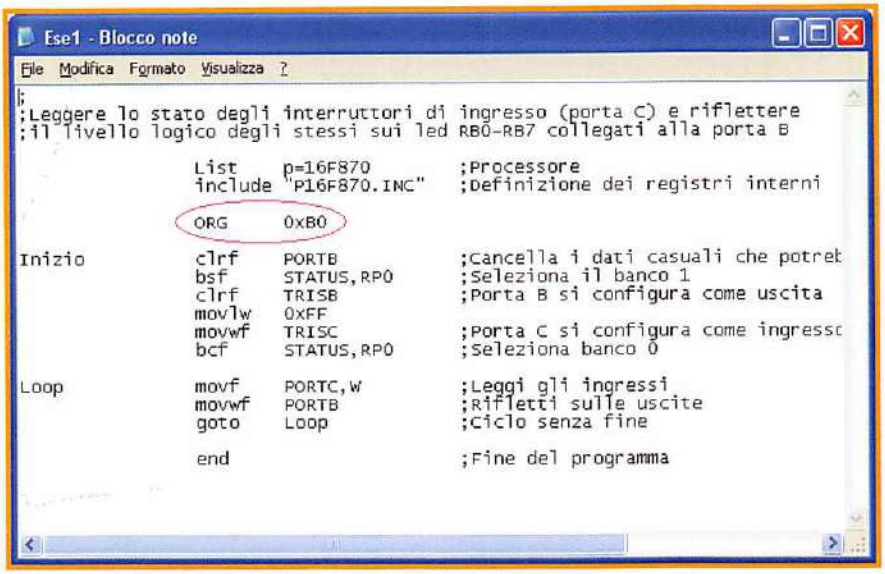
Codice

Per risolvere il progetto apriremo un editor di testo (ad esempio blocco note) e inizieremo a scrivere il nostro codice. Ricordate che è molto utile l'utilizzo dei commenti che chiariscono la funzionalità del programma e delle istruzioni.

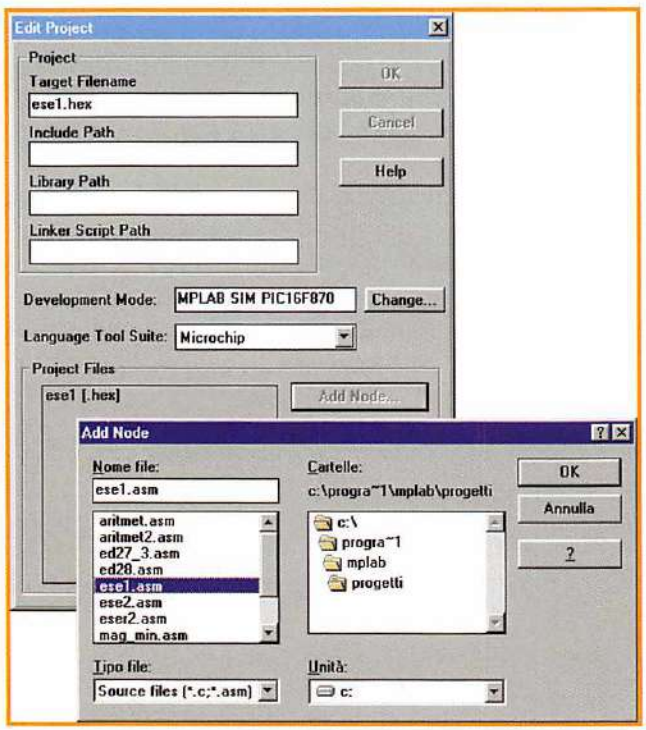
Le prime linee di codice sono uguali in tutti i programmi: il tipo di processore e la libreria che contiene le definizioni dei registri. Le linee successive sono le direttive per organizzare la memoria e nelle quali definiamo le posizioni di memoria in cui verrà contenuto il codice del programma.

Possiamo decidere che il programma inizi con la configurazione del dispositivo. In questo caso dobbiamo configurare la porta B come uscita e la porta C come ingresso. Conviene pulire la porta di uscita in modo da elimi-

```
;
;Leggere lo stato degli interruttori di ingresso (porta C) e riflettere
;il livello logico degli stessi sui led RB0-RB7 collegati alla porta B

        List    p=16F870        ;Processore
        include "P16F870.INC"   ;Definizione dei registri interni

        ORG     0x0
        goto    inizio
        ORG     0x05

Inizio  clrf    PORTB           ;Cancella i dati casuali
                                che potrebbero esserci sull'uscita
        bsf     STATUS,RP0      ;Seleziona il banco 1
        clrf    TRISB           ;Porta B si configura come uscita
        movlw   0xFF
        movwf   TRISC           ;Porta C si configura
                                come ingresso
        bcf     STATUS,RP0      ;Seleziona banco 0

Loop    movf    PORTC,W         ;Leggi gli ingressi
        movwf   PORTB           ;Rifletti sulle uscite
        goto    Loop            ;Ciclo senza fine

        end                     ;Fine del programma
```



Soluzione che si trova sul CD pensata per lavorare con il Bootloader.



Aggiungiamo il file al progetto.

Creiamo un progetto per contenere il nostro programma.

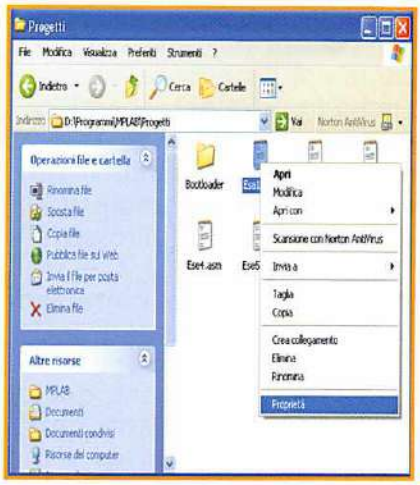
nare eventuali valori residui che potrebbero confondere il risultato finale.

Il funzionamento dell'applicazione è molto semplice: leggere l'ingresso e passare questo dato all'uscita. Questo lo ripeteremo in modo consecutivo per aggiornare frequentemente i valori. Nella figura possiamo vedere il programma risultante.

Nel CD fornito con questa opera possiamo trovare un programma che risponde a questo enunciato. Si tratta "ese1.asm" risolve il progetto, ma ha una particolarità. È scritto a partire dalla posizione di memoria 0Bh, questo perché è previsto l'utilizzo del programma residente Bootloader che vi spiegheremo più avanti. Se esiste un programma residente nel PIC, il codice che memorizzeremo non deve sovrascrivere il programma, quindi dovrà essere ubicato in una posizione di memoria libera. Se questo programma viene scritto direttamente sul PIC senza tener conto di questo, il programma residente non funzionerà correttamente.

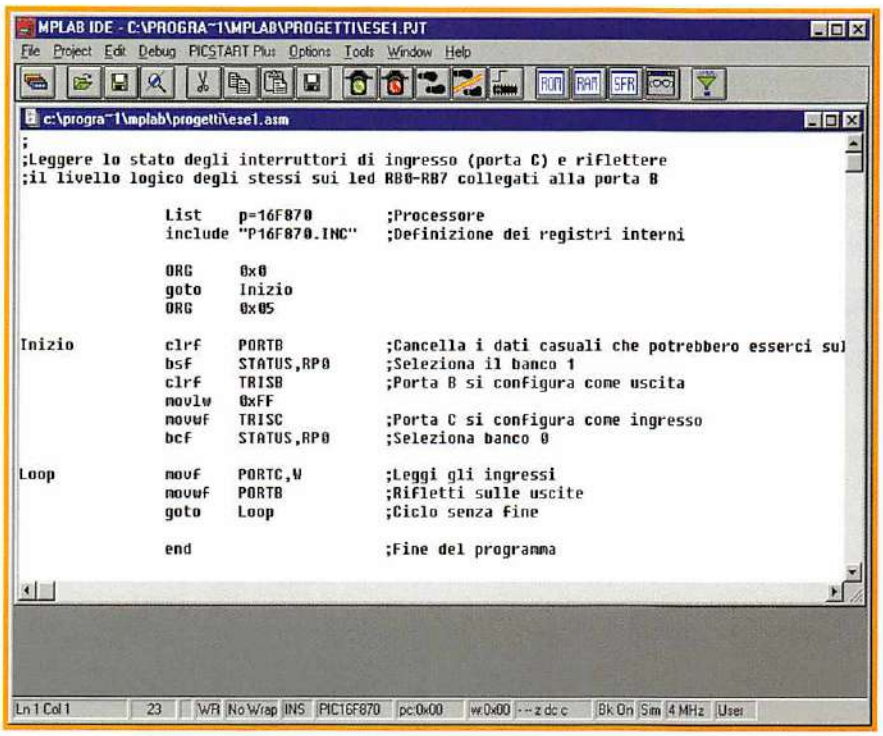
Compilazione

Per verificare che il programma fatto corrisponda ai requisiti dell'enunciato ne simuleremo il funzionamento mediante MPLAB. La prima cosa da fare è salvare il file creato con il codice nella directory dove si trovano i progetti di lavoro. Ricordate che il file deve essere salvato con l'estensione ".asm". In seguito apriremo MPLAB e creeremo un progetto come mostrato nella figura. Il nome del progetto non deve contenere più di otto lettere, nel caso fosse più lungo apparirebbe un messaggio di errore che ci comunicherà che il nome del file che stiamo cercando di creare non è valido. Fatto questo apriremo edit pro-



Clicchiamo con il pulsante destro del mouse sul file.

Disabilitiamo l'opzione di sola lettura.



Aspetto di MPLAB con il nostro file caricato.

get e aggiungeremo al progetto il file in assembler (.asm) con il nostro codice. In seguito dovremo aprire il file per poterlo vedere sul display, selezionando dal menù File l'opzione Open.

Nel caso venga copiato nella cartella dei progetti il file fornito con il CD, prima di modificarlo dovremo disabilitare all'interno delle proprietà del file stesso, l'opzione di Sola lettura. Per fare questo dovremo, tramite Explorer, cliccare il file con il pulsante destro del mouse e selezionare Proprietà. Apparirà una finestra simile a quella della figura, in cui dovremo lasciare libera l'opzione Sola lettura, applicando e confermando poi l'operazione.

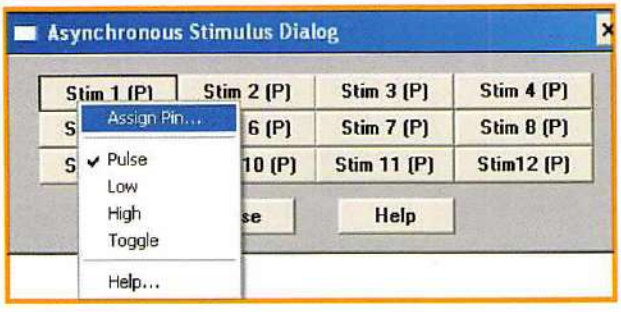
Con il file aggiunto nel nostro Progetto e visualizzato sul monitor, MPLAB avrà l'aspetto della figura in basso.

Se compiliamo il programma selezionato con l'opzione Build All sul menù Project, verificheremo come il nostro programma venga compilato e assemblato senza errori. Grazie a questo passaggio verrà generato il file in codice macchina ".hex" necessario per scrivere il PIC, prima dell'operazione di scrittura però è consigliabile simulare il funzionamento del programma per verificare che sia conforme a quanto desiderato.

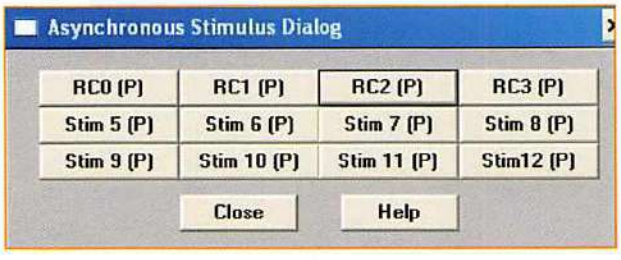
Simulazione

Per simulare questo esercizio apriremo la finestra di "Special Function Register" , creeremo una finestra in cui poter vedere unicamente i registri che ci interessano per il risultato finale (PORTB e PORTC) e programmeremo il nostro simulatore di stimoli. Se dal menù Debug selezionamo l'opzione Simulator Stimulus e all'interno di questa selezioniamo Asynchronus Stimulus apparirà una finestra come quella della figura della pagina successiva. Se clicchiamo con il pulsante destro del mouse sul primo stimolo e selezioniamo l'opzione Assign Pin ci apparirà una lista con diversi pin (terminali) che si possono assegnare; nel nostro caso assegneremo RC0. Ripeteremo i passaggi per il resto dei terminali del-

la porta C, anche se per realizzare la simulazione sarà sufficiente assegnare i primi quattro. In questo modo il simulatore di stimoli asincroni assumerà l'aspetto mostrato dalla figura.



Simulatore di stimoli asincroni.

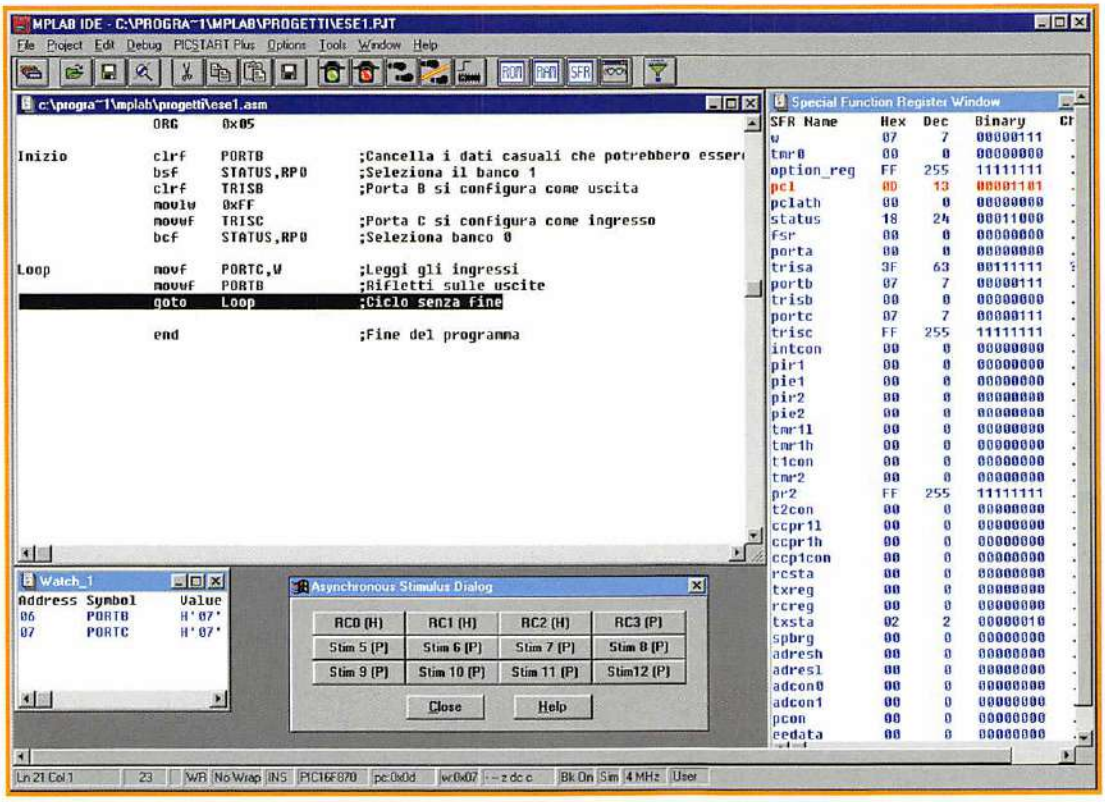


Dopo la configurazione il simulatore di stimoli avrà questo aspetto.

Possiamo ora iniziare la simulazione. Se clicchiamo F7 o le diverse opzioni viste nei fascicoli precedenti per la simulazione passo a passo, osserveremo nella finestra dei registri speciali, come si configurano le porte e i registri associati man mano che eseguiamo il programma. Osservate che quando cambia un registro quest'ultimo è visualizzato in rosso nella finestra.

Quando entriamo nel ciclo apparentemente questo viene eseguito correttamente anche se non vediamo cambiamenti dato che all'ingresso (porta C) troviamo valore zero, quindi l'uscita è allo stesso valore. Sul pulsante del simulatore possiamo selezionare il valore che desideriamo far assumere ai terminali di ingresso. Quindi con il pulsante destro clicchiamo sul pin e impostiamo High, questo significa che avremo un livello alto su questo ingresso. Quando clicchiamo il pin con il pulsante sinistro diamo l'ordine di aggiornare il valore, quindi se continuiamo la simulazione passo a passo mediante F7 (dovremo cliccare una volta sulla finestra con il codice per far riprendere la simulazione) vedremo come cambia l'ingresso e a sua volta cambia l'uscita.

Cambiando i valori del simulatore di stimoli vedremo anche come cambia l'uscita, quindi daremo per buona la simulazione e ci prepareremo per scrivere il PIC ed eseguire il montaggio sul laboratorio, anche se questo lo vedremo in un fascicolo successivo.



Verifica della corretta esecuzione della simulazione.

Esercizio 2: programma combinazionale, il programma

Con questo esercizio, utilizzato come esempio quando abbiamo studiato il repertorio delle istruzioni, faremo pratica con i salti condizionali e continuiamo ad accrescere la nostra confidenza con lo sviluppo dei progetti completi.

Enunciato

Mediante questo esempio si vogliono attivare i LED (collegati alla porta B del PIC) secondo lo stato di due interruttori (collegati ai terminali RC0 e RC1) e conformemente alla tabella della verità.

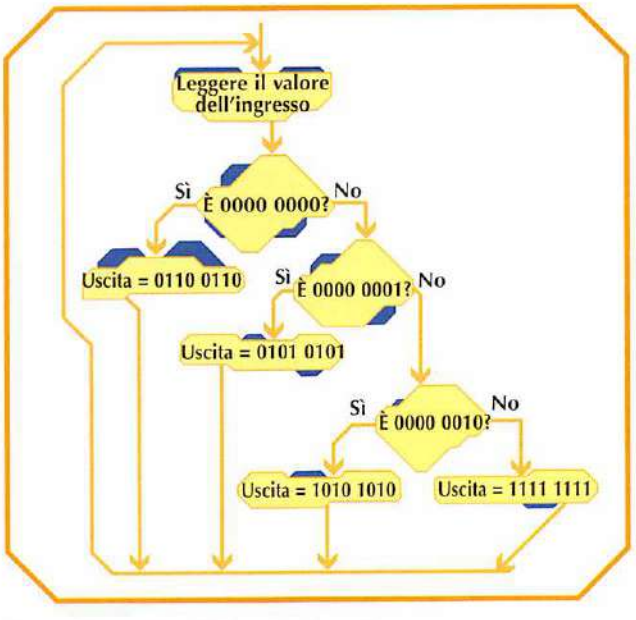
RC1	RC0	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	0	1	1	0	0	1	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1

Tabella della verità per assegnare le uscite in funzione degli ingressi.

Organigramma

Esistono diversi modi di risolvere questo esercizio. Predisponiamo due alternative diverse per arrivare alla soluzione.

Soluzione 1: se leggiamo l'ingresso e lo identifichiamo, comparandolo con dei valori potremo sapere a quale uscita è assegnato. Questo corrisponde a una struttura CASE in un linguaggio di programmazione di alto livello. Dato che questo non si può fare in assembler il valore di ingresso dovrà essere comparato con tre dei possibili valori per contemplare tutte le opzioni. Osservando l'organigramma è possibile capire meglio questa soluzione. Questo tipo di soluzione presenta una possibile sorgente di errori in quanto se sugli altri terminali della porta fossero presenti dei valori residuali, al momento di eseguire la comparazione che in realtà sarà un'operazione di sottrazione, potremmo ottenere un valore errato. Per evitare questo dovremo fare un'operazione AND per sicurezza, come mostrato nella figura della pagina successiva.



Organigramma della soluzione 1.

Soluzione 2: dato che l'ingresso è composto da due bit, ne leggeremo uno, e dopo averlo identificato leggeremo l'altro, per poter fornire l'uscita corrispondente. Questa soluzione è migliore della precedente perché è più semplice ed è più difficile che si possano generare errori. Mediante l'organigramma corrispondente si potrà apprendere più agevolmente questa soluzione.

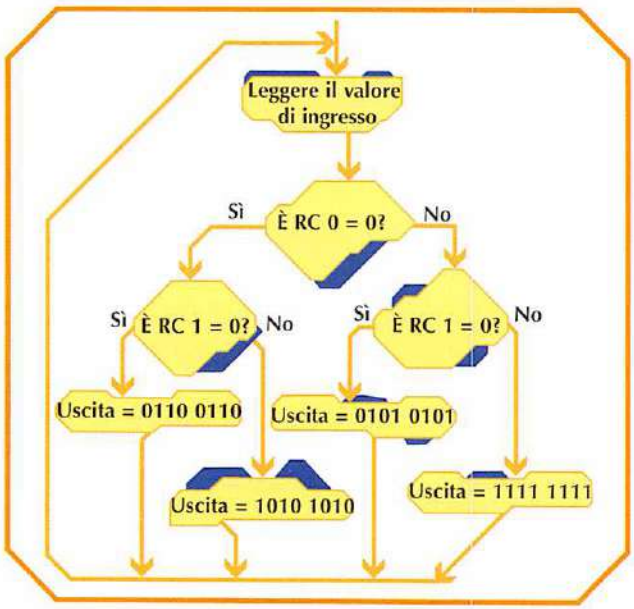
Anche se le due soluzioni proposte sono entrambe valide, lavoreremo con la seconda.

Codice

Nel CD che vi è stato fornito, potrete trovare il file "ese2.asm" che risolve l'esercizio, però come per l'esercizio "ese1.asm", ha una peculiarità. Il codice è stato preparato per essere caricato su un microcontroller precedentemente predisposto con il programma residente Bootloader. Se voi compilate questo programma e lo scrivete sul PIC, non funzionerà correttamente.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORT C	u	u	u	u	u	u	X	X
AND	0	0	0	0	0	0	1	1
Risultato	0	0	0	0	0	0	X	X

Operazione AND per la soluzione 1.



Organigramma della soluzione 2.

ese2 - Blocco note

File Modifica Formato Visualizza ?

```
;Programma combinazionale
;Secondo lo stato degli interruttori RC0 e RC1, attivare i led RB0-RB7 collegati alla
;porta B, in modo conforme alla seguente tabella della verità:
;
; RC1   RC0  |  RB7  RB6  RB5  RB4  RB3  RB2  RB1  RB0
; ---   ---  |  ---  ---  ---  ---  ---  ---  ---  ---
;  0     0   |   1    0    1    0    1    0    1    0
;  0     1   |   0    1    0    1    0    1    0    1
;  1     0   |   1    1    1    1    0    0    0    0
;  1     1   |   1    1    1    1    1    1    1    1

            List    p=16F870        ;Processore
            Include "P16F870.INC"   ;Definizione dei registri interni

            ORG     0x00
            goto    Inizio
            ORG     0x05

Inizio      clrf    PORTB           ;Cancella i valori casuali
            bsf     STATUS,RP0      ;Seleziona il banco 1
            clrf    TRISB           ;Porta B si configura come uscita
            movlw   0xFF
            movwf   TRISC           ;Porta C si configura come ingresso
            bcf     STATUS,RP0      ;Seleziona il banco 0

Loop:       clrwdt                  ;Aggiorna il WDT
            btfsc   PORTC,0         ;Verifica lo stato di RC0
            goto    RC0_vale_1      ;Vale "1"
            btfsc   PORTC,1         ;Vale "0". Verifica lo stato di RC1
            goto    Sono_a_10
            movlw   b'10101010'
            movwf   PORTB           ;Sono a 00. Uscita dalla sequenza
            goto    Loop

Sono_a_10   movlw   b'11110000'
            movwf   PORTB           ;Sono a 10. Uscita dalla sequenza
            goto    Loop

RC0_vale_1  btfsc   PORTC,1         ;Verifica lo stato di RC1
            goto    Sono_a_11       ;Vale "1"
            movlw   b'00001111'
            movwf   PORTB           ;Sono a 01. Uscita dalla sequenza
            goto    Loop

Sono_a_11   movlw   b'11111111'
            movwf   PORTB           ;Sono a 11. Uscita dalla sequenza
            goto    Loop

            end                     ;Fine del programma
```

Codice che risolve l'esercizio.

Per questo è meglio se, con l'aiuto del repertorio delle istruzioni, provate a eseguire questo programma dall'inizio. Dobbiamo iniziare definendo il processore e il file che conterrà l'identificazione dei registri e in seguito definire le posizioni di memoria dove verrà scritto il programma mediante le direttive ORG.

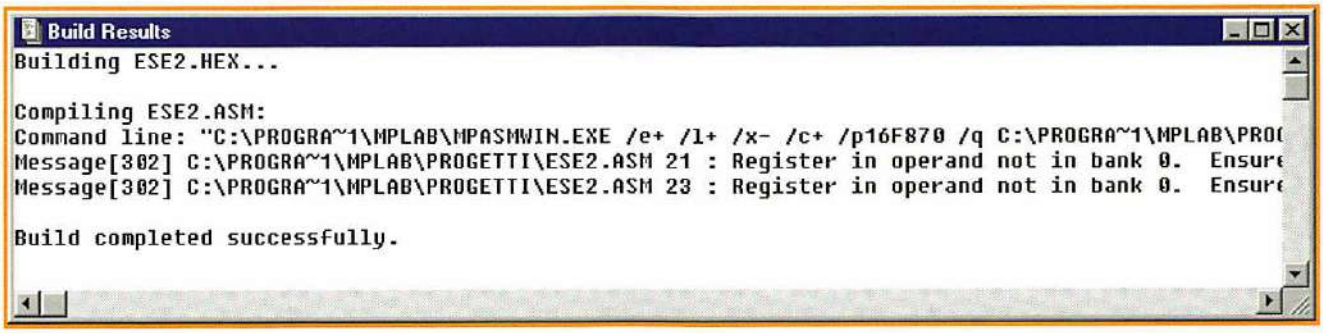
Fatto questo, che è un passaggio comune alla grande maggioranza dei programmi, dobbiamo configurare i dispositivi del PIC, per farlo lavorare secondo le nostre esigenze.

Ricordate che vogliamo che la porta C sia di ingresso e la porta B di uscita.

A questo punto possiamo iniziare a seguire l'organigramma che ci aiuterà a risolvere l'applicazione. Leggeremo uno degli ingressi, ad esempio RC0, e in base al suo valore, mediante un salto condizionale, faremo una o l'altra azione. In qualsiasi delle due opzioni dovremo leggere lo stato dell'altro bit di ingresso (RC1), però letto questo avremo già identificato il valore dell'ingresso, quindi potremo assegnare il corrispondente valore all'uscita. Alla fine delle quattro possibili combinazioni, ci dovremo indirizzare nuovamente, utilizzando un salto incondizionato, alla posizione di inizio, in cui viene letto il primo bit di ingresso, per tornare a ripetere l'operazione.

Compilazione

Per poter compilare il programma bisogna copiarlo nella directory di lavoro utilizzata per MPLAB. Apriamo questo programma e creiamo un nuovo progetto che chiameremo "ese2.pjt". Fatto questo, selezionando Edit Project potremo associare il file in assembler che avremo precedentemente copiato nel percorso specificato. Apriremo quindi il file in assembler ed eseguiremo l'assemblaggio e la compilazione mediante Build All. La compilazione verrà realizzata con successo, come mostrato nell'immagine successiva in cui troviamo due messaggi che ci chiedono di verificare se i banchi di memoria sono stati definiti correttamente, e nel nostro caso sono corretti. In realtà si tratta di avvisi e non di errori. Utilizzate il grafico di distribuzione dei registri nella memoria per verificare in quale banco si



La compilazione è stata eseguita correttamente.

trovino i registri in questione e accertatevi che il programma scritto abbia qualche errore che faccia riferimento a questi registri (TRISB e TRISC).

Ora disponiamo del nostro file "ese2.hex" in codice macchina quindi lo potremo scrivere sul PIC, tuttavia, come già abbiamo visto negli esercizi precedenti, prima di passare alla scrittura bisogna simulare il programma.

Simulazione

La simulazione di questo programma è simile a quella dell'esercizio 1. Dobbiamo aprire la finestra dei registri speciali e occorre aprire anche una finestra specifica per i registri di ingresso e di uscita (PORTC e PORTB), e mediante il simulatore di stimoli asincroni potremo simulare gli ingressi.

Utilizziamo questo esercizio per introdurre una nuova finestra di simulazione, la finestra Stopwatch che serve per controllare il tempo di esecuzione del programma.

Iniziando la simulazione (non dimenticate di posizionarvi nella finestra che contiene il codice) potremo verificare che se non eccitiamo alcun ingresso, ovvero con RC0=RC1=0, l'uscita assume il valore AAh (10101010), così come volevamo. Attivando gli stimoli sui terminali di ingresso verificheremo come con ogni combinazione di ingresso il programma vada all'etichetta corrispondente e fornisca sull'uscita il valore desiderato.

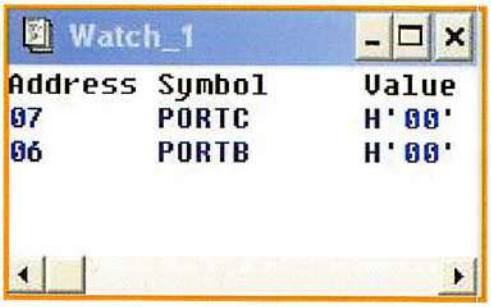
Conclusioni

Abbiamo potuto osservare come i passaggi per realizzare un progetto siano sempre gli stessi e come, una volta acquisita pratica e confidenza, risultino molto semplici. Potete

Special Function Register Window

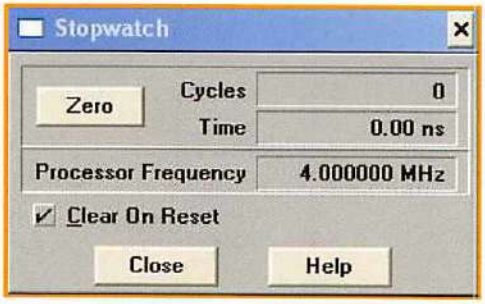
SFR Name	Hex	Dec	Binary	Char
w	00	0	00000000	.
tmr0	00	0	00000000	.
option_reg	FF	255	11111111	.
pcl	00	0	00000000	.
pclath	00	0	00000000	.
status	18	24	00011000	.
fsr	00	0	00000000	.
porta	00	0	00000000	.
trisa	3F	63	00111111	?
portb	00	0	00000000	.
trisb	FF	255	11111111	.
portc	00	0	00000000	.
trisc	FF	255	11111111	.
intcon	00	0	00000000	.
pir1	00	0	00000000	.
pie1	00	0	00000000	.
pir2	00	0	00000000	.
pie2	00	0	00000000	.
tmr1l	00	0	00000000	.
tmr1h	00	0	00000000	.
t1con	00	0	00000000	.
tmr2	00	0	00000000	.
pr2	FF	255	11111111	.
t2con	00	0	00000000	.
ccpr1l	00	0	00000000	.
ccpr1h	00	0	00000000	.
ccp1con	00	0	00000000	.
rcsta	00	0	00000000	.
txreg	00	0	00000000	.
rcreg	00	0	00000000	.
txsta	02	2	00000010	.
spbrg	00	0	00000000	.
adresh	00	0	00000000	.
adresl	00	0	00000000	.
adcon0	00	0	00000000	.
adcon1	00	0	00000000	.
pcon	00	0	00000000	.
eedata	00	0	00000000	.

Finestra dei registri delle funzioni speciali.



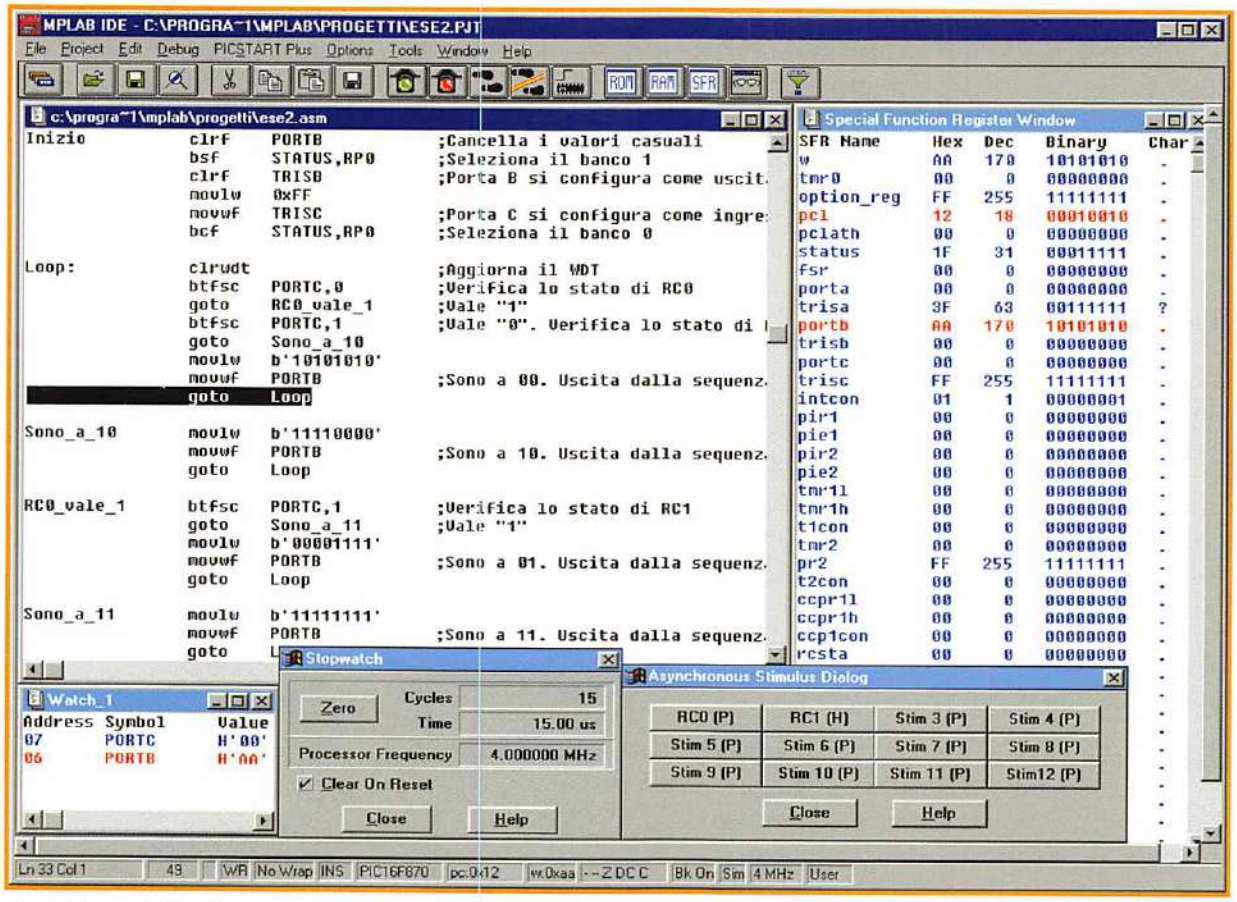
Finestra con i due registri più importanti nella simulazione.

Simulatore di stimoli asincroni configurato con gli ingressi.



Finestra per contare il tempo di esecuzione.

cercare di sviluppare l'altra soluzione prevista e simulare mediante MPLAB il suo funzionamento. In questa fase di sviluppo del programma e simulazione, potete divertirvi a provare a modificare il programma inserendo nuove istruzioni, ampliando il codice con nuove applicazioni e compilandolo. Fate pratica con le istruzioni e acquisite abilità, dato che a poco a poco, gli esercizi si complicheranno e sarà necessaria sicurezza e chiarezza sui concetti di programmazione, così come per il repertorio delle istruzioni.



Aspetto di MPLAB.

Esercizio 4: temporizzatori, il programma

Continuiamo facendo pratica con gli esercizi e in questo caso utilizzando i temporizzatori per risolvere l'accensione sequenziale dei diodi LED. Realizziamo la programmazione e il montaggio come negli esercizi precedenti.

Enunciato

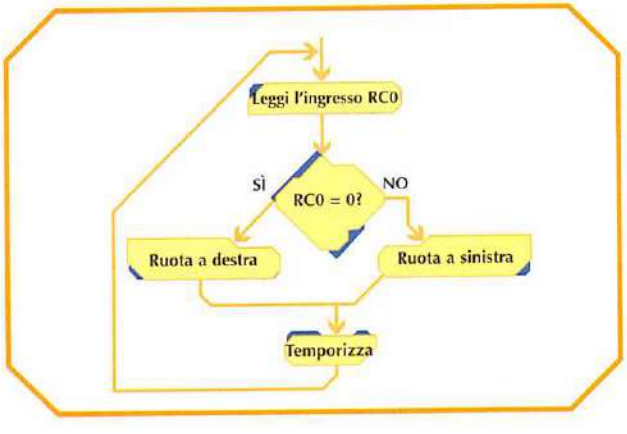
Si vuole realizzare una rotazione sequenziale nell'accensione di ogni LED collegato alla porta B. In base allo stato di un interruttore collegato al pin RC0 della porta C, la rotazione sarà da destra a sinistra (RC0 = 0) o in senso inverso (RC0 = 1). Ogni LED rimarrà acceso 250 ms.

Organigramma

Anche se in questo esercizio lavoriamo con i temporizzatori l'idea generale per risolverlo è molto semplice. Leggeremo l'ingresso e in funzione del valore di quest'ultimo sceglieremo una delle due possibili opzioni, ruotare a sinistra o ruotare a destra. In seguito eseguiremo la chiamata alla routine di temporizzazione e leggeremo nuovamente l'ingresso.

Per la routine di temporizzazione il programmatore normalmente utilizzerà la stessa routine creata per i progetti precedenti, nel nostro caso però supponiamo di non averla ancora sviluppata, quindi ne creeremo una.

Nell'organigramma della figura, possiamo vedere la semplicità del progetto. Ogni volta che chiameremo la routine di temporizzazione caricheremo il valore del contatore, e inizializzeremo il temporizzatore. Terminata la temporizzazione decrementeremo il contatore e se non è arrivato a zero torneremo a inizializzare il temporizzatore ripetendo l'operazione. Quando il contatore arriva a zero, dichiareremo terminata la routine.



Organigramma generale dell'applicazione.

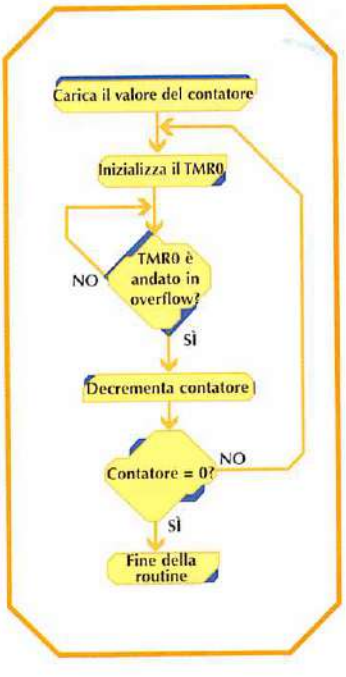
Codice

È molto importane cercare di risolvere gli esercizi da soli senza ricorrere alla soluzione che si trova sul CD. Inoltre come già succedeva per gli esercizi precedenti, la soluzione che si trova sul CD è pensata per essere scritta su un microcontroller che è stato precedentemente caricato con un programma residente.

La prima cosa che dobbiamo fare quando apriamo il Blocco note è scrivere mediante dei commenti ciò che vogliamo far eseguire al programma.

Dobbiamo utilizzare i commenti sia per la definizione generale di ciò che fa il programma che per definire la funzione di ogni istruzione utilizzata. Definiamo il tipo di processore e la libreria che contiene i registri, fatto questo dichiareremo le zone di memoria che dovranno contenere il programma con le direttive ORG.

Dobbiamo configurare i dispositivi, tenendo presente che ora ol-



Organigramma della routine di temporizzazione.

```
Inizio      clrf    PORTB           ;Cancella i valori casuali
            bsf     STATUS,RP0      ;Seleziona il banco 1
            clrf    TRISB           ;Porta B si configura come uscita
            movlw   0xFF
            movwf   TRISC           ;Porta C si configura come ingresso
            movlw   b'00000110'
            movwf   OPTION_REG      ;Prescaler da 128 per il TMR0
            bcf     STATUS,RP0      ;Seleziona il banco 0
```

Codice per configurare i dispositivi.

```
            bsf     STATUS,C        ;Attiva il carry
Loop        call    Delay           ;Temporizzazione da 250 ms
            btfsc   PORTC,0         ;RC0 = 0 ?
            goto    A_Destra        ;No, rotazione a destra
A_Sinistra  rlf     PORTB,F         ;Si, rotazione a sinistra
            goto    Loop
A_Destra    rrf     PORTB,F         ;Rotazione a destra
            goto    Loop
```

Codice che risolve l'enunciato.

```
Delay       movlw   .10
            movwf   Contatore       ;Carica il contatore con 10
Delay_0     bcf     INTCON,T0IF     ;Azzera il flag di overflow del TMR0
            movlw   0x3c
            movwf   TMR0            ;Carica il TMR0
Delay_1     clrwdt                  ;Aggiorna il WDT
            btfss   INTCON,T0IF     ;Overflow del TMR0 ??
            goto    Delay_1         ;Non sono ancora passati i 25 ms
            decfsz  Contatore,F     ;Decrementa il contatore. E' stato ripetuto 10 volte ?
            goto    Delay_0         ;Ancora no, temporizza 25 ms
            return                  ;Ora si
```

Routine di temporizzazione.

Ese4 - Blocco note

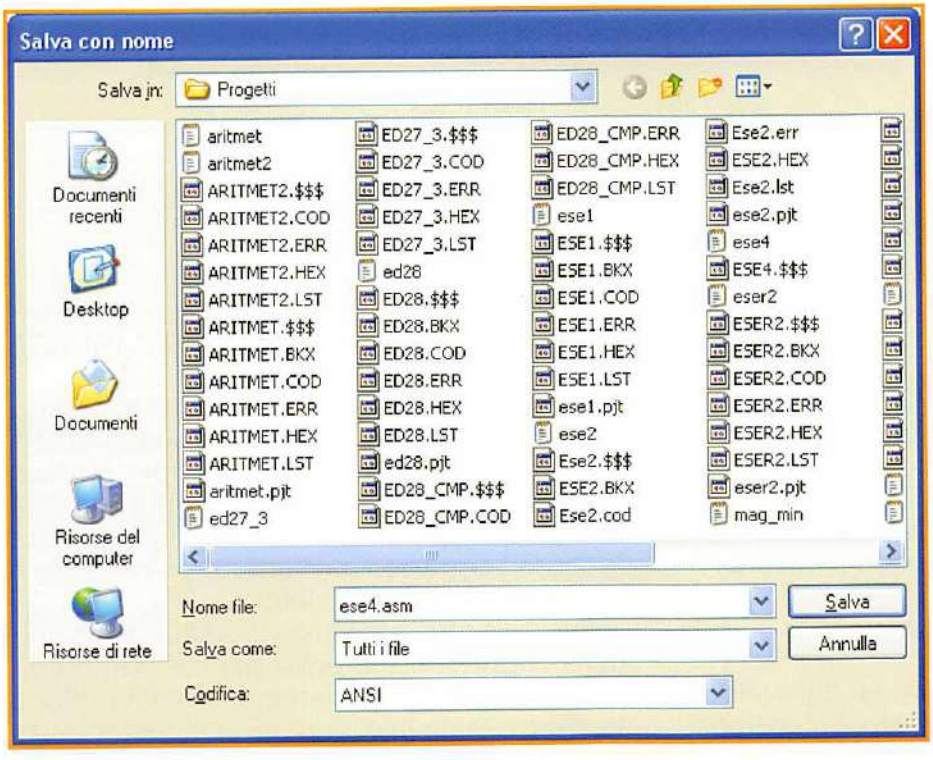
File Modifica Formato Visualizza ?

```
;Realizziamo una rotazione sequenziale nell'accensione di ogni LED collegato alla
;porta B. Se RC0 = 0, la rotazione sarà da destra verso sinistra e viceversa.
;Ogni LED rimane acceso 0.25 secondi (250 mS)

            List    p=16F870        ;Processore
            include "P16F870.INC"   ;Definizione dei registri interni

Contatore   equ     0x20            ;Variabile per la temporizzazione

            org     0x00
            goto    Inizio
            org     0x05

Inizio      clrf    PORTB           ;Cancella i valori casuali
            bsf     STATUS,RP0      ;Seleziona il banco 1
            clrf    TRISB           ;Porta B si configura come uscita
            movlw   0xFF
            movwf   TRISC           ;Porta C si configura come ingresso
            movlw   b'00000110'
            movwf   OPTION_REG      ;Prescaler da 128 per il TMR0
            bcf     STATUS,RP0      ;Seleziona il banco 0

            bsf     STATUS,C        ;Attiva il carry
Loop        call    Delay           ;Temporizzazione da 250 ms
            btfsc   PORTC,0         ;RC0 = 0?
            goto    A_Destra        ;No, rotazione a destra
A_Sinistra  rlf     PORTB,F         ;Si, rotazione a sinistra
            goto    Loop
A_Destra    rrf     PORTB,F         ;Rotazione a destra
            goto    Loop

;*****************************************************************************
;Delay è una routine che realizza una temporizzazione da 250 mS, che è il tempo che deve
;se il PIC lavora ad una frequenza di 4 MHz, il TMR0 conta ogni µs. Se si desidera tempo-
;rizzare 25000 µs (25 mS) con un prescaler da 128, il TMR0 dovrà contare 195 eventi
;(195 * 128 = 24960). Il valore 195 equivale a 0xc3 e, dato che il TMR0 è ascendente, lo dovremo
;caricare con il suo complemento a 2 (0x3c). Questa temporizzazione dovrà essere ripetuta 10 volte
;per ottenere il totale desiderato (250000)

Delay       movlw   .10
            movwf   Contatore       ;Carica il contatore con 10
Delay_0     bcf     INTCON,T0IF     ;Azzera il flag di overflow del TMR0
            movlw   0x3c
            movwf   TMR0            ;Carica il TMR0
Delay_1     clrwdt                  ;Aggiorna il WDT
            btfss   INTCON,T0IF     ;Overflow del TMR0?
            goto    Delay_1         ;Non sono ancora passati i 25 mS
            decfsz  Contatore,F     ;Decrementa il contatore. E' stato ripetuto 10 volte?
            goto    Delay_0         ;Ancora no, temporizza 25 mS
            return                  ;Ora si

            end                     ;Fine del programma
```

Codice finale dell'applicazione.

tre agli ingressi e alle uscite, utilizzeremo un temporizzatore. Definiamo la porta B come uscita e la porta C come ingresso e sul registro delle opzioni OPTION_REG inseriremo un predivisore di valore 128.

Il programma lo risolveremo utilizzando le istruzioni di rotazione, quindi prima dovremo attivare il Carry, bit che si trova sul registro STATUS. A questo punto inizia l'anello in cui si chiama la routine di temporizzazione, si legge l'ingresso e mediante un salto condizionale discriminiamo i due possibili stati. Ruotiamo in un senso o nell'altro in funzione del valore dell'ingresso e mediante un salto incondizionale ripetiamo il giro. Nell'immagine della figura potete vedere la semplicità con cui abbiamo risolto l'applicazione.



Salva il file nella directory di lavoro di MPLAB.

Routine di temporizzazione

Per realizzare la routine di temporizzazione dobbiamo fare prima dei calcoli. Se il PIC lavora a una frequenza di 4 MHz, il TMR0 evolve ogni microsecondo. Se si desidera temporizzare 25.000 µs, cioè 25 ms, con un predivisore di 128, il TMR0 dovrà contare 195 eventi (195x128 = 24.960). Il valore 195 inoltre equivale a 0xc3, dato che il TMR0 è ascendente, sarà necessario caricare il suo complemento (0x3c). Questa temporizzazione dovrà essere ripetuta 10 volte per ottenere il totale desiderato (250.000 µs o 250 ms).

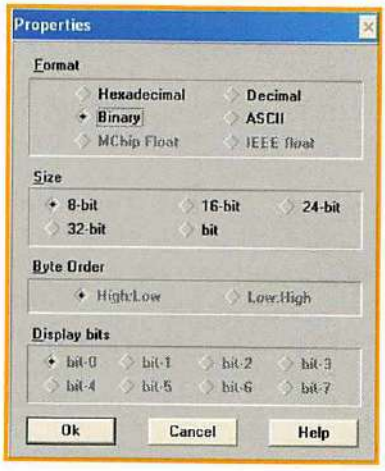
Definiremo una costante all'inizio del programma che utilizzeremo come contatore e inizializzeremo la routine caricando su di essa il valore del conteggio, che nel nostro caso è 10. Inizieremo il ciclo di temporizzazione cancellando il flag di overflow del Timer, dato che ogni volta che si ripete il ciclo è perché il timer ha terminato il suo conteggio e ha attivato questo flag. Caricheremo il temporizzatore con il valore del conteggio e attenderemo in un ciclo che il temporizzatore termini di contare. Quando questo avviene, esso attiva un flag per avvisarci dell'evento, a questo punto decrementeremo il contatore. Nel caso in cui il contatore raggiunga il valore zero, termineremo la routine, in caso contrario la ripeteremo. Nella terza figura della pagina precedente possiamo vedere il codice relativo a questa routine.

Compilazione

Completato il codice lo dovremo compilare per verificare che non abbia errori. Ricordate che si deve salvare il codice con estensione ".asm" e selezionando nel campo Tipo l'opzione "Tutti i file", all'interno della cartella di lavoro che utilizziamo in MPLAB.

Passeremo alla compilazione del file, creando un progetto, includendo in esso il nostro fi-

Risultato della compilazione.



Visualizzazione dei registri in binario.

```
Delay           movlw   .10
                movwf   Contatore       ;Carica il contatore con 10
;Delay_0                 bcf     INTCON,T0IF     ;Azzera il flag di overflow del TMR0
;               movlw   0x3c
;               movwf   TMR0            ;Carica il TMR0
;Delay_1                 clrwdt                  ;Aggiorna il WDT
;               btfss   INTCON,T0IF     ;Overflow del TMR0?
;               goto    Delay_1         ;Non sono ancora passati i 25 mS
;               decfsz  Contatore,F     ;Decrementa il contatore. E' stato ripetuto 10 volte?
;               goto    Delay_0         ;Ancora no, temporizza 25 mS
                return                  ;Ora sì

                end                     ;Fine del programma
```

Modifica della routine di temporizzazione trasformando le linee di codice in commenti.

le, e selezionando l'opzione Build All. Nella figura in basso possiamo verificare il risultato ottenuto dalla compilazione.

Simulazione

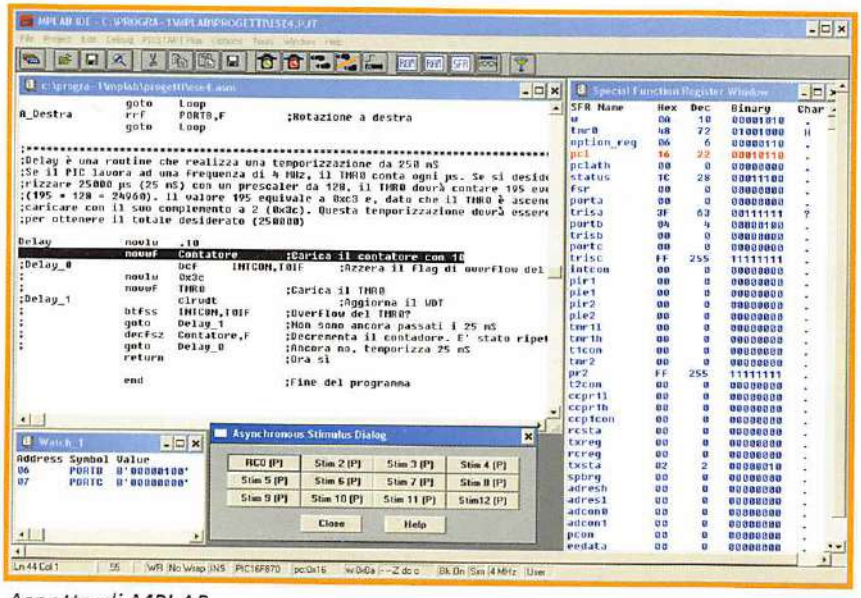
Per simulare questo programma dobbiamo aprire la finestra dei registri delle funzioni speciali e crearne una in cui potremo vedere solamente i registri delle porte. In questa finestra, per rendere più semplice la visualizzazione, definiremo la rappresentazione dei registri in modo binario. Cliccando con il pulsante sinistro del mouse l'angolo superiore sinistro della finestra, si aprirà un menù in cui selezioneremo Edit Watch. Fatto questo apparirà una nuova finestra in cui per ogni registro selezioneremo Properties, cambiando in binario la rappresentazione. Avremo bisogno anche del simulatore di stimoli asincroni per verificare che, cambiando lo stato degli ingressi, l'accensione sequenziale dei LED venga eseguita in senso contrario. Configurate unicamente il pin RC0 su questo simulatore.

Per evitare di rimanere bloccati all'interno della routine di temporizzazione durante la simulazione, modificheremo il programma e trasformeremo in commenti la maggior parte della routine di temporizzazione, evitando che vengano eseguite le istruzioni che potrebbero provocare il blocco. Nell'immagine della figura in alto possiamo vedere i cambiamenti eseguiti. Salvate il programma e compilate nuovamente.

Se simuliamo il codice in modo passo a passo mediante F7, potremo verificare che sul registro di uscita il bit attivato esegue la rotazione. Verificheremo che il funzionamento della rotazione sia quello desiderato.

Prima di dichiarare terminata la simulazione dobbiamo riportare allo stato originale la routine di temporizzazione eliminando i ";" inseriti precedentemente, e compilare nuovamente il programma per ottenere il file desiderato in codice macchina.

Il montaggio hardware di questo progetto corrisponde al montaggio degli esercizi precedenti. Provate a eseguire la scrittura del programma sul PIC e il montaggio hardware senza ricorrere ai suddetti esercizi precedenti.



Aspetto di MPLAB.

Esercizio 5: display, il programma

Affrontiamo in questo caso un nuovo esercizio in cui vogliamo lavorare con il display. Prima di continuare a studiare i dispositivi del microcontroller, è necessario fare pratica e affinare le conoscenze acquisite fino a questo momento. Le difficoltà andranno crescendo man mano che vedremo i nuovi dispositivi ed è per questa ragione che dobbiamo acquisire scioltezza nel risolvere i progetti.

Enunciato

Vogliamo visualizzare lo stato di un interruttore sul display a 7 segmenti del laboratorio. Quindi se l'interruttore è a 5 Vdc (positivo) sul display si visualizzerà "1" e se è a 0 Vdc (negativo) sul display visualizzeremo uno "0". Potremo inoltre accendere il punto decimale del display mediante un altro interruttore.

Organigramma

L'organigramma generale dell'applicazione è molto semplice, perché dispone di una sola condizione per ogni ingresso. In questo modo, leggendo il primo ingresso, lo stato che vogliamo rappresentare, in funzione del valore letto, selezionerà una o l'altra uscita. Fatto questo, leggeremo l'ingresso successivo, per accendere nel caso sia necessario, il punto decimale.

Nella figura potete vedere l'organigramma che riassume quanto spiegato.

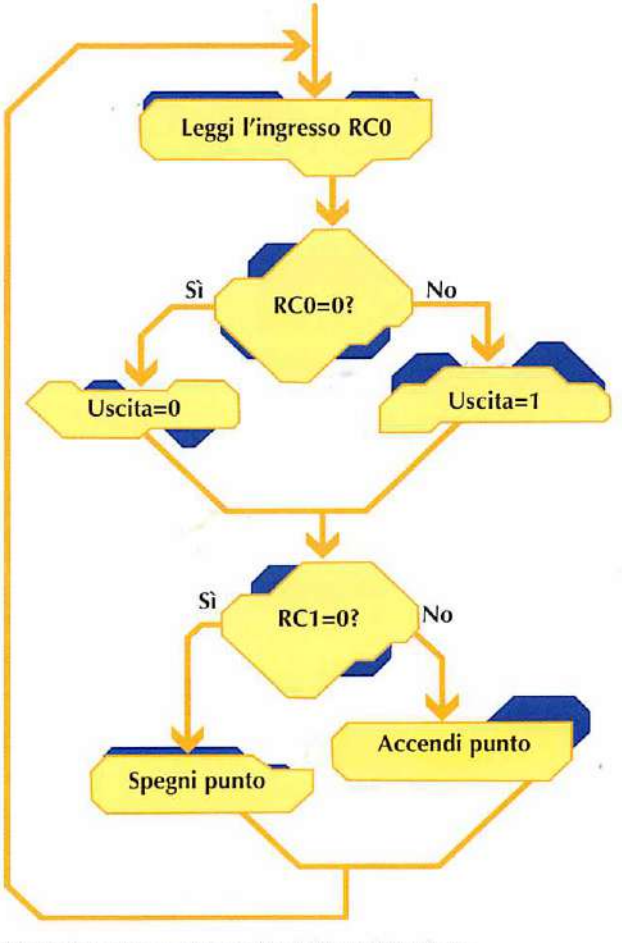
Codice

Dato che abbiamo bisogno di due ingressi, utilizzeremo come fatto sinora, i terminali della porta C, RC0 e RC1 e le uscite le manderemo sulla porta B. In questo modo abbiamo definito i dispositivi del microcontroller che vogliamo utilizzare. Se iniziamo a creare il nostro codice definiremo il processore e la libreria dei dispositivi dopodiché continueremo con la definizione della porta C come ingresso e della porta B come uscita. Se a tutto questo aggiungiamo i commenti pertinenti, otterremo il codice mostrato nella figura in basso.

A partire da questo punto dobbiamo definire il codice che risolve l'enunciato. Dato che abbiamo solamente due stati di ingresso, possiamo assegnarli a una determinata uscita. Nel caso di ingressi con più stati (più ingressi) potremmo fare una subroutine in cui includere una tabella di conversione ingresso/uscita, ma non essendo il caso, assegneremo diretta-

```
;Il Display a 7 segmenti
;
;sul display a catodo comune collegato alla porta B, visualizziamo lo stato
;logico "0" o "1" dell'interruttore RC0. Mediante l'interruttore RC1 si attiva o mero il
;punto decimale.

                List    p=16F870        ;Tipo de processore
                include "P16F870.INC"   ;Definizione dei registri interni

                org     0x00
                goto    inizio
                org     0x05

inizio          clrf    PORTB           ;Cancella i valori casuali
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   0xFF
                movwf   TRISC           ;Porta C si configura come ingresso
                bcf     STATUS,RP0      ;Seleziona il banco 0
```

Codice risultante dopo aver definito i dispositivi.



Organigramma generale dell'applicazione.

mente un'uscita ad ogni ingresso.

Nella tabella della figura a lato ricordiamo la conversione fra binario e codice a 7 segmenti in modo da assegnare l'uscita corretta.

Leggeremo il primo ingresso (RC0) utilizzando un salto condizionale, quindi se nell'istruzione successiva inseriremo goto o salto incondizionale, avremo contemplato i due possibili stati. A partire dall'indirizzo indicato da goto, risolveremo uno degli stati, però se non si esegue il goto, nell'indirizzo successivo a esso, risolveremo l'altro stato.

DIGIT	BINARIO	7 SEGMENTI b'0gfedcba
0	0000	00111111
1	0001	00000110
2	0010	01011011
3	0011	01001111
4	0100	01100110
5	0101	01101101
6	0110	01111101
7	0111	00000111
8	1000	01111111
9	1001	01100111
punto		10000000

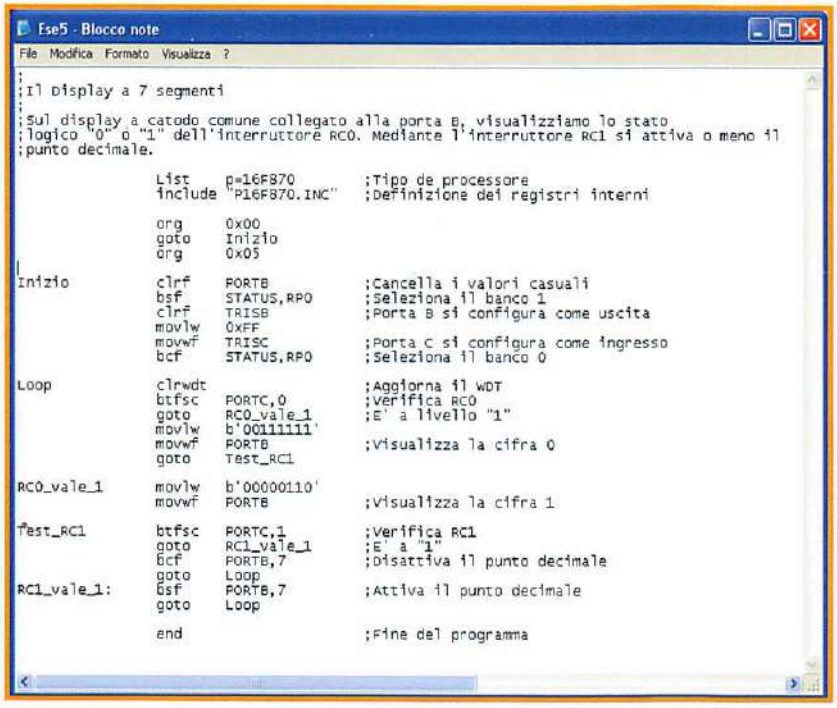
Conversione da binario in codice a 7 segmenti.

Al termine di ognuno di questi percorsi dovremo saltare all'altra parte del codice, dove elaboreremo l'altro ingresso, che verrà trattato nello stesso modo del primo. Fatto questo dovremo tornare all'inizio, e leggere nuovamente il primo ingresso, per costruire un programma ciclico.

Il codice finale che presentiamo è quello mostrato nella figura in basso, vi consigliamo però di non copiarlo subito, ma di risolvere da soli questo esercizio.

Sul CD potrete trovare questo stesso programma, preparato però per lavorare con il Bootloader.

Salvate il programma con estensione ".asm" nella directory di lavoro di MPLAB, per procedere alla sua compilazione e all'assemblaggio.



Ese5 - Blocco note

File Modifica Formato Visualizza ?

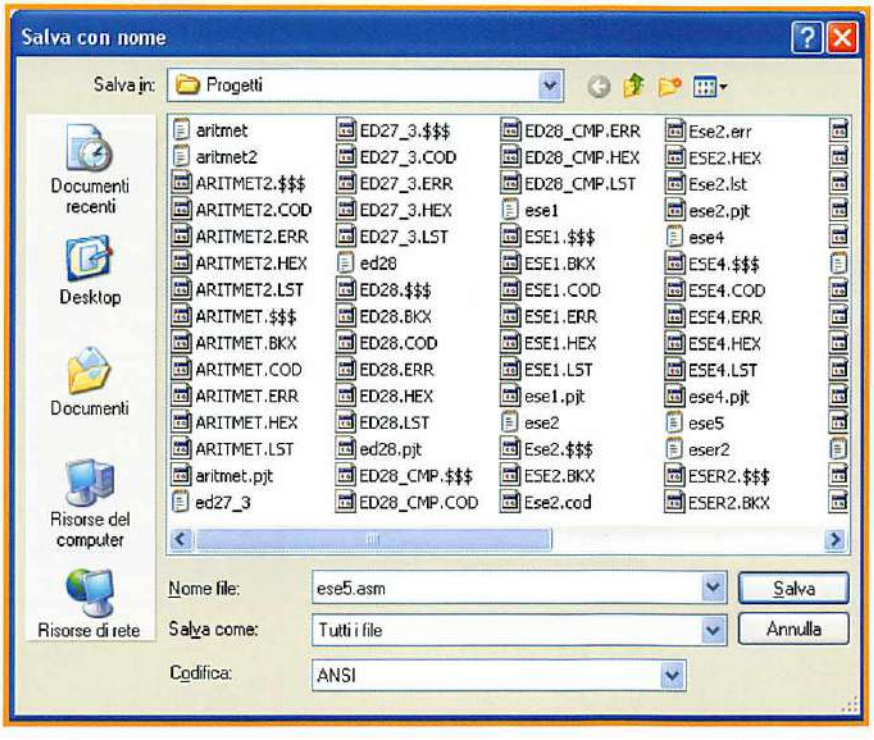
```
;
;Il Display a 7 segmenti
;
;Sul display a catodo comune collegato alla porta B, visualizziamo lo stato
;logico "0" o "1" dell'interruttore RC0. Mediante l'interruttore RC1 si attiva o meno il
;punto decimale.

                List    p=16F870        ;Tipo de processore
                include "P16F870.INC"   ;Definizione dei registri interni

                org     0x00
                goto    Inizio
                org     0x05

Inizio          clrf    PORTB           ;Cancella i valori casuali
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   0xFF
                movwf   TRISC           ;Porta C si configura come ingresso
                bcf     STATUS,RP0      ;Seleziona il banco 0

Loop            clrwdt                  ;Aggiorna il WDT
                btfsc   PORTC,0         ;Verifica RC0
                goto    RC0_vale_1      ;E' a livello "1"
                movlw   b'00111111'
                movwf   PORTB           ;Visualizza la cifra 0
                goto    Test_RC1

RC0_vale_1      movlw   b'00000110'
                movwf   PORTB           ;Visualizza la cifra 1

Test_RC1        btfsc   PORTC,1         ;Verifica RC1
                goto    RC1_vale_1      ;E' a "1"
                bcf     PORTB,7         ;Disattiva il punto decimale
                goto    Loop
RC1_vale_1:     bsf     PORTB,7         ;Attiva il punto decimale
                goto    Loop

                end                     ;Fine del programma
```

Codice finale del programma.

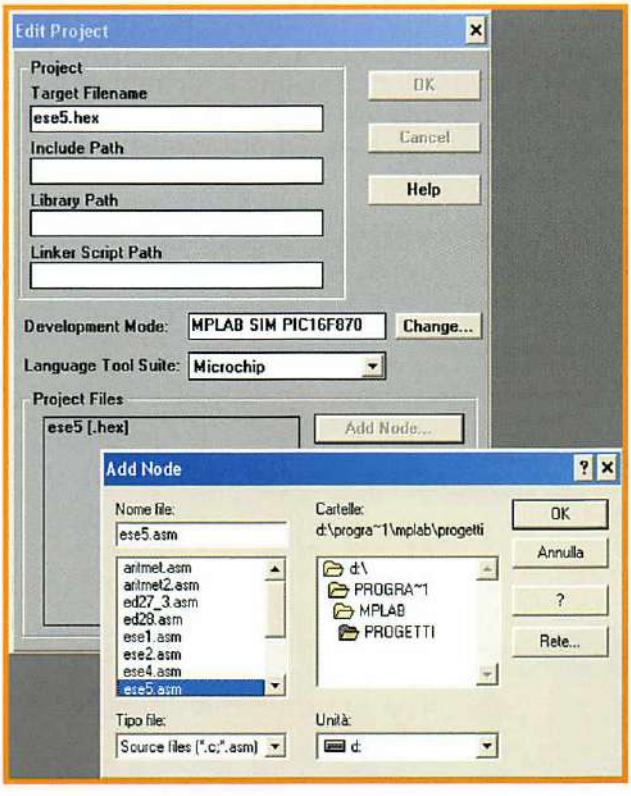
Compilazione

Continuiamo con la stessa dinamica degli esercizi precedenti, con l'unico scopo di abituarci a questo modo lavoro, quindi procederemo alla compilazione. Aprite MPLAB e create un progetto. Ricordate di dare lo stesso nome al progetto e al file assembler, per evitare possibili confusioni. Editate il progetto e mediante Add Node aggiungete il file in assembler precedentemente salvato nella directory di lavoro. Selezionate File → Open e aprite il file che abbiamo creato e aggiungetelo al progetto.

Cliccate Control+F10 o utilizzate la barra degli strumenti o il menù di controllo per assemblare e compilare il codice. Il codice si compila senza errori e con due messaggi di avviso, nel caso avessimo confuso i banchi di memoria



Salvate il programma nella directory di lavoro con l'estensione corretta.



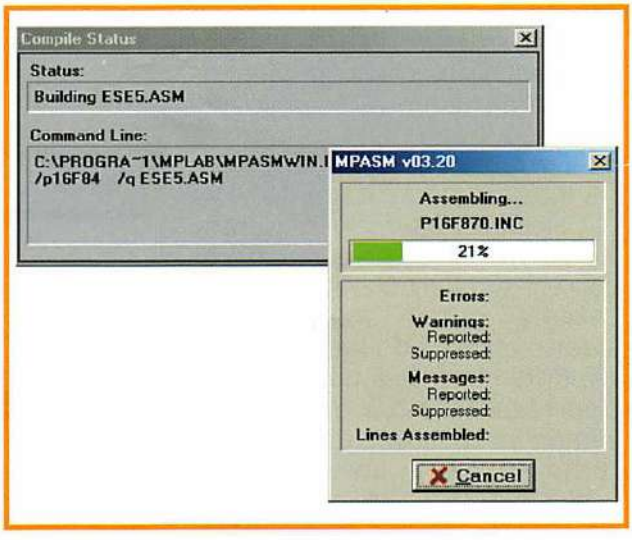
Dobbiamo aggiungere il codice creato al progetto.

riferiti ai registri TRISx. Questi messaggi non sono importanti, ma dobbiamo comunque essere attenti se in qualche altro caso veniamo avvisati da messaggi diversi. Dobbiamo verificare sempre i messaggi forniti dal compilatore.

Simulazione

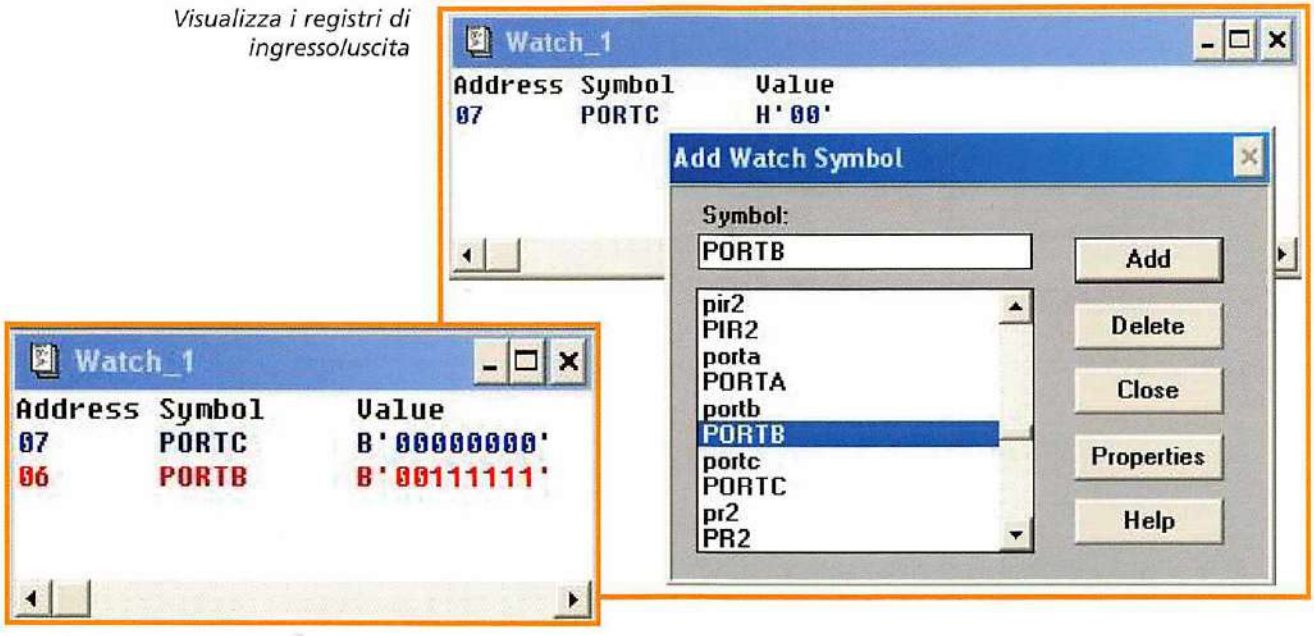
Abbiamo eseguito ciò che per molti è la parte più complicata: la creazione del codice e la sua verifica, però come in tutti i progetti, prima di scrivere dobbiamo assicurarci che il programma esegua esattamente le nostre istruzioni. Visualizziamo le finestre dei registri delle funzioni speciali e quella che contiene i registri di ingresso e di uscita (che dovrà essere stata creata in precedenza). Configurate quest'ultima finestra in modo che la visualizzazione si realizzi in binario. Infine, configurate il simulatore di stimoli asincroni per fare in modo che i primi due segnali siano RC0 e RC1.

Si simula direttamente, inizialmente i bit RC0 e RC1 saranno a zero, quindi sull'uscita troveremo il valore "00111111". Se ora cambiamo lo stato del pin RC1 a High sul simulatore e cliccheremo su di esso potremo osservare come cambia il valore dell'ingresso, e come



Processo di compilazione.

Visualizza i registri di ingressoluscita



Uscita quando RC0 = RC1 = 0.

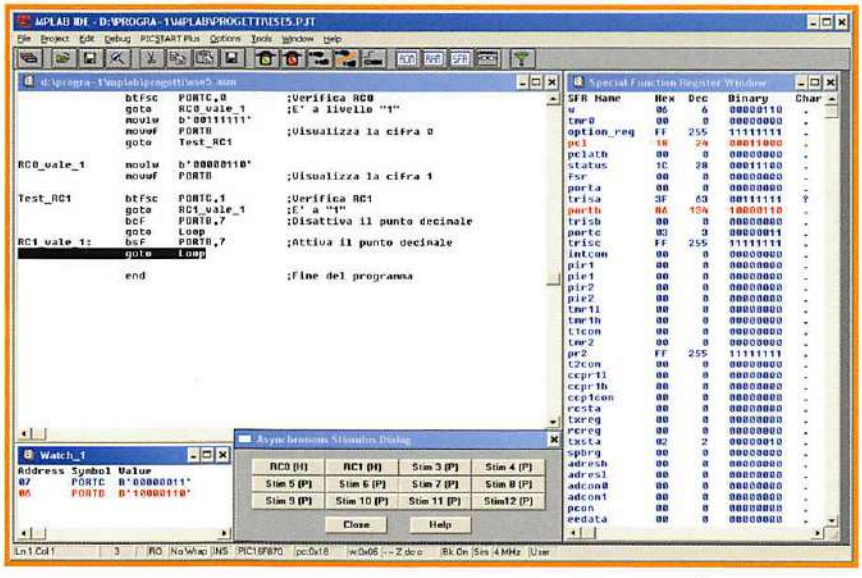
l'uscita prenderà il valore "10111111", ciò significa che il bit 7 di questa uscita, che corrisponde al punto decimale, è stato attivato.

Se ora attiviamo il pin RC0 nello stesso modo, potremo verificare come varia il valore dell'ingresso e come viene eseguita l'altra parte del programma, con l'uscita corrispondente a 1 sul display e il punto decimale attivato.

Sul programma c'è un errore perché quando torna a ripetere il ciclo, legge solamente il pin RC0 e in funzione di quest'ultimo fornisce un'uscita. L'uscita in questo caso azzera sempre il valore di RB7 (punto decimale), quindi anche se l'interruttore RC1 non cambia posizione e continua a richiedere che il punto sia attivato, sino a quando il programma non tornerà a rileggerne il valore, il punto rimarrà spento.

Calcolate il tempo di durata di questo processo e cercate di correggere l'errore.

Nella nostra applicazione questo errore non ha importanza, dato che visivamente non si apprezza questo breve lasso di tempo in cui si spegne il punto decimale, ma in un'altra applicazione questo potrebbe essere un errore critico. Immaginate, per esempio, che questo interruttore controlli il pilota automatico di un aereo, e che ciclicamente venga scollegato il segnale di pilota automatico. Anche se si tratta di un caso estremo, dobbiamo vigilare su questi possibili errori e, come volevamo dimostrarvi, è nella fase di simulazione dove possiamo correggere questi problemi. Dato che per la nostra visualizzazione il funzionamento è corretto, diamo per buona questa fase e prepariamoci a realizzare il montaggio hardware dell'applicazione.



Videata generale di MPLAB con RC0 = RC1 = 1.

Esercizio 6: display in esadecimale, il programma

Terminiamo la prima fase degli esercizi con l'ultimo che troviamo sul CD. Grazie a questo esercizio faremo pratica con l'utilizzo delle subroutines, imparando a muoverci su una tabella. Con questo esercizio e i precedenti riteniamo che abbiate acquisito l'esperienza e la scioltezza necessarie per affrontare altre sfide, quali la gestione di nuovi dispositivi, l'utilizzo di periferiche, il lavoro con un programma residente, la programmazione del PIC in altri linguaggi, ecc.

Enunciato

Si vuole realizzare la conversione di un numero inserito in binario nel suo equivalente in esadecimale. Verrà visualizzato il risultato su uno dei display del laboratorio e si piloteranno gli ingressi mediante interruttori.

Organigramma

In questo esercizio, a differenza del precedente, esistono molte combinazioni di ingresso, dato che disponiamo di quattro interruttori (2^4=16 combinazioni) quindi lo risolveremo attraverso una subroutine e mediante una tabella. Ne consegue che l'organigramma che risolve l'enunciato generale dell'applicazione potrebbe risultare tanto semplice quanto quello mostrato nella figura.

Codice

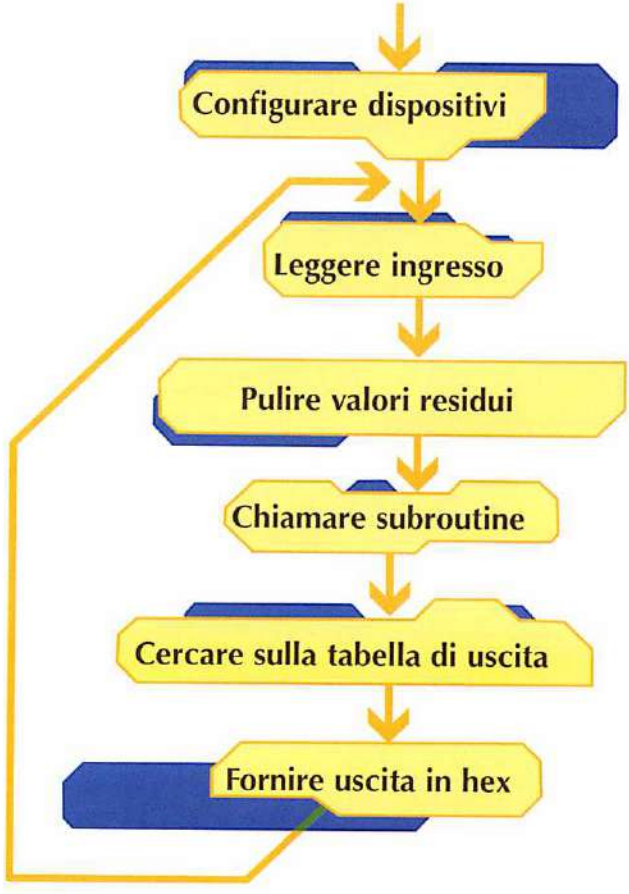
Il processo iniziale per cominciare a scrivere il nostro codice è uguale a quello degli altri programmi. Grazie ai commenti descriveremo la funzionalità finale del programma e il significato delle linee di codice, dovremo definire il processore e la libreria, e dopo aver indicato i dispositivi con cui lavoreremo, configurarli nel PIC. Nel nostro caso lavoreremo con 4 ingressi e con 8 uscite, quindi potremo utilizzare la porta C per gli ingressi e la porta B per le uscite.

Osservando il codice della figura in basso possiamo vedere che fino a questo punto l'unica differenza rispetto ai programmi precedenti è che in questo caso è stata configurata la porta C affinché i quattro bit meno significativi siano ingressi e gli altri quattro uscite.

Per sviluppare il codice che risolve l'applicazione è consigliabile iniziare aggiornando il Watch Dog, infatti, anche se in questo esercizio

```
;Il display a 7 segmenti. Decodificatore da esadecimale a 7 segmenti
;
;Mediante i quattro interruttori RC0-RC3 si imposta un valore binario da 4 bit
;che deve essere visualizzato sul display in esadecimale

                List    p=16F870        ;Processore
                include "P16F870.INC"   ;Definizione dei registri interni

                org     0x00
                goto    Inizio
                org     0x05

Inizio          clrf    PORTB           ;Cancella i valori casuali
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00001111'
                movwf   TRISC           ;Porta C si configura come ingresso
                bcf     STATUS,RP0      ;Seleziona il banco 0
```

Inizieremo il codice con le definizioni e la configurazione.



Organigramma dell'applicazione.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORT C	u	u	u	u	X	X	X	X
AND	0	0	0	0	1	1	1	1
Risultato	0	0	0	0	X	X	X	X

Operazione AND per pulire valori residui.

non ha molto senso, è consigliabile abituarsi a utilizzare questa istruzione all'inizio di una fase ciclica (potrete provare a simulare il programma senza questa istruzione). Leggeremo gli ingressi, quindi, per evitare eventuali valori residui, eseguiremo un'operazione AND per conservare unicamente i valori dei quattro bit con cui lavoreremo.

Eseguiremo una chiamata alla subroutine in cui si trova la tabella e il valore ottenuto lo porteremo all'uscita, ripetendo questa operazione ciclicamente.

Tabella di conversione

Quando eseguiamo una chiamata a subroutine il registro di lavoro contiene il valore dell'ingresso, che sarà compreso tra 0 e 15. Questo valore sarà quello che porterà il puntatore a indicare questa o quell'altra posizione della tabella. A ogni posizione si assegna un valore diverso che verrà caricato sul registro di lavoro, ritornando al programma principale. Sappiamo come convertire un numero binario al suo corrispondente decimale in 7 segmenti, però per convertire questo stesso numero in esadecimale dobbiamo tener conto di 6 nuovi caratteri. Nella tabella della figura della pagina successiva possiamo vedere qual è la conversione da esadecimale a codice a 7 segmenti. Termineremo il programma con l'istruzione end alla fine del codice.

```
;Il display a 7 segmenti. Decodificatore da esadecimale a 7 segmenti
;
;Mediante i quattro interruttori RC0-RC3 si imposta un valore binario da 4 bit
;che deve essere visualizzato sul display in esadecimale

                List    p=16F870        ;Processore
                include "P16F870.INC"   ;Definizione dei registri interni

                org     0x00
                goto    Inizio
                org     0x05

Inizio          clrf    PORTB           ;Cancella i valori casuali
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00001111'
                movwf   TRISC           ;Porta C si configura come ingresso
                bcf     STATUS,RP0      ;Seleziona il banco 0

Loop            clrwdt                  ;Aggiorna il WDT
                movf    PORTC,W
                andlw   b'00001111'     ;Legge il codice di RC0-RC3
                call    Tabella         ;Converte a 7 segmenti
                movwf   PORTB           ;Visualizza sul display
                goto    Loop
```

Codice risultante senza includere la subroutine.

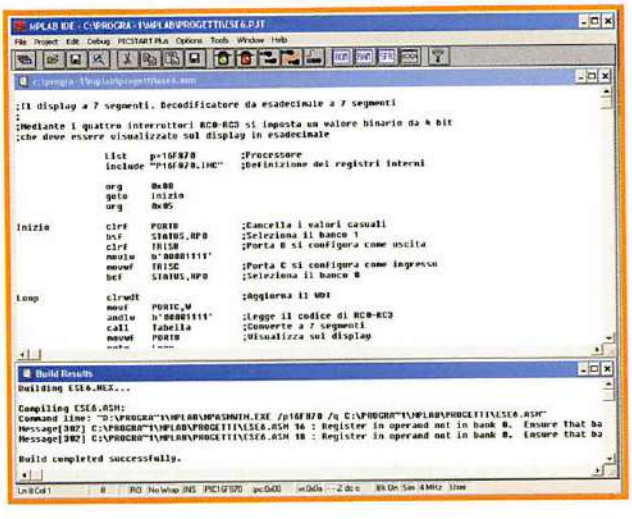
```
;*******************************************************************************
;Tabella: Questa routine converte il codice binario presente sui 4 bit meno significativi
;del reg. W nel suo equivalente a 7 segmenti. Il codice a 7 segmenti viene portato anche
;sul registro W

Tabella:        addwf   PCL,F           ;Spostamento sulla tabella
                retlw   b'00111111'     ;Cifra 0
                retlw   b'00000110'     ;Cifra 1
                retlw   b'01011011'     ;Cifra 2
                retlw   b'01001111'     ;Cifra 3
                retlw   b'01100110'     ;Cifra 4
                retlw   b'01101101'     ;Cifra 5
                retlw   b'01111101'     ;Cifra 6
                retlw   b'00000111'     ;Cifra 7
                retlw   b'01111111'     ;Cifra 8
                retlw   b'01100111'     ;Cifra 9
                retlw   b'01110111'     ;Cifra A
                retlw   b'01111100'     ;Cifra b
                retlw   b'00111001'     ;Cifra C
                retlw   b'01011110'     ;Cifra d
                retlw   b'01111001'     ;Cifra E
                retlw   b'01110001'     ;Cifra F
```

Codice della subroutine con la tabella di conversione.

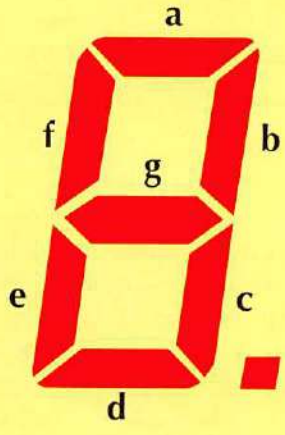
Compilazione

Per compilare dovremo eseguire gli stessi passaggi degli esercizi precedenti. Salveremo il file nella directory di lavoro all'interno di MPLAB con estensione ".asm". Apriremo MPLAB e creeremo un nuovo progetto che chiameremo con lo stesso nome dell'esercizio salvato. Editando il progetto includeremo, all'interno di esso, il file in assembler e successivamente apriremo quest'ultimo per visualizzarlo sul monitor.

Selezioneremo Build All, oppure i tasti Control+F10 o ancora l'icona destinata a questo scopo sulla barra degli strumenti, e il programma si compilerà e si assemblerà correttamente, generando così i file risultanti di questo processo, tra i quali troveremo il file in codice macchina (estensione ".hex").



Videata di MPLAB con il file caricato.



DIGIT	BINARIO	7 SEGMENTI b'0gfedcba'
0	0000	00111111
1	0001	00000110
2	0010	01011011
3	0011	01001111
4	0100	01100110
5	0101	01101101
6	0110	01111101
7	0111	00000111
8	1000	01111111
9	1001	01100111
A	1010	01110111
B	1011	01111100
C	1100	00111001
D	1101	01011110
E	1110	01111001
F	1111	01110001
punto		10000000

Tabella di conversione: Binario/ Esadecimale/ - 7 segmenti.

La compilazione avviene senza errori e con due messaggi già commentati negli esercizi precedenti.

Simulazione

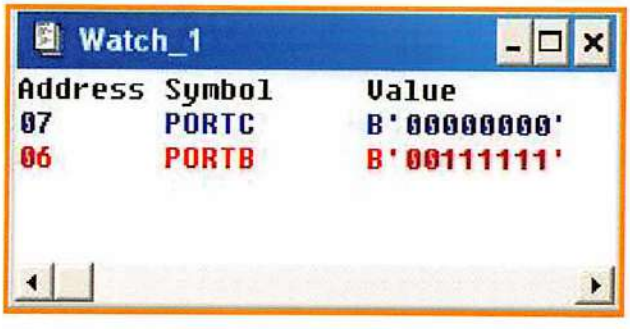
L'ultimo passaggio prima della scrittura del programma sul dispositivo è simulare il funzionamento del programma stesso. Come per gli esercizi precedenti, vi consigliamo di caricare la finestra dei registri di funzioni speciali e quella contenente i registri di ingressi-uscite, che ci servirà per osservare direttamente il risultato. Cambiate la configurazione della rappresentazione di quest'ultima finestra per visualizzare il contenuto dei registri in formato binario e poter verificare in modo più semplice se il risultato è quello desiderato.

Attivate il simulatore di stimoli asincroni e configuratelo in modo che i primi quattro stimoli corrispondano ai quattro pin di ingresso.

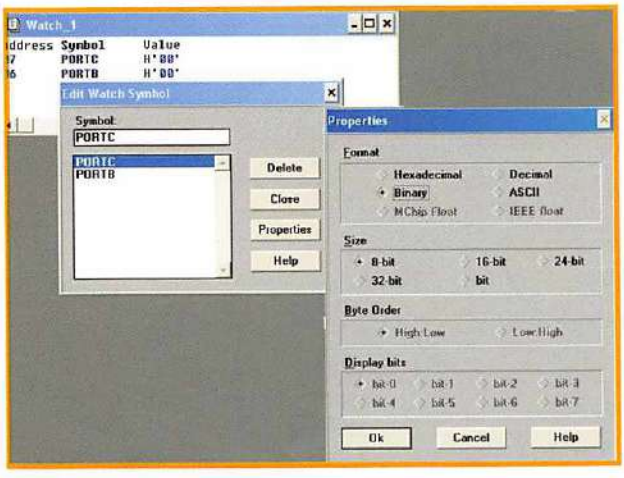
Siete pronti, a questo punto, per eseguire la simulazione, quindi posizionatevi sulla finestra che contiene il codice, premete F7 e iniziate l'esecuzione passo a passo.

Guardando la finestra dei registri speciali verificate come si aggiornano i valori man mano che vengono eseguite le linee di codice. Configuriamo il dispositivo e leggiamo gli ingressi, mettiamo questi valori sul registro di lavoro (dopo aver eseguito la AND di sicurezza) e con questi valori andiamo sulla tabella. Il puntatore punterà il primo indirizzo, dato che il primo valore (senza attivare nessun pin sul simulatore di stimoli) è 0. Sul registro di lavoro verrà portata la conversione a 7 segmenti e quindi passata all'uscita.

Per il momento il funzionamento è quello corretto, però per verificarlo completamente proveremo diversi valori di ingresso e verifiche-



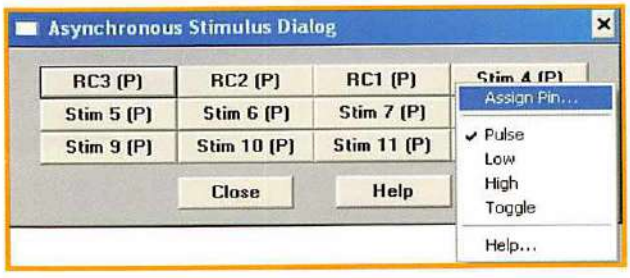
Risultato del primo ciclo con tutti gli ingressi a 0.



Visualizzare la rappresentazione in binario.

remo che il valore ottenuto sull'uscita sia quello desiderato.

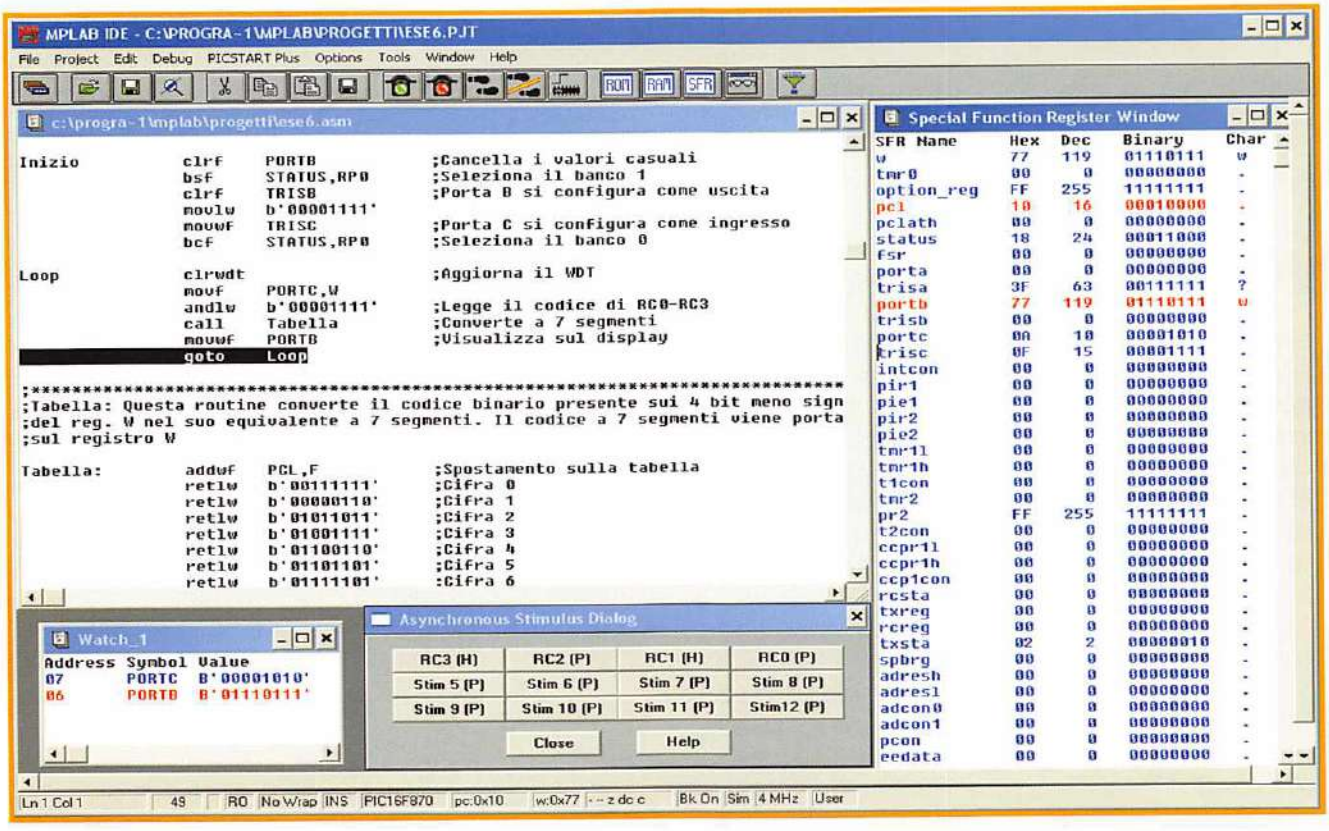
Nella figura possiamo osservare che caricando sull'ingresso il valore "1010" il risultato ottenuto è quello corretto: "01110111". Durante la simulazione potrete vedere che il puntatore va direttamente alla posizione della tabella contenente il dato ricercato.



Simulatore di stimoli asincroni.

Conclusioni

Il montaggio di questo progetto è molto simile a quello dell'esercizio 5, quindi studieremo i due montaggi in modo parallelo, con le piccole differenze che li distinguono. Per il momento consideriamo terminata questa prima fase dell'esercizio, vi consigliamo comunque di affiancare con la pratica ogni avanzamento della teoria esercitandovi con nuovi esempi o modificando quelli già esistenti. Pensate a ciò che vi piacerebbe fare con il PIC e provate a realizzarlo con un vostro progetto.



Vista generale di MPLAB che simula altri valori di ingresso.

Modulo CCP

Riprendiamo lo studio dei dispositivi del microcontroller con i moduli di capture/compare e di modulazione di ampiezza degli impulsi, che chiameremo in modo abbreviato CCP. I microcontroller della famiglia 16F87X normalmente dispongono di due moduli CCP, CCP1 e CCP2, che funzionano nello stesso modo salvo quanto concerne il modo di attivazione speciale. Il nostro PIC dispone solamente di un modulo CCP.

Funzione del modulo CCP

I moduli CCP realizzano tre funzioni:

– Acquisiscono informazioni a 16 bit provenienti dal TMR1.

– Comparano il valore di un registro con quello del TMR1.

– Modulano o controllano l'intervallo di tempo in cui si commuta da 1 a 0 un pin del microcontroller.

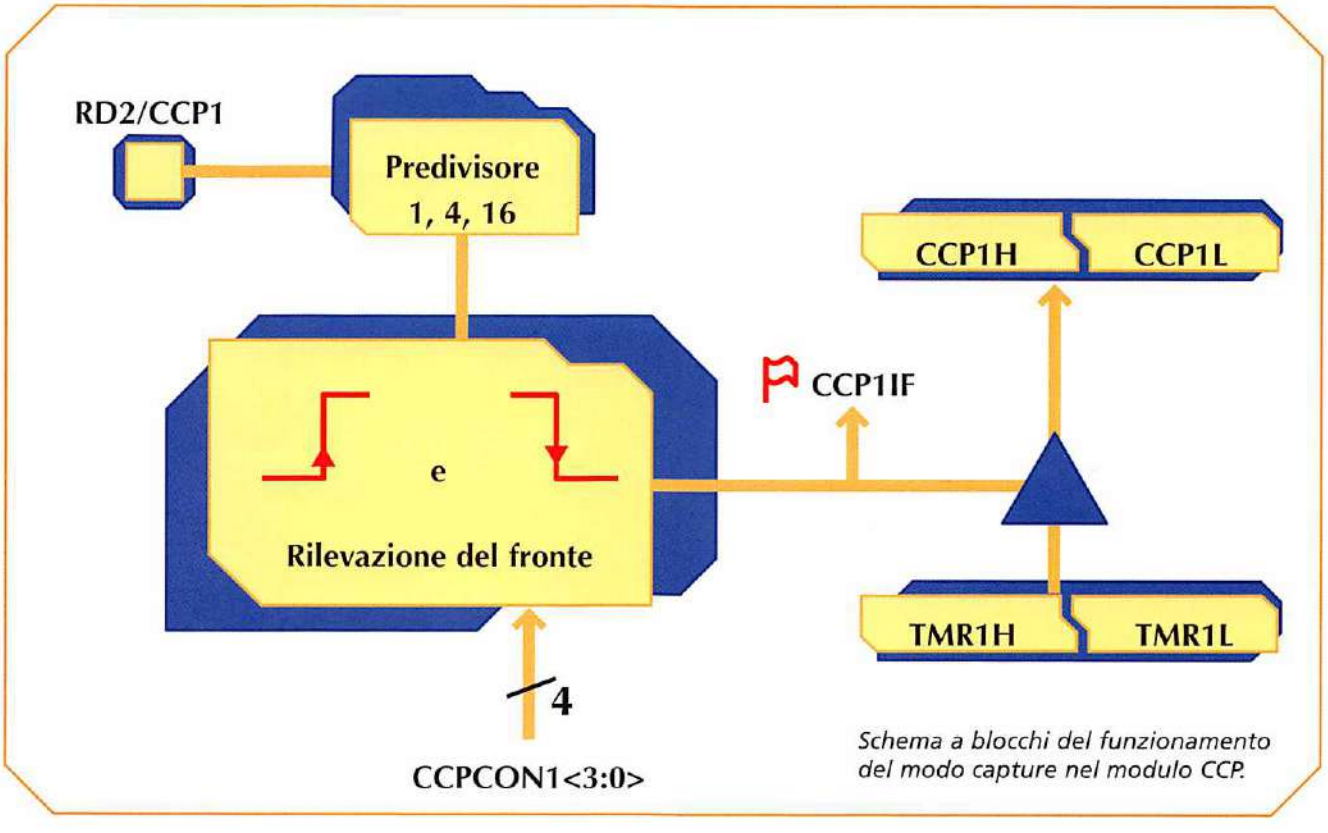
Per controllare un modulo CCP e scegliere le funzioni che vogliamo fargli eseguire esistono i registri CCP1CON e CCP2CON.

Modo capture

In questo modo il registro CCPRxH/L da 16 bit acquisisce il valore contenuto nel TMR1, sempre che si verifichi una delle seguenti condizioni:

– Un fronte di discesa.
– Un fronte di salita.
– Ogni quattro fronti di salita.
– Ogni sedici fronti di salita.

Sia il modo lavoro che la scelta dell'evento che provoca il capture si realizza configurando i quattro bit meno significativi del registro di controllo CCPxCON.



Schema a blocchi del funzionamento del modo capture nel modulo CCP.

Registro di controllo CCP1CON.

Normalmente si rimane in attesa di un evento esterno al microcontroller per mettere in marcia una routine specifica che realizzi una determinata funzione. Quando si genera un evento fra quelli citati in precedenza sul pin RC2/CCP1 (per il CCP1) o su RC1/T1OSI/CCP2 (per il CCP2) la coppia di registri concatenati CCPRxH:CCPRxL si carica con il valore presente sul TMR1 in quel momento.

Nel nostro caso, non disponendo del modulo CCP2, i registri coinvolti sono CCP1CON, CCPR1H:CCPR1L e il pin sul quale si riceve l'evento è il RC2/CCP1. A partire da questo momento ci concentreremo unicamente sul modulo che dispone il nostro PIC, anche se l'altro modulo è simile, eccetto che nella parte di attivazione.

Eseguendo il capture si attiva il flag CCP1IF, ubicato nel registro PIR1, e se i bit di abilitazione GIE e PIE1 (entrambi a 1) sono stati programmati correttamente, si genererà un interrupt nel processore. In questo momento si potrà leggere il contenuto del registro CCPR1H/L.

```
clrf    CCP1CON    ;Disattiva il CCP1
movlw   NUOVO      ;Nuova configurazione
movwf   CCP1CON    ;Il CCP1 funziona in un altro modo
```

Disattivazione del modulo CCP.

Quando si utilizza il modulo CCP1 in modo capture, il TMR1 deve essere configurato per lavorare in modo sincrono, come temporizzatore o contatore, non in modo contatore asincrono.

Quando si produce un capture e non viene letto il contenuto di CCPR1H/L si cancella il valore precedente, memorizzando il nuovo

CCP1M<3:0>	MODO LAVORO DEL MODULO
0000	Modulo CCP1 scollegato
0100	Modo capture a ogni fronte di discesa su RC2/CCP1
0101	Modo capture a ogni fronte di salita su RC2/CCP1
0110	Modo capture ogni 4 fronti di salita su RC2/CCP1
0111	Modo capture ogni 16 fronti di salita su RC2/CCP1
1000	Modo compare e attiva il pin RC2/CCP1 quando coincidono i valori
1001	Modo compare che disattiva (0) il pin RC2/CCP1 quando coincidono i valori
1010	Modo compare che genera interrupt software
1011	Modo compare in cui si genera un'attivazione speciale diversa per ogni modulo
11xx	Modo PWM

Tabella delle combinazioni CCP1M <3:0>

Indirizzo	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valore in POR, BOR	Valore nel resto dei resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	-----	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
87h	TRISC	Registro di configurazione della porta C								1111 1111	1111 1111
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	-----	CCPIE	TMR2IE	TRM1IE	0000 -000	0000 -000
0Eh	TMR1L	Registro di carico del byte meno significativo del registro da 16 bit TMR1								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Registro di carico del byte più significativo del registro da 16 bit TMR1								xxxx xxxx	uuuu uuuu
10h	T1CON	-----	----	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Registro del byte meno significativo del modulo CCP								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Registro del byte più significativo del modulo CCP								xxxx xxxx	uuuu uuuu
17h	CCP1CON	-----	----	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000

Registri che intervengono sul funzionamento dei moduli CCP in modo capture e compare.

valore acquisito. Se si eseguono delle modifiche alle condizioni in cui si va a effettuare l'operazione di capture conviene fermare o disattivare il modulo CCP, in modo che non si producano interrupt indesiderati durante l'operazione.

Nel riquadro della pagina precedente è possibile vedere la sequenza di istruzioni per realizzare la disattivazione.

Quando si disattiva il modulo CCP1 o smette di funzionare in modo capture, si cancella la codifica dei bit CCP1M3:M0.

Un'applicazione tipica del modo capture è la misura degli intervalli di tempo che intercorre fra gli impulsi che arrivano al pin RC2/CCP1 configurato come ingresso. In questo modo lavoro il TMR1 deve essere utilizzato come ingresso di clock esterno sincronizzato.

Modo comparazione

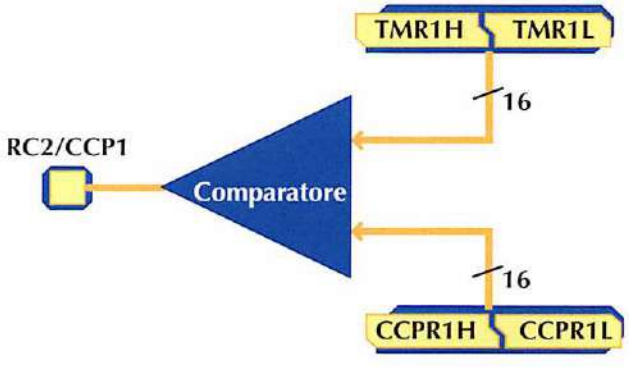
In questo modo il contenuto del registro CCPR1H/L si confronta continuamente con il contenuto del TMR1 (ricordiamo che questo deve lavorare in modo sincrono). Quando coincidono entrambi i valori, il pin RC2/CCP1, che sarà stato precedentemente configurato come uscita, può commutare a 1 oppure a 0 o può non variare, però si attiva il flag CCP1IF. Se i bit di abilitazione di interrupt sono stati attivati (GIE = PIE = 1) si genera un interrupt.

Nelle figure possiamo vedere il funzionamento di base del comparatore, così come l'architettura esterna o schema a blocchi dello stesso.

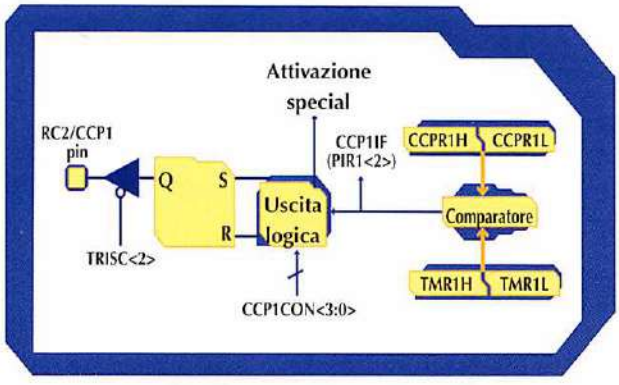
Nel modo di attivazione speciale, una comparazione corretta genera i seguenti eventi:

– Per CCP1: il TMR1 si imposta a 0 per cui il CCPR1 funziona come un registro dei periodi capace di generare interrupt periodicamente.

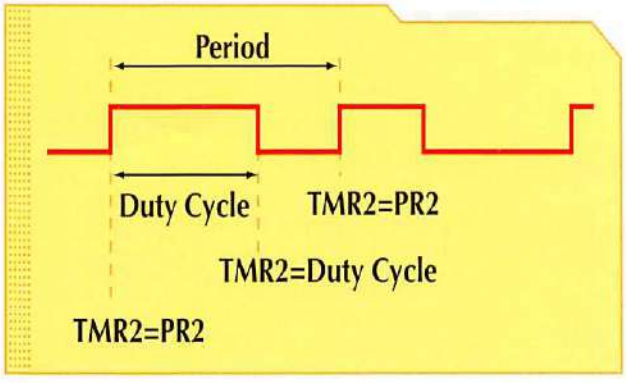
– Per CCP2: il TMR1 si imposta a 0 e si attiva una conversione analogico/digitale se questa è stata precedentemente autorizzata dal convertitore. Questo permette di eseguire periodicamente conversioni A/D senza che sia necessario il controllo del programma delle istruzioni.



Schema di base del funzionamento del comparatore.



Schema a blocchi del funzionamento del modulo CCP1 in modo compare.

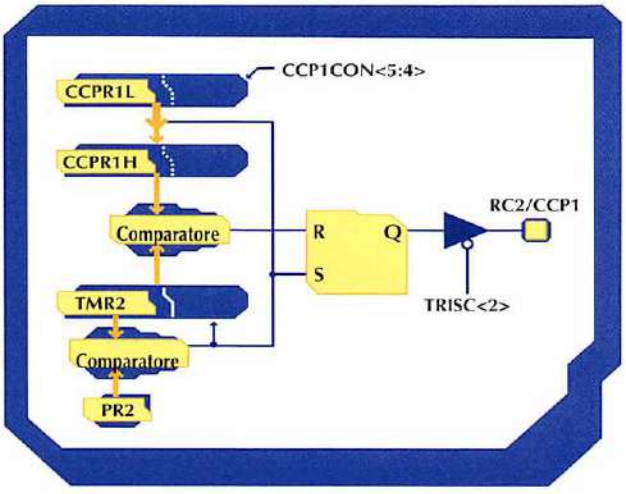


Possiamo controllare l'ampiezza dell'impulso.

Modo di modulazione di ampiezza degli impulsi o PWM

In questo modo il pin RC2/CCP1, programmato come uscita, commuta fra 0 e 1 a intervalli variabili di tempo. Quando il valore del registro PR2 coincide con gli 8 bit più significativi del TMR2, il pin di uscita passa a 1 e il TMR2 prende il valore 00 e ricomincia il conteggio. Il valore caricato su CCPR1L, passa a CCPR1H (registro slave del precedente) e si confronta con il TMR2. Quando entrambi coincidono il pin RC2/CCP1 passa a 0 e si ripete la sequenza.

Nell'immagine della figura si può capire meglio quanto esposto in precedenza, comunque riassumendo si può spiegare nel seguente modo: la comparazione fra PR2 e il TMR2 imposta a 1 il pin quando coincidono i registri, e la comparazione fra CCPR1H e TMR2 imposta a 0 il pin quando coincidono.



Schema a blocchi del modulo CCP1 nel modo PWM.

Tempo a 1 = (PR2 + 1) · 4 · Tosc · Predivisore
Tempo a 0 = DC1 · Tosc · Predivisore

Formule per calcolare il tempo degli impulsi.

Variando il valore di PR2 e CCPR1L si varia l'intervallo di tempo in cui il pin di uscita rimane a 1 oppure a 0 rispettivamente.

Per calcolare questi tempi possiamo utilizzare le formule della figura.

DC1 rappresenta il valore a 8 bit del registro CCPR1L concatenato con i bit <5:4> di CCP1CON. Gli 8 bit del TMR2 si concatenano con i due bit Q del clock interno facendolo contare ogni Tosc, invece di ogni 4 Tosc. Questo avviene quando lavoriamo con una risoluzione da 10 bit, però possiamo lavorare con una risoluzione da 9 bit (uno dei due bit meno significativi – CCP1CON<5:4> - a 0) o con una da 8 bit (i due bit meno significativi impostati a 0).

I passaggi da seguire per configurare il CCP1 in modo PWM sono quindi i seguenti:

1.- Caricare il valore che determina il periodo in PR2.

2.- Caricare il valore che corrisponde all'ampiezza dell'impulso (duty cycle) su CCPR1L:CCP1X:CCP1Y.

3.- Configurare il pin RC2/CCP1 come uscita.

4.- Assegnare il range al predivisore di frequenza del TMR2 e attivarlo programmando T2CON.

5.- Configurare il modulo CCP1 in modo PWM mediante il registro CCP1CON.

Il convertitore A/D del PIC16F870

Uno dei dispositivi più importanti e potenti del nostro microcontroller è il convertitore A/D. L'utilizzo di questi convertitori è molto comune in qualsiasi tipo di applicazione, per questo motivo ripasseremo il suo utilizzo e il suo funzionamento.

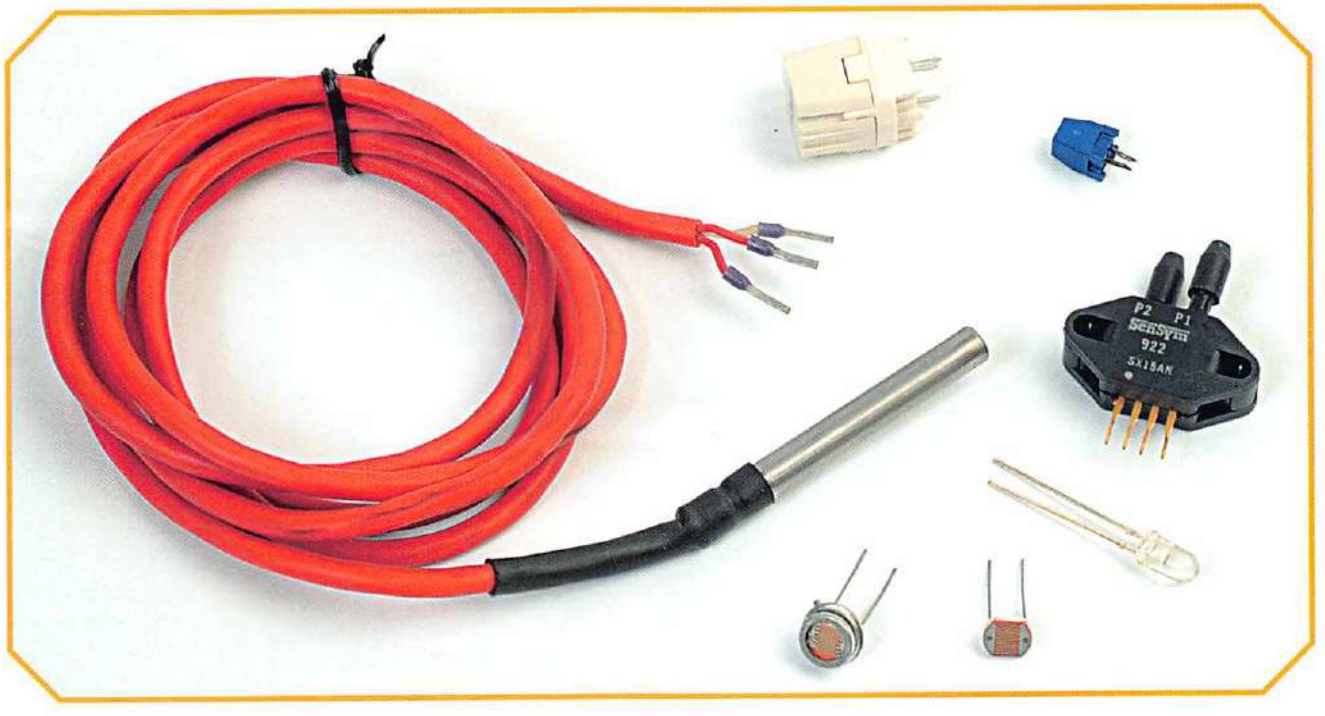
Convertitori A/D: necessari in mondo digitale

Tutti i microcontroller e i computer in generale lavorano con informazioni digitali. Definiscono qualsiasi grandezza con un insieme di bit che possono avere solamente due valori: 0 e 1. Nel mondo reale però la maggior parte delle informazioni e delle grandezze esistenti sono analogiche, variano fra un minimo e un massimo, passando fra infiniti valori presenti fra questi limiti. Così la temperatura, la velocità, la pressione, ecc. sono grandezze analogiche. La grafica della figura della pagina successiva corrisponde a un tipico segnale analogico che evolve lungo il tempo in modo continuo.

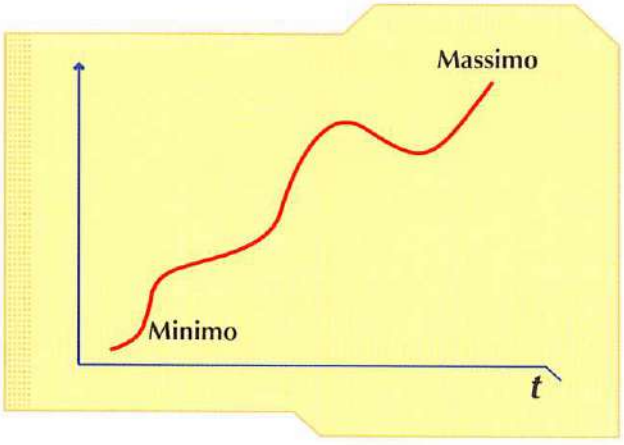
Le nostre macchine basate su computer sono digitali, però l'informazione che riceviamo è analogica. È necessario quindi, convertire l'informazione analogica in formato digitale per la sua elaborazione. Il computer genererà i risultati nello stesso formato digitale.

Se vogliamo che il sistema acquisisca l'informazione di un sensore di temperatura e in funzione di questa generi dei suoni, dobbiamo convertire in digitale il segnale del sensore, in modo che il microcontroller lo possa accettare, processare e generare i risultati, convertendoli successivamente in formato analogico per poterli ascoltare su un altoparlante. Sono necessari un convertitore A/D (Analogico/Digitale) e un altro D/A (Digitale/Analogico).

Vediamo ora alcuni aspetti molto importanti che ci permetteranno di capire bene il funzionamento dei convertitori A/D e D/A.



Sensori che forniscono un valore analogico.



Rappresentazione di una grandezza analogica.

RISOLUZIONE PER BIT

$$\text{Risoluzione} = \frac{((\text{Vref}+) - (\text{Vref}-))}{1024}$$

$$\text{Es: Risoluzione} = \frac{(5-0)}{1024} = 4.8 \text{ mV / bit}$$

Calcolo della risoluzione per bit.

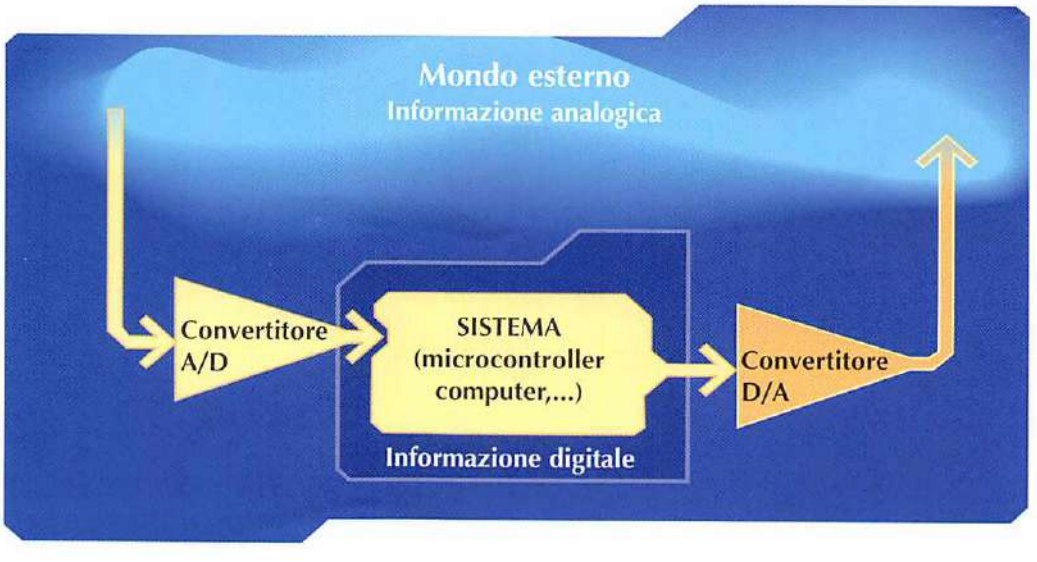
La risoluzione di un convertitore

A causa della pressante necessità di inserire convertitori nella maggioranza delle applicazioni industriali per la conversione di un segnale analogico in digitale, i modelli della famiglia PIC16F87X presentano un convertitore a 10 bit di risoluzione e 5 canali di ingresso, sui modelli a 28 pin e da 8 canali su quelli da 40 pin.

La risoluzione da 10 pin implica il poter distinguere fra $2^{10} = 1.024$ combinazioni, ovvero fra un minimo di 10 zero e un massimo di 10 uno. Con 10 bit di risoluzione il convertitore A/D, lavorando fra una tensione massima e un'altra minima, avrà una risoluzione per bit che sarà la 1.024ª parte del range fra questi limiti. Se lavoriamo fra un range massimo chiamato Vref+, e un altro minimo Vref-, la risoluzione per bit sarà quella indicata nella formula. Nel caso di Vref+ = 5 Vdc e Vref- = 0 V, la risoluzione per bit sarà 4,8 mV/bit, quindi fra ogni due valori digitali successivi avremo una risoluzione da 4,8 mV. Possiamo vedere nella tabella come ogni valore binario avrà un equivalente di tensione, e come fra ogni valore esiste una differenza uguale alla risoluzione.

Struttura e funzionamento di un convertitore A/D

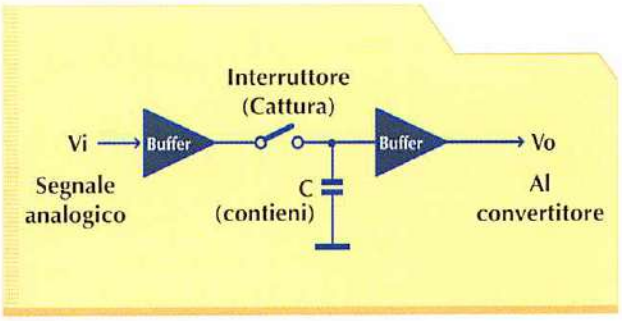
Il convertitore utilizza una tecnica di campionamento e ritenzione, chiamata anche di cattura e mantenimento (S&H: Sample and Hold), che equivale a caricare una capacità con la tensione che dobbiamo misurare. Si campiona un



Utilizzo del convertitore per il trattamento del dato.

VALORE DIGITALE	VALORE ANALOGICO
00 0000 0000	0,0000
00 0000 0001	0,0048
00 0000 0010	0,0096
...	...
...	...
...	...
11 1111 1101	4,9904
11 1111 1110	4,9952
11 1111 1111	5,0000

Equivalenza digitalelanalogica con risoluzione da 10 bit, Vref+ = 5V e Vref- = 0 V.



Schema del circuito (Sample and Hold) S&H all'ingresso del convertitore.

segnale analogico in un momento determinato e si mantiene il valore all'interno del condensatore per inviarlo successivamente per essere elaborato dal convertitore.

Nella figura in alto a destra è riportato lo schema di un circuito S&H formato da un interruttore che quando si chiude acquisisce un campione, mentre il condensatore mantiene il valore acquisito. Il tempo di campionamento o di presa del dato dipende dalla frequenza di clock del PIC e dalla velocità di conversione desiderata.

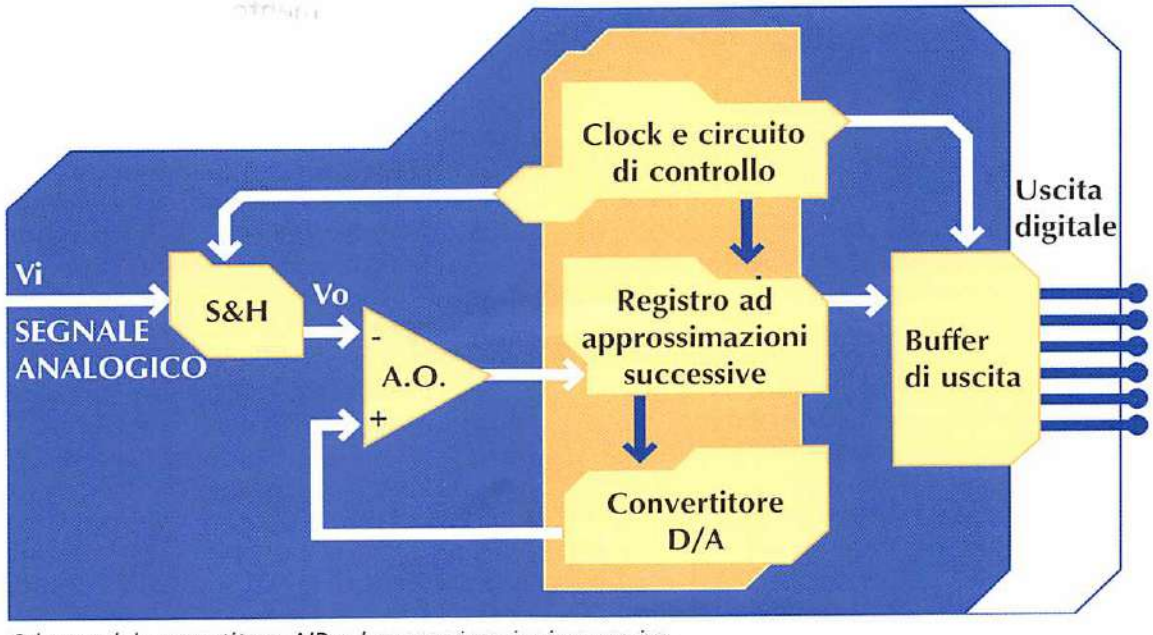
Convertitore A/D ad approssimazioni successive

Questo tipo di convertitore è quello di cui dispongono i PIC 16F87X. All'ingresso del circuito troviamo il circuito S&H seguito da un comparatore, la cui funzione consiste nel comparare il segnale analogico d'ingresso con quello che arriva da un convertitore D/A.

In funzione di quale dei due valori è maggiore, si cambia il valore dei bit del registro ad approssimazioni successive, fino a far coincidere entrambi i valori.

Il registro ad approssimazioni successive inizia prendendo il valore medio, quindi si pone a 1 il bit più significativo, lasciando gli altri a 0. Così se il registro è da 5 bit, acquisirà il valore 10000.

Questo valore è trasformato in segnale analogico mediante il convertitore D/A ed è portato sull'ingresso del comparatore. Se Vc>Vo il comparatore passerà a 1 questo farà sì che il registro abbassi il valore contenuto in esso,



Schema del convertitore A/D ad approssimazioni successive.

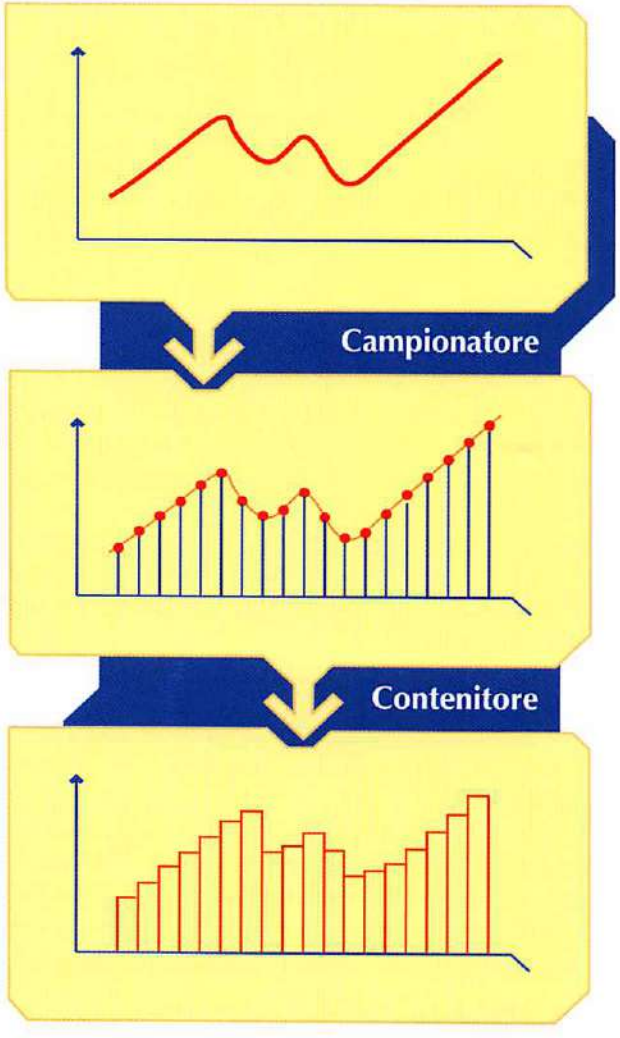
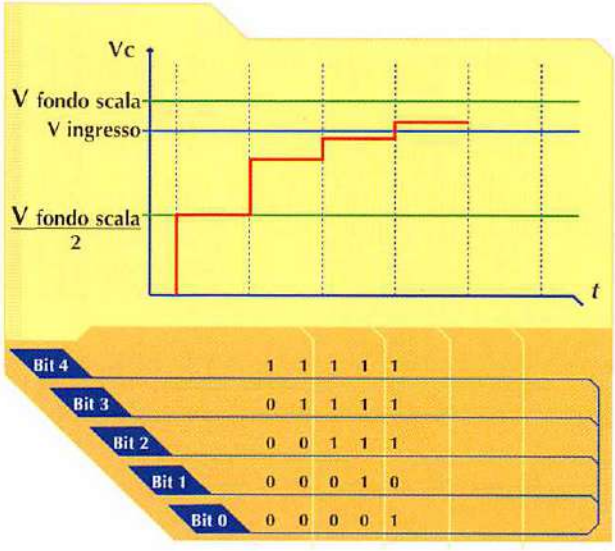


Grafico del funzionamento di un circuito S&H.

quindi il bit più significativo del registro passerà a 0 e verrà impostato a 1 il successivo, ottenendo così 01000. Si converte, si compara, e se ora Vc<Vo l'uscita del comparatore sarà 0, e il registro saprà che il valore che contiene è minore di quello richiesto.

A questo punto il registro eleva il suo valore e imposta a 1 il bit successivo, generando 01100. Si procede in questo modo sino a quando i valori coincidono, e avremo sul registro il valore digitale che corrisponde a quello analogico. Se il valore del registro supera quello di ingresso, si abbassa nuovamente il valore del registro stesso impostando a 0 il bit che è stato impostato a 1 l'ultima volta e passa a 1 il successivo. Se il valore del registro è minore di



Valori che adotta un registro ad approssimazioni successive durante una conversione.

quello dell'ingresso si imposta a 1 il bit successivo che dovrà essere comparato.

Con 5 comparazioni un registro ad approssimazioni successive da 5 bit termina la conversione, quindi possiamo garantire che il tempo della conversione è fisso e non dipende dalla grandezza da convertire.

Nella figura in alto è riportato il campionamento del registro ad approssimazioni successive.

Conclusioni

Dopo aver analizzato e capito il Convertitore Analogico/Digitale dobbiamo studiare come si utilizza all'interno del nostro microcontroller. Come tutti i dispositivi di quest'ultimo, il convertitore lavorerà con una serie di registri che dovremo conoscere per poterli utilizzare correttamente. Sono molte le applicazioni che hanno bisogno dell'utilizzo del convertitore, dato che è molto comune interagire con elementi di natura analogica.

Attualmente molti sensori comprendono al loro interno un sensore per fornire un'uscita digitale che può essere trattata direttamente da qualsiasi processore digitale. A causa del prezzo di questi sensori, che risulta essere elevato, continua a essere di uso comune l'utilizzo di convertitori integrati all'interno del processore.

Convertitore A/D del PIC16F870 (II)

Tutti i modelli di microcontroller PIC16F87X integrano al loro interno il convertitore A/D. Vediamo come funziona in questo microcontroller.

Generalità

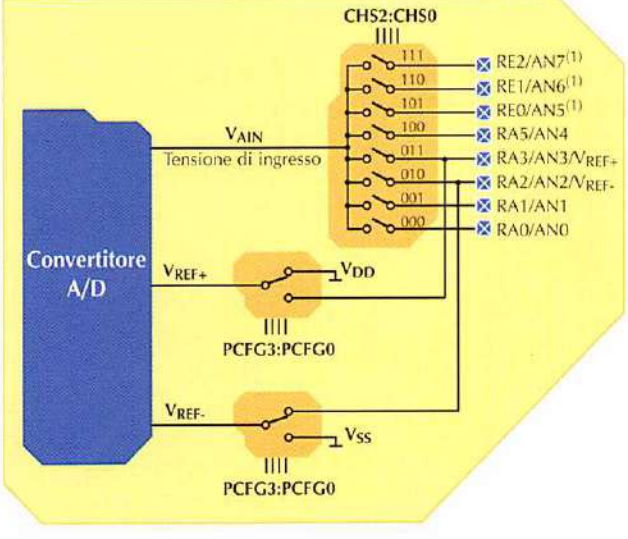
Il convertitore A/D si trova implementato sul silicio dei PIC16F87X. Si tratta di un convertitore ad approssimazioni successive da 10 bit di risoluzione. I modelli a 28 pin dispongono di cinque canali di ingresso, e quelli da 40 pin di otto canali. Nei PIC16F870/3/6 i cinque canali di ingresso sono supportati dai terminali della porta A.

Per il suo funzionamento sono necessarie due tensioni di rifrimento, chiamate VREF+ e VREF-, che si devono selezionare via software e che possono utilizzare tensioni interne (VDD e Vss) allo stesso microcontroller o tensioni esterne applicate ai pin RA3/AN3/VREF+ e RA2/AN2/VREF-.

Nello schema di collegamento si può vedere il segnale analogico che entra dalla linea VAIN tramite i canali di ingresso. La tensione di riferimento si può prendere internamente o esternamente.

Registri di lavoro e di controllo

Il convertitore utilizza quattro registri, due di controllo, chiamati ADCON0 e ADCON1, e due di risultato ADRESH e ADRESL.



Struttura interna del convertitore A/D del PIC.

I primi due registri governano il funzionamento del CAD (Convertitore A/D).

Il registro ADCON0

I due bit più significativi ADCS1:0 si utilizzano per selezionare la frequenza del clock impiegata nella conversione (Tosc = 1/Fosc). Nella tabella possiamo vedere la corrispondenza di ogni combinazione di bit con un tempo di conversione. Questo tempo è chiamato TAD, e lavorando con un CAD da 10 bit di risoluzione, avremo bisogno di un minimo di 12·TAD per terminare la conversione. La durata minima di TAD nei PIC16F87X è di 1,6 µs.

I bit CHS2:0 selezionano il canale tramite il quale entra il segnale analogico che deve essere convertito. Nella tabella della scheda successiva è riportata questa corrispondenza.

Il bit GO/$\overline{\text{DONE}}$ serve per inizializzare la conversione. Dopo averlo impostato a 1, rimarrà in questo stato per tutta la durata della conversione, e passerà automaticamente a 0 al termine. Il bit ADON è il bit di abilitazione del convertitore. Se il suo valore è 1 permette il funzionamento del convertitore, però se il suo valore è 0 disabilita il funzionamento dello stesso.

Il bit ADCON0<1> non è implementato e non ha nessuna funzione.

Il registro ADCON1

Il bit più significativo del registro ADCON1 è ADFM e serve per selezionare il formato del risultato della conversione. Se il suo valore è 1 il risultato sarà giustificato a destra, quindi, tenendo presente la risoluzione del convertitore da 10 bit, i sei bit più significativi del registro ADRESH saranno interpretati come 0. Quando il bit ADFM ha valore 0, il risultato della conversione si giustificherà a sinistra, quindi i sei bit meno significativi del registro ADRESL avranno valore 0. Nel grafico della figura è possibile vedere come si esegue la giustificazione del risultato. I bit successivi ADCON1<6:4> non sono implementati e i quattro bit meno significativi del registro PCFG3:0

Registri di controllo ADCON0 e ADCON1.

ADCON0 (INDIRIZZO : 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit7							bit0

ADCON1 (INDIRIZZO : 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit7							bit0

ADCS1	ADCS0	TAD
0	0	2·Tosc
0	1	8·Tosc
1	0	32·Tosc
1	1	Oscillatore RC interno

Selezione della frequenza di lavoro del CAD.

si utilizzano per configurare i pin del microcontroller. Quindi potremo definire quale pin desideriamo utilizzare come ingresso analogico, quali linee di I/O (digitali) vogliamo utilizzare per le tensioni di riferimento, ecc.. Nella tabella che riporta la configurazione della porta A, troveremo tutte le possibili combinazioni. La prima colonna contiene i bit PCFG3:0, le tre colonne successive fanno riferimento a canali che non sono implementati sul PIC16F870, le cinque colonne che seguono sono i canali della porta A, le due seguenti da dove verranno prese le tensioni di riferimento e l'ultima colonna indica il numero di canali analogici disponibili e il numero di canali utilizzati come tensioni di riferimento.

La lettera A indica che il canale si utilizzerà come canale analogico e la lettera D che si utilizzerà come I/O digitale.

I registri di lavoro

Nei registri di lavoro sono contenuti i risultati della conversione. Il registro ADRESH (1Eh) conterrà la parte alta del risultato e il registro ADRESL (9Eh) la parte bassa. Il risultato sarà a 10 bit, quindi, in funzione del bit ADFM, verranno riempiti con degli 0 i sei rimanenti.

Funzionamento in modo SLEEP

È possibile programmare una conversione mentre il microcontroller si trova in modo SLEEP, ma in questo caso è necessario utilizzare come clock di conversione il modo RC, dato che il clock normale del PIC nella condizione

Selezione TAD		Frequenza di lavoro			
TAD	ADCS1:ADCS0	20 MHz	5 MHz	1,25 MHz	333,3 kHz
2·Tosc	00	100 ns	400 ns	1,6 ns	6 µs
8·Tosc	01	400 ns	1,6 ns	6,4 µs	24 µs
32·Tosc	10	1,6 µs	6,4 µs	2,6 µs	96 µs
RC	11	2-6 µs	2-6 µs	2-6 µs	2-6 µs

Valore che prende TAD per diverse frequenze di lavoro.

CHS2	CHS1	CHS0	CANALE
0	0	0	Canale 0 (RA0/AN0)
0	0	1	Canale 1 (RA1/AN1)
0	1	0	Canale 2 (RA2/AN2)
0	1	1	Canale 3 (RA3/AN3)
1	0	0	Canale 4 (RA4/AN4)
1	0	1	Canale 5 (RA5/AN5) (1)
1	1	0	Canale 6 (RA6/AN6) (1)
1	1	1	Canale 7 (RA7/AN7)
(1) Non implementati sui PIC16F87X a 28 pin.			

Selezione del canale da convertire.

SLEEP è fermo. Quando si seleziona questo clock, il CAD attende un ciclo di istruzione prima di iniziare la conversione, questo permette l'esecuzione dell'istruzione SLEEP.

Al termine della conversione il risultato è caricato sui registri ADRESH:ADRESL, il bit GO/$\overline{\text{DONE}}$ passa a 0 e si genera un interrupt (se è stato abilitato), questo permette che il PIC esca dal modo SLEEP. Se l'interrupt non è stato abilitato, il CAD attende in questo stato che il processore esca dal modo SLEEP, fino a quel momento il risultato e il bit GO/$\overline{\text{DONE}}$ continueranno a essere validi.

Realizzazione di una conversione

Per eseguire una conversione sono necessarie sette fasi diverse:

1ª Fase: Configurazione del convertitore A/D: Configurazione dei canali di ingresso e delle tensioni di riferimento con ADCON1. Selezione del canale di ingresso da convertire, selezione del clock per la conversione e abilitazione o permesso di funzionamento in ADCON0.

2ª Fase: Configurazione dell'interrupt A/D: Se si desidera generare un interrupt al termine di una conversione dobbiamo cancellare il flag ADIF e abilitare o impostare a 1 i bit di abilitazione ADIE, PEIE e GIE.

3ª Fase: Attendere il tempo di acquisizione del campione da convertire.

4ª Fase: Inizio della conversione: dobbiamo impostare a 1 il bit GO/$\overline{\text{DONE}}$ del registro ADCON0.

5ª Fase: Attendere fino a quando si completa la conversione: Testeremo il bit GO/$\overline{\text{DONE}}$ fino a quando sarà 0 e attenderemo fino a quando si produce un interrupt o si attiva il bit di flag ADIF.

6ª Fase: Lettura del risultato della conversione: leggeremo il risultato di ADRESH:ADRESL e imposteremo il bit ADIF a 0.

7ª Fase: Conversione successiva: per iniziare una nuova conversione dobbiamo attendere come minimo un tempo di 2·TAD per eseguire l'acquisizione successiva, ripetendo tutti i passaggi dal numero 3.

Esempio

Vogliamo configurare i terminali RA0, RA1 e RA3 come ingressi analogici per il CAD, e i pin RA2 e RA5 come I/O digitali. Come deve essere programmato il PIC16F870?

La soluzione consisterà nello scrivere sui quattro bit meno significativi del registro ADCON1 (PCFG3:0) il valore 0100. La tensione di riferimento in questo caso sarà V_{DD}.

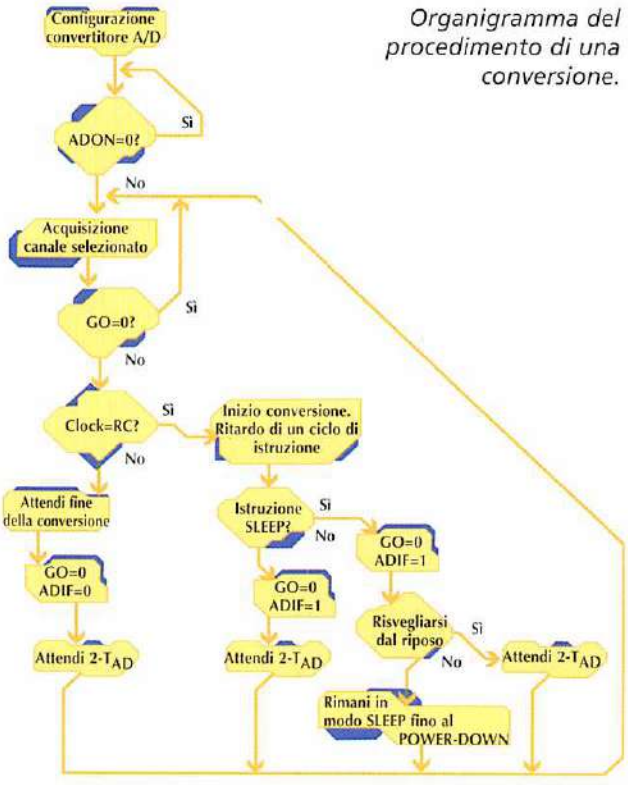
Funzionamento del bit ADFM nel registro ADCON1.

PCFG3:0	AN7/(1) RE2	AN6/(1) RE1	AN5/(1) RE0	AN4/ RE5	AN3/ RE3	AN2/ RE2	AN1/ RE2	AN0/ RE0	VREF+ RE2	VREF- RE2	CHAN/ Refs
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011X	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

Configurazione della porta A in funzione del valore di PCFG3:0.

Passo	Procedimento
1	Configurazione del CAD
2	Configurazione dell'interrupt A/D (opzionale)
3	Attendere il tempo di acquisizione
4	Inizio della conversione
5	Attendere il tempo di conversione
6	Lettura del risultato
7	Conversione successiva

Riassunto dei passaggi da eseguire per realizzare una conversione.



Organigramma del procedimento di una conversione.

Porta seriale sincrona (SSP)

La porta Seriale Sincrona (SSP) è un dispositivo progettato per supportare un'interfaccia seriale sincrona. Questa interfaccia risulta particolarmente adatta per la comunicazione del microcontroller con dispositivi quali display, memorie, convertitori, ecc. Lavora nei due modi possibili:

1. *Interfaccia Seriale di Periferiche (SPI)*
2. *Interfaccia Inter-Circuiti (I^2C)*

Modo SPI

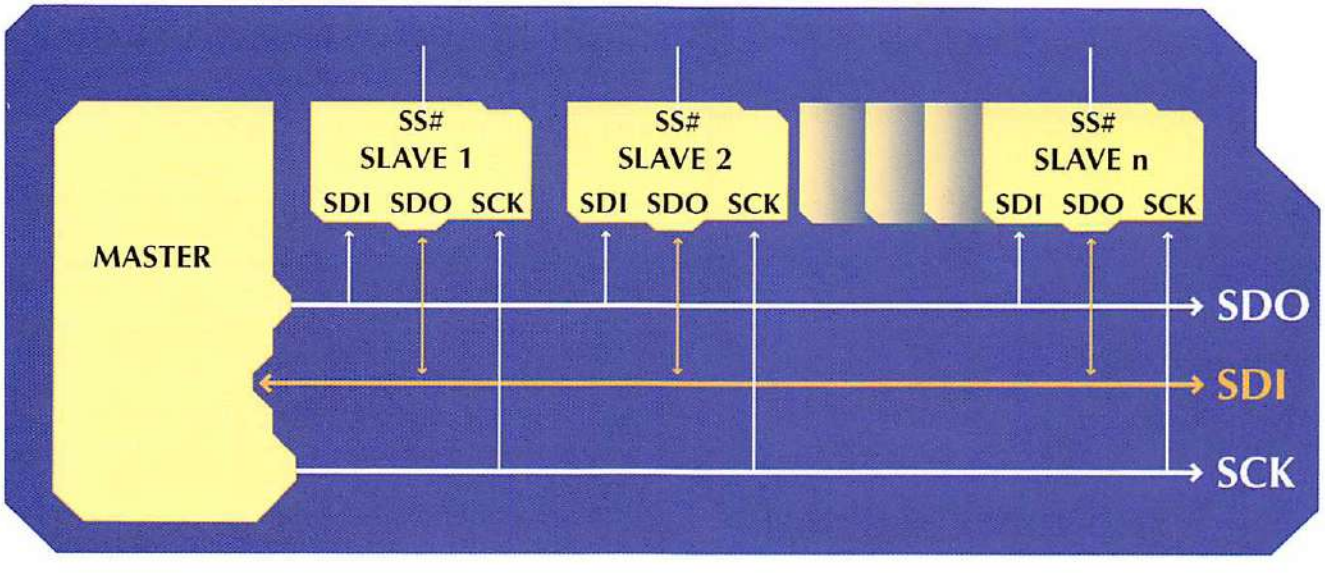
Questo modo si utilizza per collegare diversi microcontroller, della stessa famiglia o di famiglie differenti, sotto il formato "master-slave", sempre che dispongano di una interfaccia compatibile. In questo formato esiste sempre un elemento che svolge la funzione di master, che è quello che genera gli impulsi di clock per la sincronizzazione nell'invio e nella ricezione dei bit in serie. La trasmissione utilizzata è seriale sincrona ed è supportata da tre linee, come possiamo vedere nella tabella della figura:

SDO: Uscita dei dati
SDI: Ingresso dei dati
SCK: Clock

La quarta linea, SS#, è un segnale ausiliario che serve per attivare lo slave con il quale il master stabilirà la comunicazione.

Il supporto per lavorare in modo SPI si trova integrato sul silicio, e per poter eseguire la trasmissione sono necessarie delle linee di ingresso e uscita. Dato che i pin del PIC16F870 sono multifunzionali, le funzioni di comunicazione sono state assegnate a pin della porta A e della porta C: RA5/SS#, RC3/SDO, RC4/SDI e RC5/SCK. Ciascun segnale deve essere programmato come ingresso o uscita secondo la condizione, utilizzando i bit del registro TRIS. In questo modo si possono espandere facilmente le risorse del microcontroller (memoria, convertitori, ecc.) utilizzando queste linee per adattare dispositivi esterni che supportano questo protocollo.

Qualsiasi funzione del modo SPI si annulla impostando il valore opposto alla sua condizione nel bit corrispondente del registro TRIS. In questo modo, se vogliamo solamente ricevere dei dati, definiremo il pin che supporta SDO come ingresso, annullandone, in pratica, la funzione.



Sistema di comunicazione seriale sincrono in modo SPI.

Nome	Funzionamento
SD0	Uscita dei dati seriali
SDI	Ingresso dei dati seriali
SCK	Segnale di clock per la sincronizzazione tra l'emettitore e il ricevitore
SS#	Selezione dello slave con cui il master comunicherà

Linee di trasmissione del modo SPI.

Programmazione del modo SPI

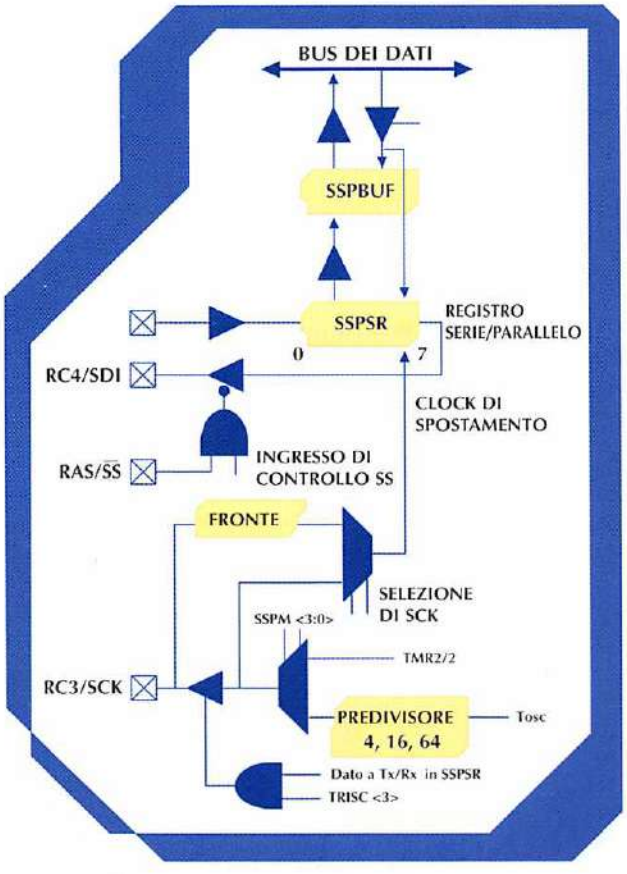
Per determinare tutte le caratteristiche del modulo di comunicazione nel modo SPI si utilizzano due registri specifici della memoria RAM: SSPCON (indirizzo 14h) e SSPSTAT (indirizzo 94h).

Il registro di controllo SSPCON

Con il registro di controllo SSPCON si selezionano le diverse opzioni di lavoro. I quattro bit meno significativi SSPM3:0 stabiliscono il modo lavoro del modulo di comunicazione secondo la codifica della tabella della figura. Possiamo vedere come programmare il modulo MSSP nei modi SPI e I2C.

Se il bit CKP vale 1 significa che lo stato di inattività del segnale di clock è quello alto e se vale 0 sarà il contrario. SSPEN è il bit di abilitazione del modulo di comunicazione MSSP, quindi se il suo valore è 0 le linee che supportano i segnali SDO, SDI, SCK e SS# si possono utilizzare come I/O, mentre se vale 1 queste linee svolgono le funzioni di comunicazione.

Il bit SSPOV è un flag o segnalatore che indica, quando è a 1, che si è verificato l'overflow sul registro SSPBUF, ovvero è stato ricevuto un nuovo dato e non è stato letto quello precedente. Il bit più significativo WCOL è un altro flag che indica, quando vale 1, che si è verificata una "collisione", in altre parole è stato caricato il registro SSPBUF su SSPSR senza aver dato il tempo all'informazione contenuta in precedenza di uscire. Quando si riceve un dato durante una trasmissione, esso viene ignorato e si attiva il bit WCOL.



Struttura interna del modulo MSSP quando funziona in modo SPI.

Il registro di stato SSPSTAT

Il bit più significativo di questo registro, SMP, svolge la sua funzionalità unicamente nel modo master (nel modo slave non ha significato e vale sempre 0) e serve per mostrare il dato di ingresso al termine dell'impulso del clock (SMP=1) o a metà dello stesso (SMP=0). Il bit CKE dipende dal bit CKP, quindi in funzione di questo bit del registro SSPCON, il bit CKE realizzerà una funzione o l'altra. Nella tabella della figura è riportata la funzionalità di questo bit. L'ultimo bit che ha importanza in questo tipo di comunicazione è il BF, il bit meno significativo, che quando è a 1, indica che il registro SSPBUF è pieno.

Struttura interna dei registri SSPCON e SSPSTAT.

Funzionamento del modo SPI

Quando si riceve un dato utile, questo viene caricato via seriale su SSPSR, e passa a SSPBUF in parallelo. Il dato da trasmettere si deposita su SSPBUF e da qui passa a SSPSR. Si possono ricevere e trasmettere dati simultaneamente. SSPSR è un registro di spostamento che funziona serie/parallelo/serie.

Quando si termina di trasmettere o ricevere un dato completo si attiva il bit BF (Buffer Full) del registro SSPSTAT. Viene anche attivato il flag SSPIF e, se il bit di abilitazione è attivato, si genera un interrupt.

Quando si inizia un trasferimento il master invia tramite la linea SDO a tutti gli slave il codice con il quale vorrà mantenere la comunicazione, i comandi determinano il tipo di operazione (lettura/scrittura) e altre caratteristiche come il numero di byte da leggere o da scrivere.

Alcune informazioni sul bus I2C

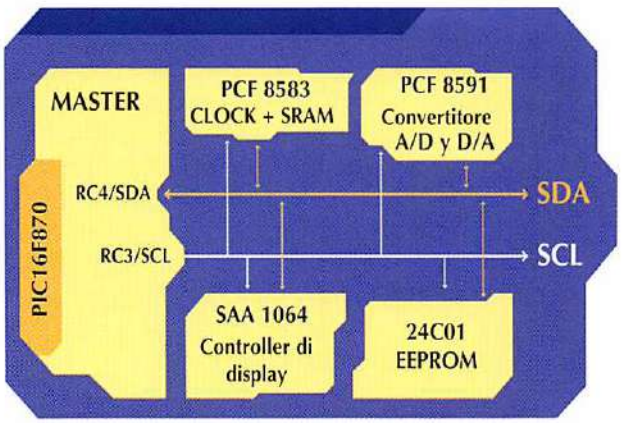
Questa interfaccia di comunicazione fu sviluppata da Philips e utilizza solamente due fili twistati e una massa comune per il collegamento dei diversi dispositivi, progettati in modo specifico per supportare questo protocollo, assicurando quindi un'alta affidabilità della comunicazione. Oltre alla semplicità, dato che utilizza solamente due linee, raggiunge una velocità massima di 400 Kbps, può collegare 128 dispositivi ed è possibile realizzare collegamenti a distanze considerevoli. Per queste ragioni, questo metodo è molto utilizzato nella building automation, nel controllo di distribuzione di acqua, gas ed elettricità, ecc., oltre che per l'interconnessione tra circuiti integrati all'interno di circuiti stampati.

SSPM3:0	Modo di lavoro
0000	Modo master SPI con Fosc/4
0001	Modo master SPI con Fosc/16
0010	Modo master SPI con Fosc/64
0011	Modo master SPI con clock uguale all'uscita di TMR2/2
0100	Modo slave SPI con clock su SCK e SS# a livello basso
0101	Modo slave SPI con clock su SCK e SS# libero per essere utilizzato come I/O
0110	Modo slave I2C con indirizzo da 7 bit
0111	Modo slave I2C con indirizzo da 10 bit
1000	Modo master I2C con clock (Fosc/4)x(SSPAD+1)
1001	Non implementato
1010	Non implementato
1011	Modo master I2C controllato da firmware
1100	Non implementato
1101	Non implementato
1110	Modo master I2C controllato da firmware con indirizzo da 7 bit e interrupt attivato
1111	Modo master I2C controllato da firmware con indirizzo da 10 bit e interrupt attivato

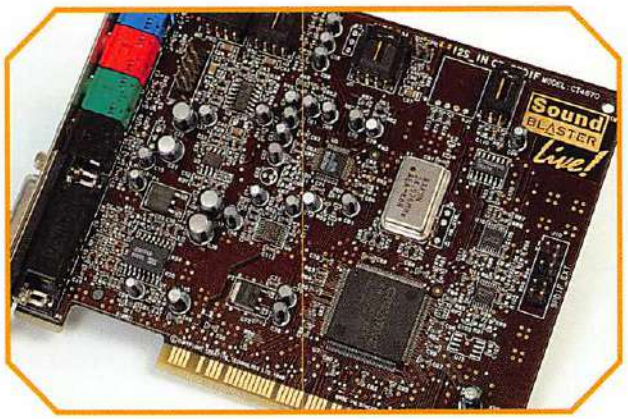
Modo lavoro del modulo di comunicazione MMSP in funzione del valore dei bit SSPM3:0.

CKP	CKE	Funzionalità
0	0	Lo stato di inattività del segnale di clock è quello basso e il dato si trasmette sul fronte di discesa
	1	Lo stato di inattività del segnale di clock è quello basso e il dato si trasmette sul fronte di salita
1	0	Lo stato di inattività del segnale di clock è quello alto e il dato si trasmette sul fronte di discesa
	1	Lo stato di inattività del segnale di clock è quello alto e il dato si trasmette sul fronte di salita

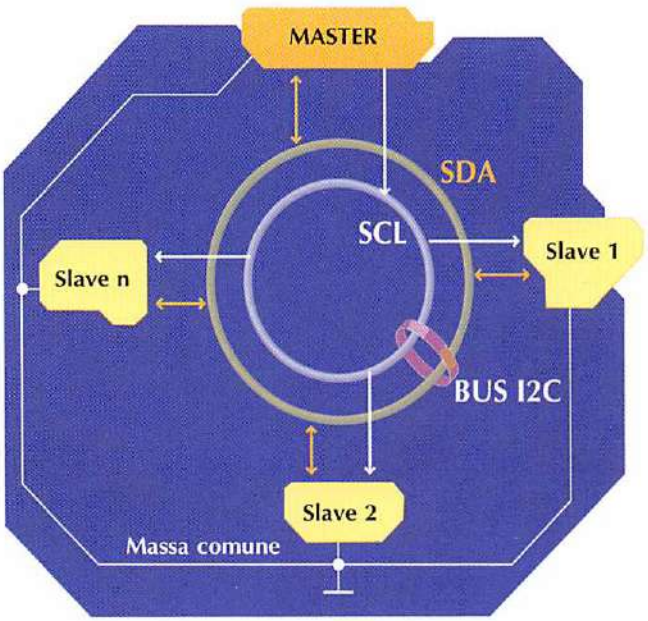
Funzione del bit CKE del registro SSPSTAT.



Sistema di comunicazione seriale sincrono in modo I2C.



Il modo I2C è utilizzato anche per i collegamenti dei circuiti integrati all'interno dei circuiti stampati.



Architettura della rete di collegamento con il bus I2C.

La linea SDA supporta i dati trasmessi in modo bidirezionale, e l'altra, chiamata SCL, gli impulsi di clock per il sincronismo dell'emettitore e del ricevitore.

Anche in questo protocollo esiste un master che è quello che inizia e termina il trasferimento generale e fornisce il clock al resto dei dispositivi. Lo slave è il dispositivo indirizzato al master, mediante 7 bit, questo limita il numero dei componenti a 128 (2^7=128).

L'inizio della trasmissione si determina con il bit di inizio (S) e la fine con un altro bit di stop (P).

Esistono numerosi tipi di circuiti integrati quali memorie, A/D e D/A, controller di display, ecc. progettati in modo specifico per lavorare con il bus I2C, in quanto l'utilizzo di questa interfaccia è molto comune nel mondo dell'elettronica.

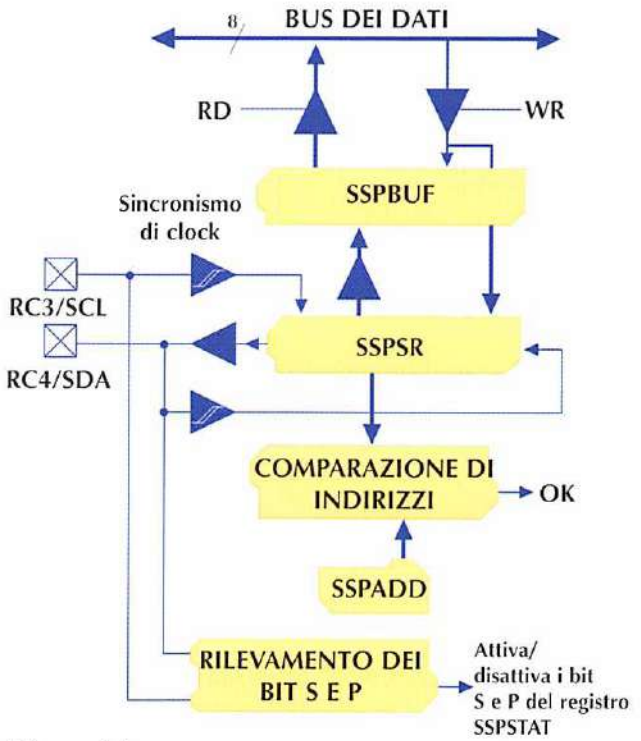
Il bus I2C

Dato che il protocollo di comunicazione I2C è molto diffuso nel mondo dell'elettronica lo analizzeremo nel dettaglio. Ricordiamo quanto visto nel fascicolo precedente riguardo al modo lavoro del SSP, cioè due sole linee di trasmissione (SDA: trasferimento e SCL: clock), modo lavoro master-slave in cui il master determina le caratteristiche della comunicazione, velocità massima 400 Kbps, ecc.

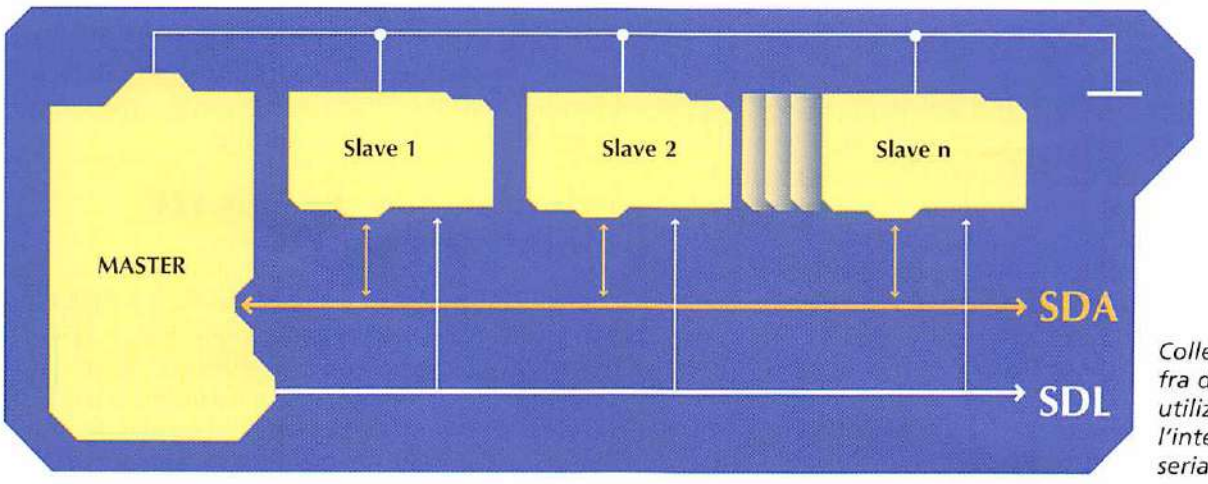
Funzionamento del bus I²C

Nel protocollo I2C ogni dispositivo ha assegnato un indirizzo da 7 oppure 10 bit, che viene inviato dal master quando inizia il trasferimento con uno di essi. Dopo l'indirizzo si aggiunge il bit di ricezione/trasmissione o lettura/scrittura (R/W#). Inoltre con alcuni dispositivi I2C è necessario aggiungere anche una serie di comandi per specificare le condizioni di funzionamento degli stessi. Ad esempio ci sono memorie di tipo I2C per le quali dobbiamo indicare gli indirizzi a cui vogliamo accedere, altri dispositivi quali controller di display a cui dobbiamo inviare comandi che determinano su quale display dovrà essere visualizzata l'informazione, ecc.

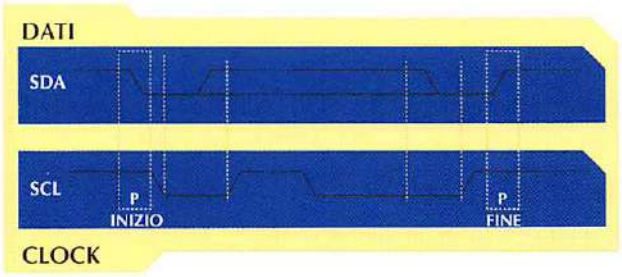
Sia i dati che i comandi si trasmettono sotto forma di byte e al termine del trasferimento di ogni byte si genera un bit speciale di trasferimento (ACK#) da parte del ricevitore, che permette il riallacciamento della comunicazione. In mancanza del bit di riconoscimento si termina la procedura di trasferimento. È necessaria la presenza di un modulo di arbitrag-



Schema interno del funzionamento dell'interfaccia I2C.



Collegamento fra dispositivi utilizzando l'interfaccia seriale I2C.



L'inizio della trasmissione è segnalato con un bit di inizio (S) e la fine con il bit di stop (P).

Formato per indicare un indirizzo da 7 bit.

Esempio di trasmissione di indirizzo di formato a 7 bit.

gio che si incarichi di gestire l'esistenza di un unico master nell'istante sul bus condiviso.

Trasferimento di dati nel modo I2C

Il master inizia e termina il trasferimento. Inizia con la condizione START, che consiste nella generazione di un fronte di discesa sulla linea SDA, mentre sulla linea SCL si mantiene un livello alto. Per terminare la trasmissione il master genererà il segnale di STOP, il quale consiste in un fronte di salita su SDA con il livello alto su SCL.

Nel primo byte di trasferimento il master invia il primo indirizzo dello slave con cui vuole comunicare. L'indirizzo è fornito con i primi sette bit, in quanto l'ottavo è quello di lettura/scrittura (R/W#). Dopo il segnale di START gli slave lasciano in alta impedenza la linea SDA e attendono l'indirizzo inviato dal master. Sul nono impulso generato da quest'ultimo, si attende che lo slave scelto imposti a livello basso la linea SDA come segno di riconoscimento (ACK#), in modo da poter riannodare il trasferimento. Se si utilizza il formato da 10 bit per indirizzare gli slave, potremo collegare più di 128 dispositivi sulla rete, (limite esistente quando l'indirizzamento è da 7 bit). In questo caso il master utilizza i due byte iniziali per inviare l'indirizzo, come si può vedere nelle immagini delle figure.

SMP: Bit di presentazione dei dati
In modo Master:
1 = I dati si campionano alla fine del periodo di clock
0 = I dati si campionano alla metà del periodo di clock
SCE: Bit di selezione dei livelli sulle linee SDA e SCL
1 = Livello alto
0 = Livello basso
D/A#: Bit dei dati/indirizzi
1 = L'ultimo byte ricevuto è un dato
0 = L'ultimo byte ricevuto è un indirizzo
P: Bit di stop (Si cancella scollegando il modulo SSP)
1 = È stato rilevato il bit di stop P
0 = Non è stato rilevato il bit di stop
S: Bit di inizio (Si cancella scollegando il modulo SSP)
1 = È stato rilevato il bit di inizio S
0 = Non è stato rilevato il bit di inizio
R/W#: Bit di lettura/scrittura
1 = Lettura
0 = Scrittura
UA: Aggiorna indirizzo (Modo Slave da 10 bit)
1 = Indica che l'utente deve aggiornare l'indirizzo su SSPADD
0 = Non è necessario aggiornare l'indirizzo
BF: Buffer pieno
Ricezione
1 = Ricezione completa, SSPBUF pieno
0 = Ricezione incompleta, SSPBUF vuoto
Trasmissione
1 = Trasmissione in corso
0 = Trasmissione completa

Registro SSPSTAT.

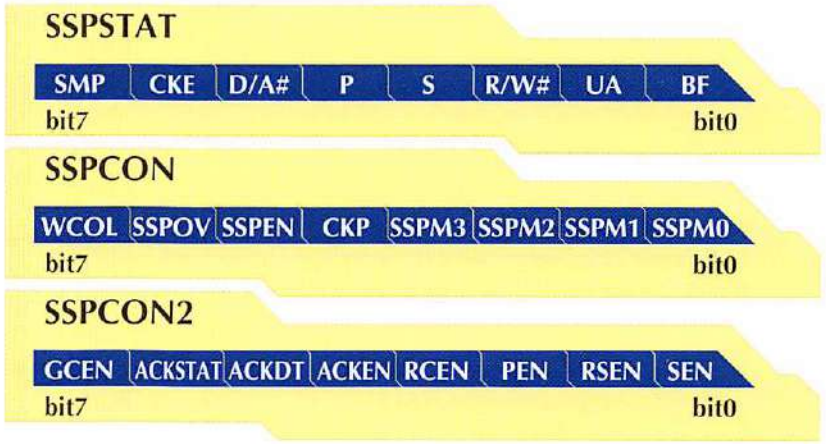
Integrazione del bus I2C all'interno del PIC

Il bus I2C è supportato dai pin RC3 e RC4, che svolgono le funzioni delle linee SCL e SDA rispettivamente, se in precedenza sono stati configurati come ingressi. I dispositivi fisici interni del PIC si riducono ai registri SSPBUF, SSPSR e SSPADD, a quelli di configurazione

SSPSTAT, SSPCON e SSPCON2 oltre a una circuiteria complementare che possiamo osservare nella figura della pagina precedente.

Il registro SSPBUF è il registro che contiene il byte da trasmettere o quello che si riceve. SSPSR è un registro di spostamento serie/parallelo che ha il compito di convertire i dati che sono caricati in modo seriale in parallelo, e viceversa. SSPADD è il registro degli indirizzi che identifica il dispositivo (modo slave) o che lo direziona (modo master).

SSPSTAT è il registro di stato che come SSPCON, si utilizza nel modo SPI. In ultimo per lavorare con il bus I2C, esiste un registro ausiliario di controllo chiamato SSPCON2.



Configurazione interna dei registri di stato e controllo.

Come funziona l'hardware

Quando si vuole trasferire un byte lo si scrive sul buffer o registro SSPBUF, nel caso di ricevimento di un byte da parte del bus I2C, quest'ultimo è caricato ugualmente su SSPBUF. Dopodiché il registro convertitore SSPSR esegue la trasformazione dell'informazione dal modo parallelo a quello seriale, in caso della trasmissione, oppure trasforma l'informazione dal modo seriale in cui è stata ricevuta tramite SDA, in modo parallelo.

Quando SSPSR è stato riempito con gli otto bit ricevuti tramite SDA, questi ultimi vengono spostati sul buffer SSPBUF, che li mette a disposizione della CPU in formato parallelo.

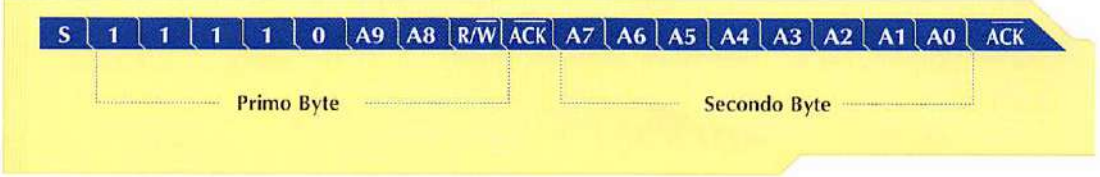
Se si riceve un altro byte su SSPRS prima che la CPU abbia letto quello precedente, si attiva il flag SSPOV di "overflow" o superamento del flusso dei dati, sesto bit di SSPCON. Sul registro SSPADD si scrive l'indirizzo dello slave con cui vogliamo stabilire la comunicazione.

SSPM3:0	Funzione
1000	Modo master I2C con clock (Fosc/4)x(SSPAD+1)
1011	Modo master I2C controllato da firmware
1110	Modo master I2C controllato da firmware con indirizzo da 7 bit e interrupt attivato
1111	Modo master I2C controllato da firmware con indirizzo da 10 bit e interrupt attivato

Bit SSPM3:0 del registro SSPCON.

I registri SSPSTAT, SSPCON e SSPCON2

Il primo registro di cui parleremo, fra quelli che hanno il compito di programmare in modo lavoro nel bus I2C, è il registro di stato SSPSTAT. Questo registro si utilizza anche per la trasmissione SPI, però adesso presenteremo le funzioni di ogni bit nel modo di trasmissione che ci interessa. Nella tabella della figura possiamo studiare il funzionamento del registro. Quando abbiamo studiato il modo SPI abbiamo analizzato le possibili combinazioni dei bit SSPM3:0. Quattro di questi appartengono al modo lavoro I2C. Nella tabella della pagina precedente, sono raccolte queste com-



Formato della trama per indicare un indirizzo da 10 bit.

Fase	Processo
1	Si genera la condizione di START impostando il bit SEN=1
2	Il bit flag SSPIF=1 quando termina la condizione di START
3	Si carica sul registro SSPBUF l'indirizzo dello slave e R/W#
4	Si manda in modo seriale su SDA il contenuto di SSPBUF
5	Lo slave indirizzato genera la condizione di riconoscimento sul nono impulso di clock e si rileva sul registro SSPCON2
6	Si genera un interrupt al termine del nono impulso di clock e SSPIF=1
7	Si carica un dato su SSPBUF
8	Quando lo slave ha ricevuto il dato il master riceve il suo riconoscimento
9	Si genera un interrupt sul nono bit e SSPIF=1
10	Con PEN=1 si genera la condizione di STOP per terminare il trasferimento
11	Completata la condizione di STOP si genera un interrupt

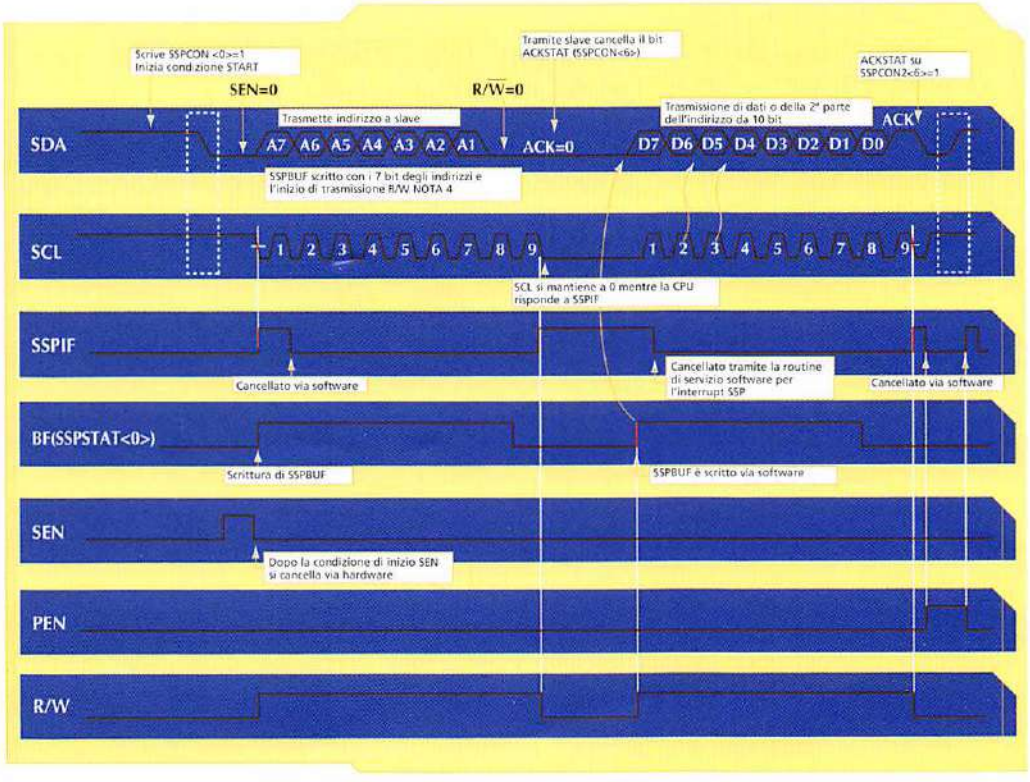
Fasi della trasmissione (master).

binazioni. Dei rimanenti bit che formano il registro SSPCON dobbiamo sapere che per fare in modo che il modulo MSSP funzioni è necessario impostare il bit SSPEN=1, perché altrimenti le linee RC3 e RC4 si comporteranno come linee convenzionali di I/O digitali. Se CKP=0 i livelli di questi pin si configurano secondo le specifiche del bus I2C e si mette in marcia il clock. Quando SSPOV=1 significa che si è verificato un overflow, cioè è stato riempito SSPSR senza aver letto il contenuto precedente di SSPBUF. In ultimo, il bit WCOL indica se si è verificata una collisione nel tentativo di utilizzare SSPBUF senza aver utilizzato il valore precedente.

Per quanto riguarda SSPCON2, il bit GCEN funziona solamente in modo slave. Quando si riceve il bit di riconoscimento ACK# dello slave, si imposta a 1 il bit ACKSTAT. Per iniziare la sequenza di generazione della condizione di riconoscimento, si imposta a 1 il bit ACKEN. Si pone RCEN=1 se vogliamo che il master funzioni come ricevitore e PEN=1 per generare la condizione di STOP. La ripetizione della condizione di START si ottiene con RSEN=1 e con SEN=1 si inizia una normale condizione di START.

Sequenza di lavoro in modo master

Nella figura a lato, possiamo vedere un cronogramma di una tipica trasmissione tramite il bus I2C di un PIC 16F87X che funziona come master. Nella tabella sono riportate le fasi di una operazione di trasmissione da parte del master.



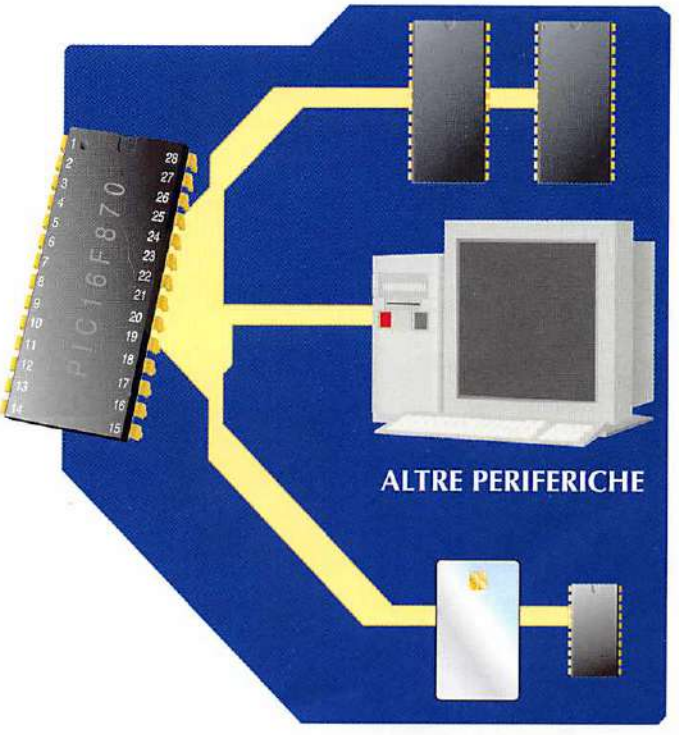
Cronogramma del PIC che funziona come master.

Il modulo USART

Tra i dispositivi interni del nostro microcontroller che andiamo ad analizzare, uno dei più importanti e potenti è il modulo di comunicazione USART (Universal Synchronous Asynchronous Receiver Transmitter). Si tratta di uno dei due moduli di I/O seriali del microcontroller, anche conosciuto come Interfaccia di Comunicazione.

Generalità

USART è un'interfaccia che permette la comunicazione con altri dispositivi. Se si configura come un sistema asincrono di tipo Full-Duplex potremo far comunicare il PIC con periferiche tipo terminali CRT e personal computer. Se si configura come sistema sincrono tipo Half-Duplex potrà comunicare con periferiche quali circuiti convertitori A/D o D/A, memorie EEPROM, ecc. Questa capacità di comunicare con altri dispositivi è indispensabile in molti progetti, come ad esempio, il controllo della climatizzazione di un edificio, dove ogni microcontroller si occupa di una determinata zona, anche se tutti risponderanno agli ordini di un processore centrale. Possiamo anche prendere come esempio i lettori di SmartCard, in cui il PIC deve comunicare con il chip di memoria integrato sulla scheda.



Il PIC può comunicare con diverse periferiche.

Configurazione della USART

Abbiamo visto che in funzione di come si configura, il modulo USART potrà stabilire una comunicazione con diversi tipi di periferiche, dobbiamo però vedere quali sono le opzioni di configurazione. Il modulo USART si può configurare nei seguenti modi:

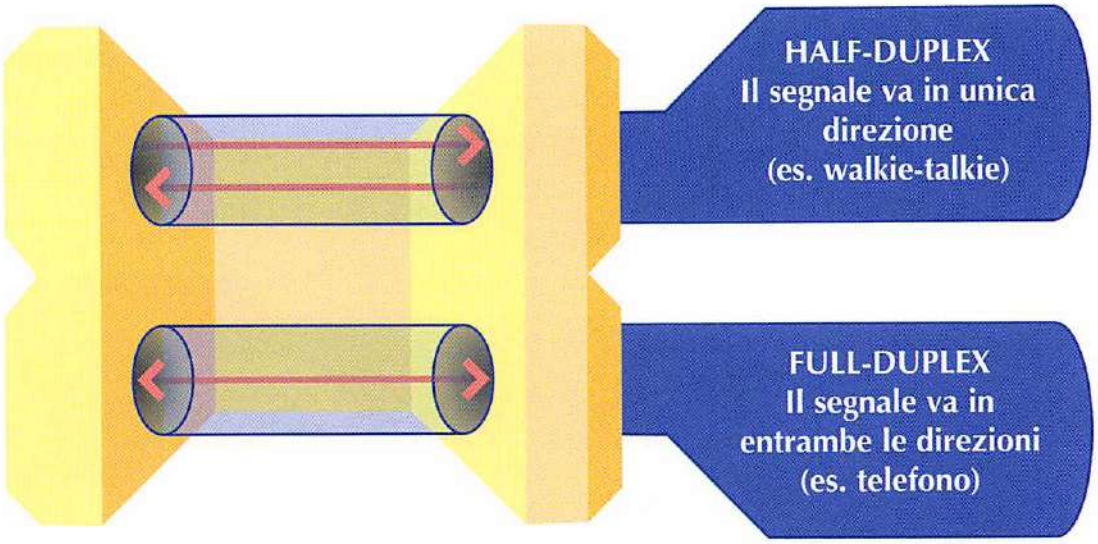
- Asincrono (Full-Duplex)
- Sincrono – Master (Half-Duplex)
- Sincrono – Slave (Half-Duplex)

Per configurare questo dispositivo si utilizzano dei registri specifici: TXSTA e RCSTA.

La comunicazione Full-Duplex è una comunicazione bidirezionale, si può trasmettere e ricevere allo stesso tempo. Al contrario, la comunicazione Half-Duplex è una comunicazione unidirezionale, ovvero avviene solamente in una direzione alla volta. Un telefono può essere un esempio di comunicazione Full-Duplex, dato che è possibile ascoltare e parlare contemporaneamente, può trasmettere e ricevere allo stesso tempo. Invece un walkie-tal-

Modo	Modo trasmissione	Caratteristica	Indirizzo dei dati
1	Asincrono		Full-Duplex
2	Sincrono	Master	Half-Duplex
3	Sincrono	Slave	Half-Duplex

Linee di trasmissione del modo SPI.



Comunicazione Full-Duplex o Half-Duplex.

Bit 7	CSRC	Selezione della sorgente del clock (ha importanza solamente nel modo sincrono)
	1	Modo master. Il clock si genera internamente tramite il BRG
	0	Modo slave. Il segnale di clock proviene dall'esterno
Bit 6	TX9	Abilitazione della trasmissione a 9 bit
	1	Seleziona la trasmissione a 9 bit
	0	Seleziona la trasmissione a 8 bit
Bit 5	TXEN	Bit di abilitazione della trasmissione
	1	Trasmissione abilitata
	0	Trasmissione disabilitata
Bit 4	SYNC	Selezione del modo della trasmissione
	1	Modo sincrono
	0	Modo asincrono
Bit 3	-----	Non implementato. Letto come 0
Bit 2	BRGH	Bit di selezione dell'alta velocità di trasmissione (si utilizza solamente nel modo asincrono,
	1	Alta velocità
	0	Bassa velocità
Bit 1	TRMT	Stato del registro di cambio di trasmissione (Transmit Shift Register)
	1	TSR vuoto
	0	TSR pieno
Bit 0	TX9D	Nono bit della trasmissione. Può essere il bit di parità

Configurazione dettagliata del registro TXSTA.

kie non permette questo, possiamo solamente trasmettere oppure ricevere, ma non entrambe le cose contemporaneamente, quindi siamo davanti a un chiaro esempio di comunicazione Half-Duplex.

La comunicazione si può eseguire in modo asincrono, ovvero non risponde a nessun segnale di clock, si trasmette il dato quando il trasmettitore lo desidera; oppure in modo sincrono, facendo riferimento a un segnale di clock che sincronizza l'invio e la ricezione dei dati. All'interno di questo modo è possibile differenziare tra master e slave. Il master sarà il dispositivo che stabilisce il segnale di clock e impone questo segnale ai rimanenti dispositivi che prendono parte alla comunicazione. Questi riceveranno il nome di slave.

I registri di configurazione

I registri con cui lavora la USART sono due: TXSTA e RCSTA. Il primo è il registro di controllo che determina lo stato della trasmissione e si trova all'indirizzo di memoria 98h nel banco 1. Il registro RCSTA è il registro di controllo che determina lo stato della ricezione. Si trova sul banco 0 all'indirizzo 18h.

Il registro TXSTA

Nella figura possiamo vedere la configurazione di questo registro insieme al dettaglio

Come configurare il modulo di comunicazione per effettuare una trasmissione in modo sincrono a 8 bit in cui il nostro microcontroller funzioni come "master"?

Dobbiamo configurare il registro TXSTA in modo che i bit successivi assumano i valori:

CSRC = TXEN = SYNC = 1

TX9 = 0

Il registro RCSPA deve avere il bit SPEN a 1 per abilitare la comunicazione.

Quale valore dovrebbe avere il registro TXSTA per una comunicazione asincrona a 9 bit e con velocità alta?

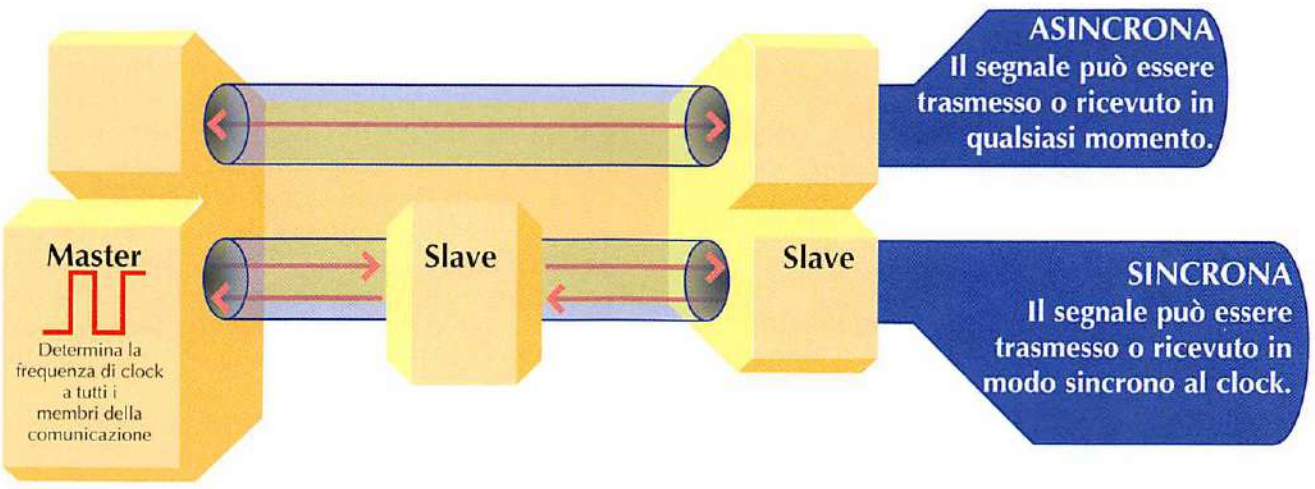
TX9 = TXEN = BRGH = 1

SYNC = 0

Esempio di configurazione della trasmissione.

dei bit che lo formano. Il bit più significativo, CSRC, è quello che determina la sorgente del clock (esterna o interna), quindi determina anche se il PIC lavorerà come slave (clock ricevuto da sorgente esterna) o master (interna). Il bit 6, TX9, serve per selezionare la trasmissione da 8 o 9 bit, e il bit successivo, TXEN, è il bit di abilitazione della trasmissione. Il bit SYNC determina se la trasmissione si realizza in modo sincrono o asincrono, quindi è determinante rispetto agli altri bit. Il bit successivo non è implementato fisicamente e quello seguente a esso, il bit BRGH, permetterà di selezionare la velocità di trasmissione in modo asincrono del generatore di frequenza interno del dispositivo. Il bit successivo, TRMT, si utilizza per segnalare lo stato del registro di cambio di trasmissione, e in ultimo, il bit 0 o bit TX9D si usa unicamente nella trasmissione a 9 bit, e questo nono bit può svolgere la funzione di bit di parità.

Nella tabella relativa a questo registro sono descritti uno a uno questi bit e i valori che assumono per ognuna delle loro funzioni.



Comunicazione asincrona e sincrona.

Registro RCSTA.

Bit 7	SPEN	Bit di abilitazione della porta seriale
	1	Abilitato (configurazione di RC7/RX/DT e RC6/TX/CK come pin della porta seriale)
	0	Porta seriale disabilitata
Bit 6	RX9	Bit di abilitazione della ricezione in comunicazione con 9 bit
	1	Seleziona ricezione da 9 bit
	0	Seleziona ricezione da 8 bit
Bit 5	SREN	Bit di abilitazione per un'unica ricezione (utilizzato solamente nel modo sincrono e master)
	1	Abilita un'unica ricezione
	0	Disabilita l'opzione
Bit 4	CREN	Bit di abilitazione per ricezione continua
	1	Abilitazione
	0	Disabilitazione
Bit 3	ADDEN	Bit di abilitazione di rilevamento di indirizzo (solo nel modo asincrono da 9 bit)
	1	Abilita il rilevamento di indirizzo, abilita l'interrupt e carica il dato del buffer di ricezione quando il bit RSR<8> è 1
	0	Disabilita il rilevamento di indirizzo, si ricevono tutti i bit e il no bit può essere utilizzato come bit di parità
Bit 2	FERR	Bit di rilevazione di errore nel formato del messaggio ricevuto
	1	Si è verificato un errore
	0	Non ci sono errori
Bit 1	OERR	Bit di rilevazione di errore di eccesso di tempo nella ricezione
	1	Si è verificato un errore
	0	Non ci sono errori
Bit 0	RX9D	Nono bit della ricezione; può essere il bit di parità

Configurazione dettagliata del registro RCSTA.

Il registro RCSTA

Il registro di stato della ricezione si compone dei bit che possiamo vedere nell'immagine della pagina precedente e che descriveremo di seguito. Il bit più significativo, bit SPEN, è un bit di permesso o abilitazione della comunicazione tramite la porta seriale (pin RC6 e RC7 del PIC). Il bit RX9 serve per selezionare il numero di bit che vogliamo ricevere, 8 o 9. Quello successivo, SREN, determina se si riceverà un'unica ricezione ed è utilizzato solamente nel modo sincrono quando il dispositivo funziona come master, e, nel caso sia selezionato, al termine della ricezione il bit passa automaticamente a 0.

Il bit successivo, CREN, è simile al precedente e abilita la ricezione dei dati in modo continuo. Invece di permettere un'unica ricezione, mediante questo bit si configura la trasmissione in modo che il microcontroller sia sempre pronto per ricevere dati.

Il bit ADDEN si utilizza solamente quando la ricezione è da 9 bit (RX9=1) e serve per abilitare la rilevazione di indirizzo, un interrupt e il carico del buffer di ricezione. I due bit successivi sono bit di indicazione di errore nella comunicazione. Così, il bit FERR indica se si è generato un errore nel formato del dato trasmesso e il bit OERR se si è generato un errore di eccesso di tempo nella ricezione. L'ultimo bit del registro, RX9D, sarà il nono bit della ricezione quando la comunicazione è da 9 bit.

Come per il registro di trasmissione, nella tabella della figura possiamo vedere i bit descritti in precedenza e le funzioni che eseguono in base al valore che assumono.

Come dovremo configurare il modulo di comunicazione per effettuare una ricezione continua in modo sincrono a 9 bit in cui il nostro microcontroller funzioni come "slave"?

Dobbiamo configurare il registro TXSTA in modo che SYNC = 1 e CSRC = 0

Il registro RCSPA deve avere configurati i suoi bit nel seguente modo:

SPEN = RX9 = CREN = 1

SREN = 0

Esempio di configurazione della ricezione.

Registro TXSTA.

Il modulo USART (II)

Il modulo USART è un dispositivo specializzato nella comunicazione sincrona e asincrona che accetta tre modi di funzionamento: trasmissione seriale asincrona (full duplex bidirezionale), trasmissione seriale sincrona modo master (half duplex unidirezionale) e trasmissione seriale sincrona modo slave (half duplex unidirezionale).

Generalità

Come abbiamo già visto questo dispositivo ha la funzione di convertire l'informazione parallela, che è quella gestita dalla CPU, in informazione seriale, gestita dalle periferiche. Normalmente la CPU comunica con un bus parallelo da otto bit, e con dispositivi periferici seriali tramite una linea di trasmissione e un'altra di ricezione, come mostrato nella figura.

L'USART si può trovare come circuito integrato specifico, ad esempio USART 8251 di Intel, o può essere implementato fisicamente all'interno di un dispositivo più potente, come nel caso di alcuni microcontroller della famiglia PIC16F87X e PIC16C7X, permettendo così la trasmissione dell'informazione seriale in modo sincrono e asincrono.

Funzionamento dell'USART

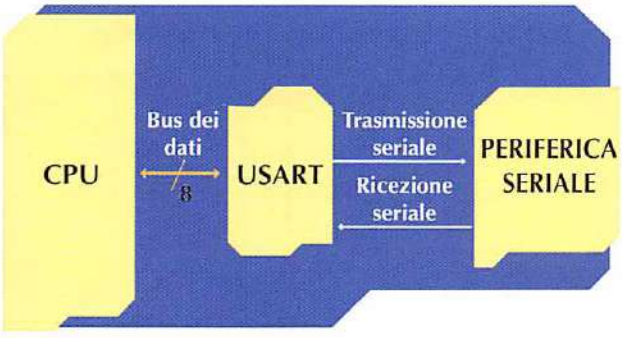
Il modulo USART integrato nei PIC16F87X si può dividere in quattro blocchi:

- Registro di trasmissione asincrono TXREG.
- Registro di ricezione asincrono RCREG.
- Generatore di Baud.
- Registro di spostamento TSR o RSR.

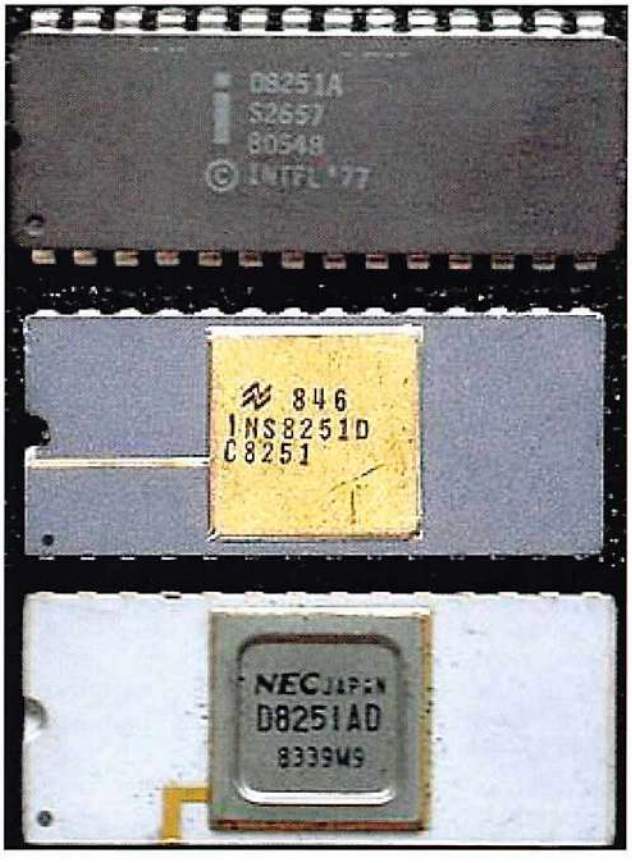
I bit ricevuti dalla linea RC7/RX passano sul registro di spostamento RSR fino a quando non è pieno.

A questo punto vengono depositati sul registro di ricezione RCREG. Quando si vuole trasmettere un byte si carica sul registro TXREG e poi sul registro di spostamento, che manda i bit uno a uno sul pin RC6/TX al ritmo degli impulsi di clock.

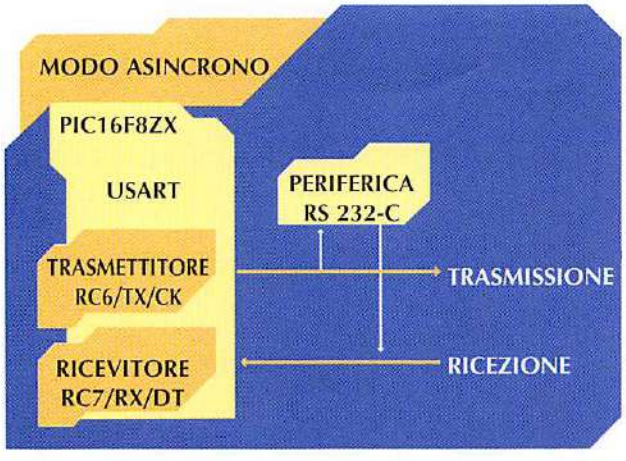
La frequenza del generatore di Baud dipende dal valore X caricato sul registro PSBRG e



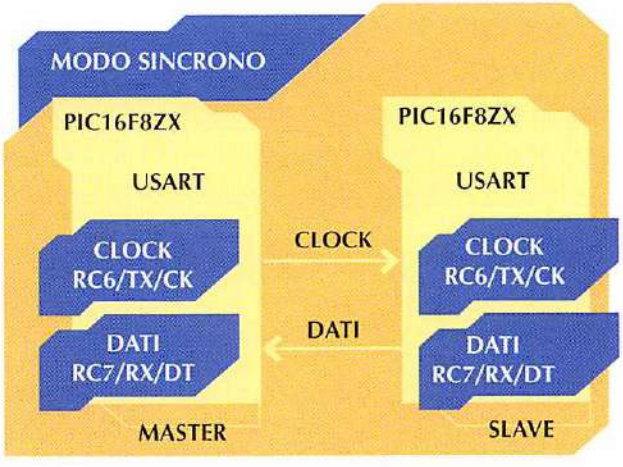
L'USART trasforma l'informazione seriale in parallelo e viceversa.



Diversi circuiti integrati contengono l'USART: I8251, INS8251 e UPD8251.



Implementazione della comunicazione seriale asincrona e sincrona nei PIC16F87X.



Blocchi principali del modulo USART.

dal bit BRGH del registro TXSTA. Se BRGH = 0, si lavora a bassa velocità e la costante K della formula vale 64. Quando BRGH = 1, il valore di K è uguale a 16.

$$\text{Frequenza} = \frac{\text{Fosc}}{\text{K-(X+1)}}$$

USART come trasmettitore e ricevitore asincrono

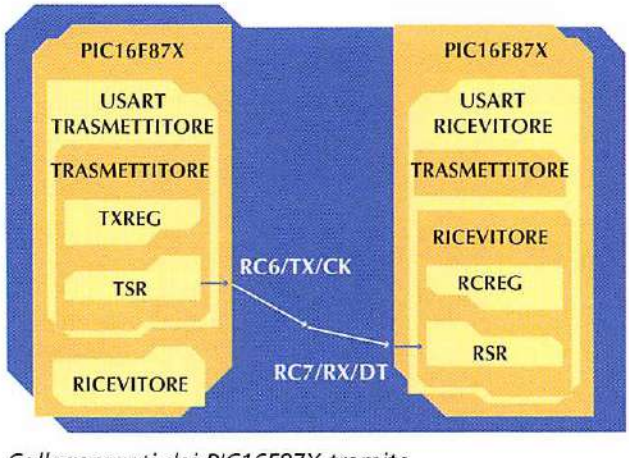
La comunicazione in questo caso è bidirezionale. La linea RC6/Tx/CK funziona come linea di trasmissione e la RC7/Rx/DT come linea di ricezione. Il dato inviato tramite il trasmettitore si deposita sul registro TXREG e successivamente viene traslato sul registro di spostamento TSR, che spedisce i bit in modo sequenziale. Inoltre, prima di spedirli, genera il bit START e, quando ha spedito tutto, il bit di STOP.

USART ricevitore, riceve i bit ed elimina il bit di START e quello di STOP, memorizzando l'informazione sul registro di spostamento RSR. Quando questo è pieno, sposta l'informazione sul registro RCREG.

I bit di controllo che sono contenuti sul registro di stato del trasmettitore TXSTA li abbiamo visti nel capitolo precedente, così come i bit del registro RCSTA che eseguono il controllo nel modo di ricezione. Mediante questi registri si controlla il funzionamento della comunicazione, stabilendo il modo di quest'ultima e le sue diverse opzioni.

USART come trasmettitore e ricevitore sincrono

Questo tipo di comunicazione è unidirezionale. Si utilizza una sola linea per i dati ed è la linea RC7/Rx/DT. Nel modo master il segnale di



Collegamenti dei PIC16F87X tramite i loro moduli USART.

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	–	BRGH	TMRT	TX9D
bit7							bit0

RCSTA: RECEIVE STATUS AND CONTROL REGISTER (18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit7							bit0

Registri di controllo del trasmettitore e del ricevitore nel modo SCI.

clock esce sulla linea RC6/Tx/CK, e nel modo slave entra da essa. In entrambi i modi i dati possono essere da 8 o 9 bit, ed è possibile utilizzare il nono bit come bit di parità, trasmettendolo o ricevendolo come bit 0 di RXSTA e/o RCSTA.

Il registro specifico TXSTA funziona come registro di stato e controllo del trasmettitore e RCSTA esegue la stessa funzione per il ricevitore, così come succede per il modo asincrono.

I Baud, come abbiamo visto in precedenza, vengono definiti dal valore caricato sul registro SPBRG e il bit BRGH del registro TXSTA, col quale si può stabilire la velocità della trasmissione in modo asincrono (1 = velocità alta; 0 = velocità bassa).

Nella figura sottostante possiamo vedere

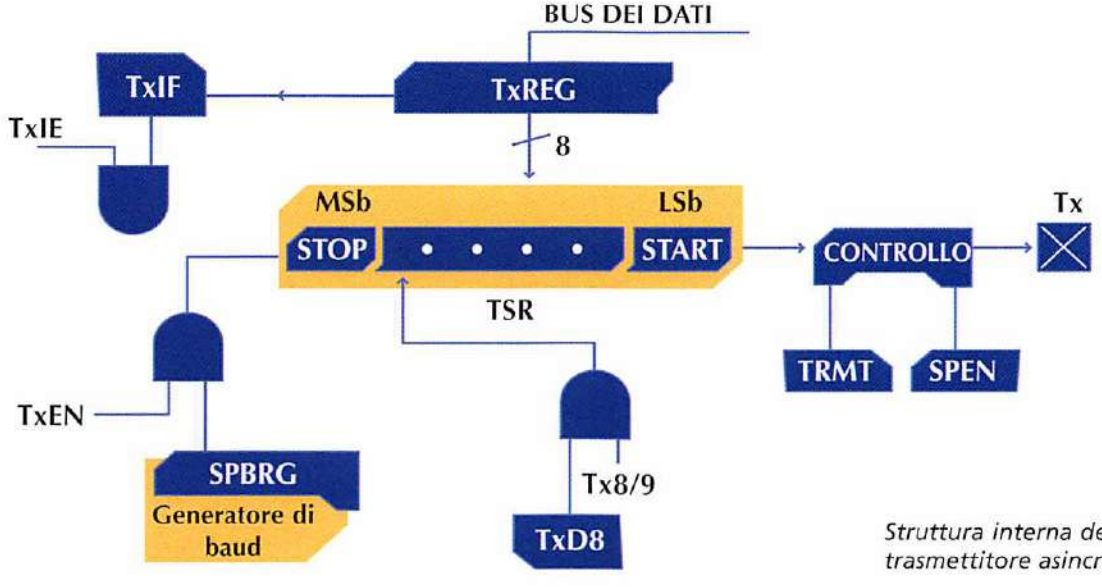
$$BAUD = \frac{Fosc}{K-(X+1)}$$

K = 4 nel modo sincrono
K = 16 nel modo asincrono ad alta velocità
K = 64 nel modo asincrono a bassa velocità
X = valore caricato nel registro SPBRG

$$BAUD = \frac{Fosc}{K \cdot BAUDI} - 1$$

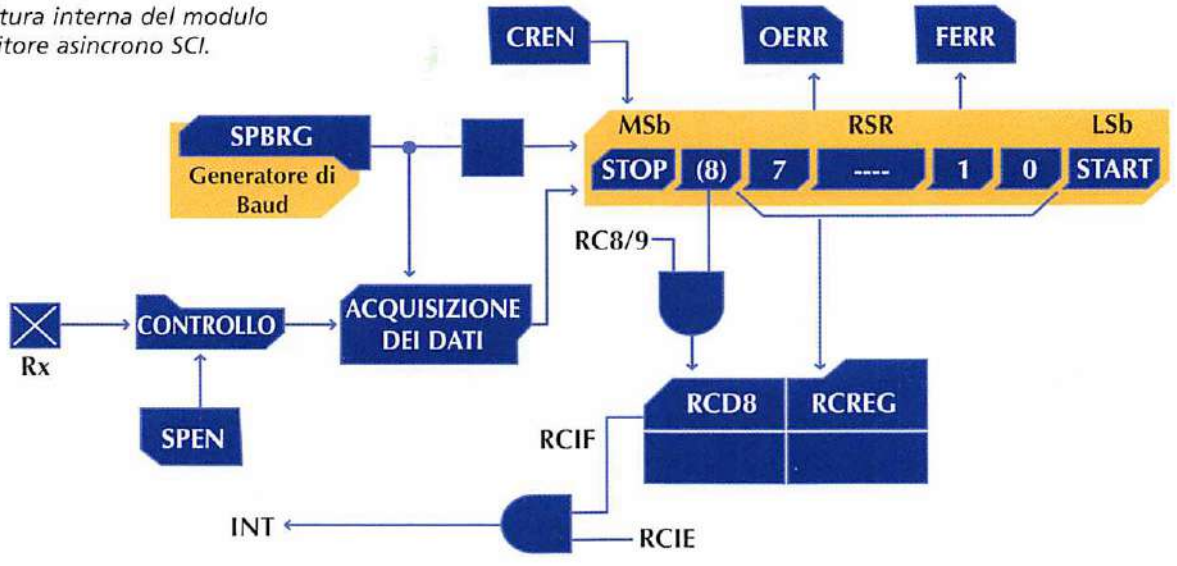
Velocità della trasmissione.

l'architettura interna del modulo trasmettitore asincrono. Mediante la programmazione dei bit del registro di controllo TXSTA e RCSTA si configura il modo lavoro. Nel fascicolo precedente abbiamo analizzato il registro e spiegato



Struttura interna del modulo SCI, trasmettitore asincrono.

Struttura interna del modulo ricevitore asincrono SCI.



la funzione di ognuno dei suoi bit, e ora, per capire come viene eseguita la trasmissione in questo modo, è necessario fare riferimento a quella descrizione. SPEN configura i pin RC7/Rx e RC6/Tx come linee di comunicazione seriale, attivando il trasmettitore con il bit TxEN. Il dato da trasmettere si carica su TxREG e poi si passa al registro di trasmissione TSR, quando è stato trasmesso il bit di stop del dato precedente. In questo momento si attiva il flag TxIF e, se il bit di abilitazione è attivo, si genera un interrupt. Attivando Tx9 si inserisce il nono bit memorizzato sul bit 0 di TXSTA.

Nella figura in alto è riportata l'architettura interna del modulo ricevitore asincrono, che si attiva col bit CREN. Il dato si riceve su RSR e quando si completa si passa sul registro RCREG per la sua successiva lettura, attivando il flag RCIF e, come per la trasmissione, se sono stati abilitati gli interrupt, ne verrà generato uno.

Se si attiva il bit RC9 di RCSTA il nono bit si deposita sul bit 0 di RCSTA. I bit OERR e FERR indicano rispettivamente errori di overflow e di trama.

Nel modo sincrono SCI lavora in half duplex, non potendo ricevere e trasmettere contemporaneamente. Il segnale di clock si invia al trasmettitore (master) insieme ai dati. Il funzionamento in questo modo è molto simile a quello spiegato per il modo asincrono, bisogna solamente selezionare questo modo lavoro caricando correttamente i registri di controllo TXSTA e RCSTA.

Esecuzione pratica

Quando lavoriamo con questo tipo di comunicazione dobbiamo programmare in modo adeguato i registri del nostro microcontroller. Il programmatore deve aver ben chiari i diversi tipi di comunicazione che ha a disposizione, con i quali deve o vuole lavorare e come si programma questo modo di comunicazione nel microcontroller, dato che fatto questo è il chip stesso che esegue tutto il processo di comunicazione.

Negli esercizi che realizzeremo lavoreremo con la comunicazione del PIC e capiremo facilmente quanto studiato negli ultimi capitoli.

Con la comunicazione appena vista consideriamo terminata la teoria del PIC16F870, e passiamo ad applicarla tramite gli esercizi. Il lettore potrà continuare ad approfondire le questioni spiegate o il mondo dei microcontroller attraverso la grande varietà di modelli che esistono sul mercato.

Calcolare il valore di X da caricare sul registro SPBRG se si desidera trasmettere a 9.600 Baud in modo asincrono con la frequenza di funzionamento del microcontroller a 4 MHz.

$9.600 = (4 \cdot 10^6) / [16 \cdot (X+1)] \rightarrow X = 25.04$

Se carichiamo 25 la frequenza esatta che si ottiene è:

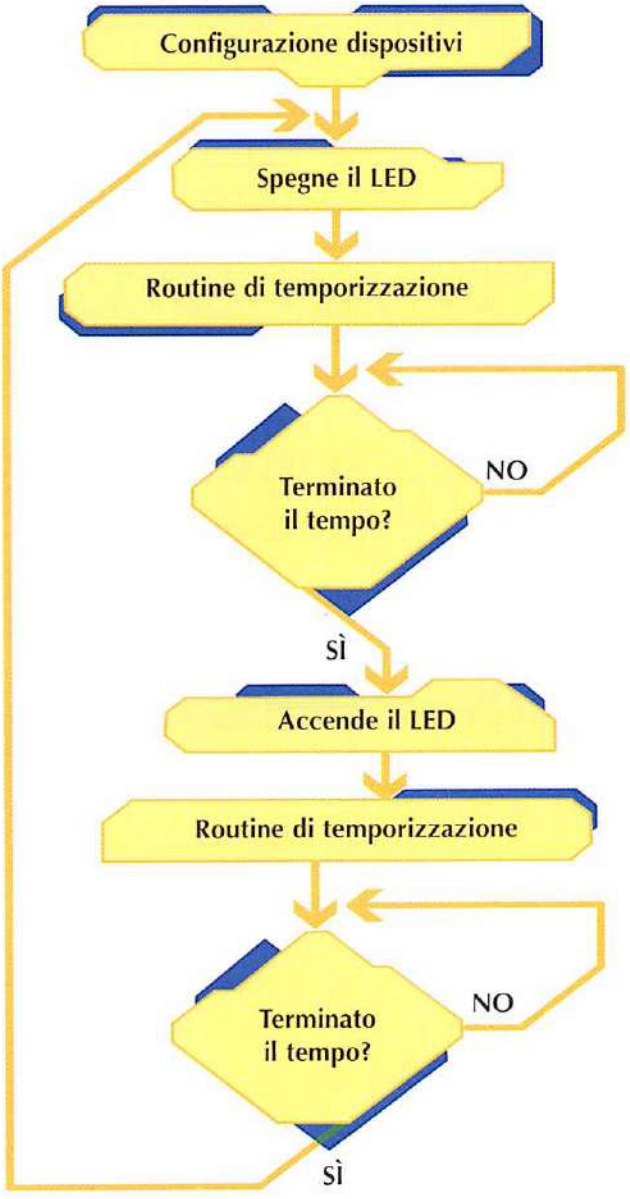
$Baud = (4 \cdot 10^6) / [16 \cdot (25+1)] = 9615,38$ Il che suppone un errore dello 0,16%

Calcolare la soluzione lavorando in bassa velocità (K=64)

Calcolo del valore da caricare per stabilire la comunicazione a una velocità

Esercizi di ripasso

Lavoreremo con i nuovi dispositivi del microcontroller che abbiamo studiato, quindi faremo una serie di esercizi sul laboratorio che simuleranno problemi reali che potremmo trovare nel mondo della programmazione. Prima di iniziare a lavorare con questi dispositivi faremo alcuni esercizi che miglioreranno le nostre conoscenze e ci prepareranno per risolvere gli enunciati più complessi.



Organigramma corrispondente all'enunciato 1.

Enunciato 1: TMR0

Il primo esercizio si basa sulla funzione del Timer 0 (TMR0). Si tratta di sviluppare un programma che faccia lampeggiare un LED collegato a RB2 ogni 65,28 ms, utilizzando un range del divisore di 1:256 e preveda l'utilizzo del TMR0.

Per lo sviluppo di questo programma dobbiamo disporre unicamente di un'uscita per collegare il diodo LED e lavorare con il Timer 0. Nell'organigramma della figura si può vedere la semplicità del nostro obiettivo. Configureremo i dispositivi, che in questo caso sono solamente la porta B per l'uscita del LED e il predivisore di frequenza per il temporizzatore. Inizieremo poi il programma spegnendo il LED e chiamando la routine di temporizzazione, che ritornerà al programma principale dopo che sarà trascorso il tempo desiderato, accenderemo il LED e torneremo a utilizzare la routine di temporizzazione. Ripeteremo il ciclo spegnendo nuovamente il LED e ripetendo i passaggi precedenti.

Codice che risolve l'enunciato 1

Per sviluppare il codice dobbiamo anzitutto calcolare il predivisore di frequenza che vogliamo utilizzare per ottenere il tempo desiderato, e a questo scopo utilizzeremo la for-

Tempo = 4·(1/Fosc)Valore·Predivisore

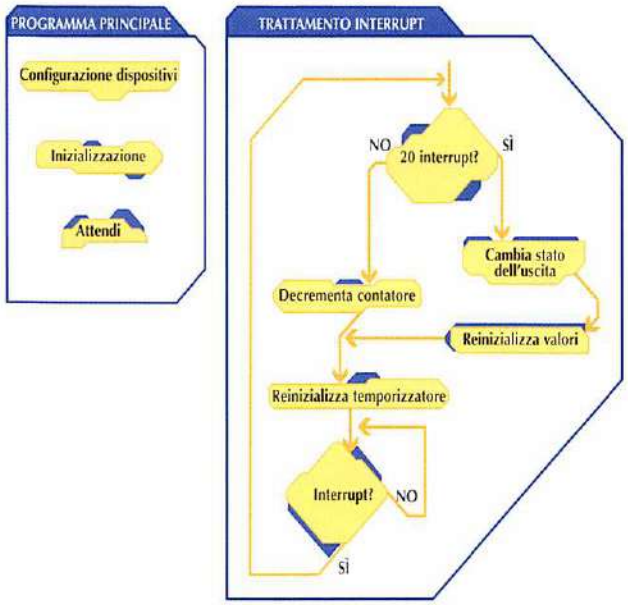
Quindi:

65,28 ms = 4·(1/410^6) Valore 256 —> Valore = 255

Formula per calcolare il tempo del temporizzatore.

```
a1 - Blocco note
File Modifica Formato Visualizza ?
;                    ESERCIZIO 1: TMR0
;                    -----------------

;Gestione del timer 0 (TMR0) con il PIC16F870.

;Programma che fa lampeggiare il LED collegato a RB2 ogni 65,28 ms utilizzando
;il TMR0 con un range del predivisore di 1:256

;Il calcolo del valore da contare è:
;65,28ms=4*(1/4MHz)*valore*256 ---> valore=255

        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni

        ORG     0
        GOTO    INIZIO
        ORG     5

;Programma principale.

INIZIO  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        clrf    TRISB           ;Porta B uscite
        movlw   b'11010111'     ;Predivisore 256
        movwf   OPTION_REG
        bcf     STATUS,RP0      ;Torniamo al banco 0

LAMPEG  bcf     PORTB,2         ;Spegnamo il terzo LED (RB2)
        call    RITARDO         ;Saltiamo alla routine di ritardo
        bsf     PORTB,2         ;Accendiamo il terzo LED
        call    RITARDO
        goto    LAMPEG

RITARDO movlw   b'00000001'     ;Valore da contare 255
        movwf   TMR0            ;(complemento a 2)
        btfss   INTCON,T0IF
        goto    $-1             ;Torna all'istruzione precedente
        bcf     INTCON,T0IF
        return

        END
```

Codice che risolve l'enunciato 1.



Organigramma corrispondente all'enunciato 2.

mula della figura sottostante. In seguito intesteremo il nostro programma con i commenti pertinenti e le direttive appropriate in cui vengono definiti il microprocessore e le librerie da utilizzare. Configureremo la porta B come uscita e caricheremo sul registro OPTION_REG il valore del predivisore di frequenza. Il programma principale sarà un ciclo in cui si cancella il bit corrispondente al terminale RB2, si chiama la temporizzazione, si attiva nuovamente il terminale e si ritorna alla temporizzazione. Nella routine di temporizzazione caricheremo solamente il valore che abbiamo calcolato, e attenderemo che si compia la condizione che indica che il tempo è trascorso per ritornare al programma principale.

Enunciato 2: TMR0 e interrupt

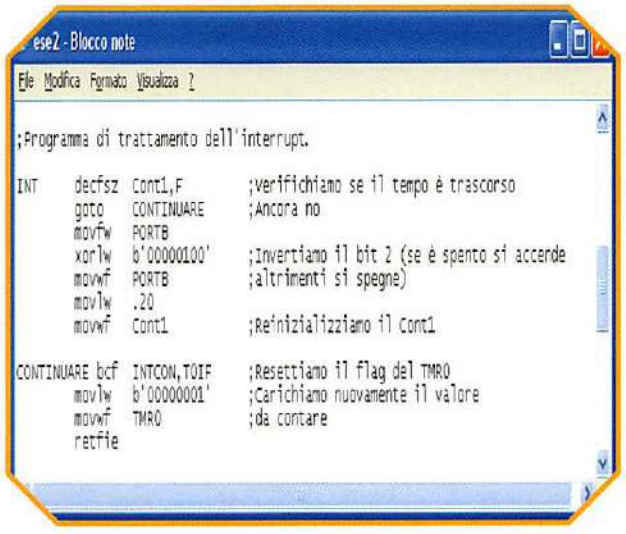
In questo secondo esercizio combineremo l'utilizzo del TMR0 con quello degli interrupt, provocando un interrupt per overflow del TMR0. Si tratta di sviluppare un programma

```
ese2 - Blocco note
File Modifica Formato Visualizza ?
;                    ESERCIZIO 2: Interrupt e TMR0
;
;Interrupt per overflow del TMR0 con il PIC16F870.

;Programma che fa lampeggiare il LED collegato a RB2 ogni 1,3 s mediante
;interrupt del TMR0 ogni 65,28 ms facendolo illuminare o spegnere solamente
;ogni 20 interrupt, utilizzando un range del predivisore di 1:256.

;Il calcolo del valore da contare è: 65,28ms=4*(1/4MHz)*valore*256 --> valore=255

        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni

Cont1   equ     0x20

        ORG     0
        GOTO    INIZIO
        ORG     4
        GOTO    INT
        ORG     5
```

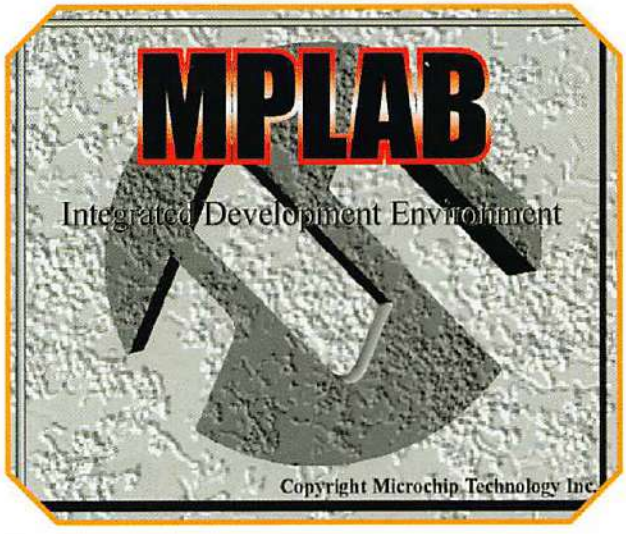
Codice nell'intestazione dell'enunciato 2.

```
ese2 - Blocco note
File Modifica Formato Visualizza ?

;Programma principale.

INIZIO  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        clrf    TRISB           ;Porta B uscite
        movlw   b'11010111'
        movwf   OPTION_REG
        bcf     STATUS,RP0
        movlw   b'10100000'     ;Attiviamo gli interrupt
        movwf   INTCON          ;per il TMR0
        movlw   b'00000001'
        movwf   TMR0            ;Inizializziamo il conteggio
        movlw   .20
        movwf   Cont1

NULLA   clrwdt
        goto    NULLA           ;Ciclo che non fa nulla sino
                                ;a quando non si genera un interrupt.
        END
```

Programma principale.



Programma di trattamento dell'interrupt.

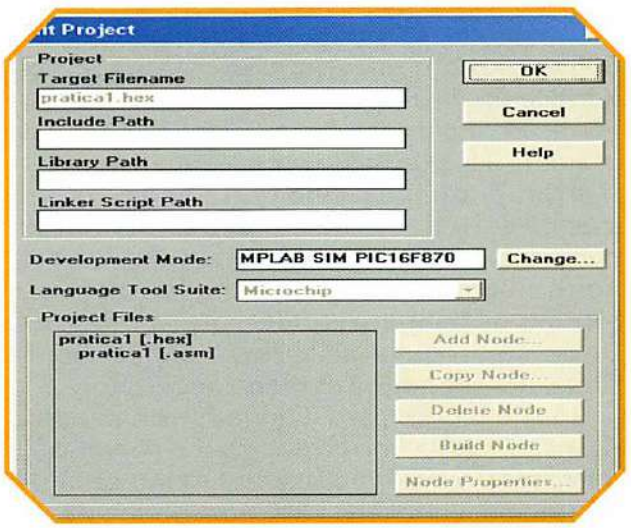


Torniamo a lavorare con MPLAB.

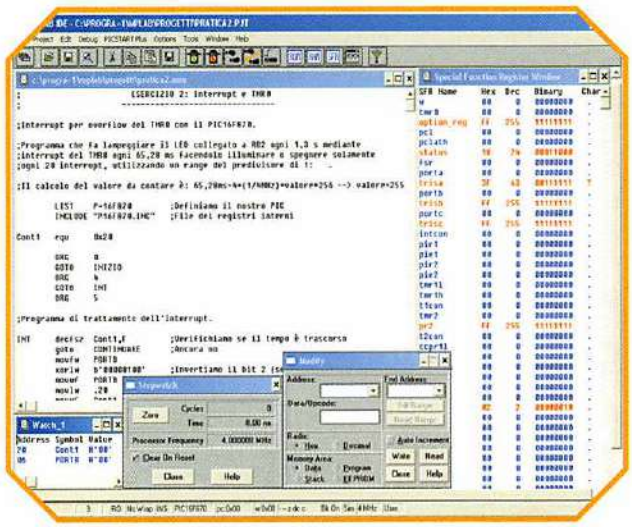
che faccia lampeggiare un LED collegato a RB2 ogni 1,3 s. A questo scopo utilizzeremo l'interrupt del TMR0 ogni 65,28 ms, in modo che si illumini o si spenga solamente dopo 20 interrupt, utilizzando un range del divisore di 1:256.

Questo esercizio utilizza gli interrupt, oltre ai dispositivi dell'esercizio precedente, quindi il modo con cui abbiamo previsto di risolverlo è molto diverso dal precedente. Nel programma principale si configurano i dispositivi: porta B, predivisore di frequenza e interrupt. In seguito inizializzeremo i valori del conteggio e in ultimo, entreremo nel ciclo in cui attenderemo che si generino gli interrupt.

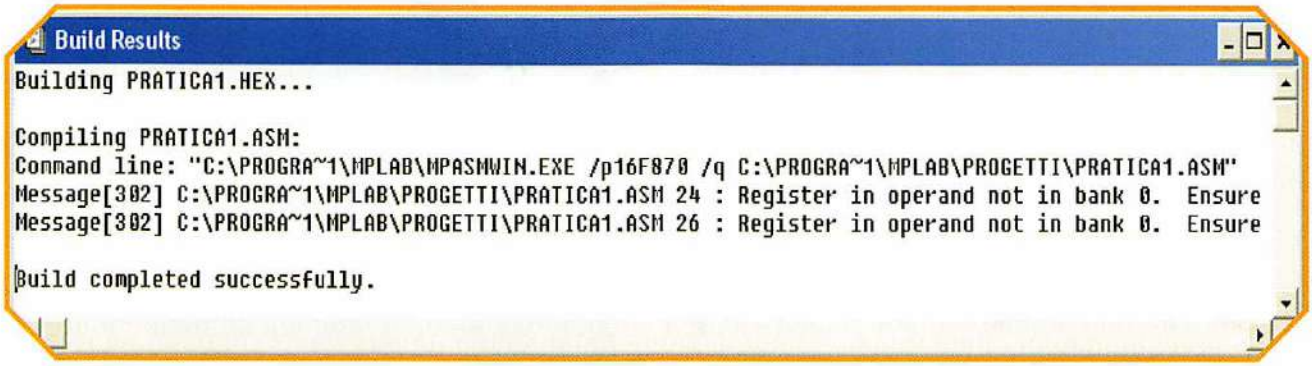
Nel programma per il trattamento degli interrupt risiede la "difficoltà" dell'esercizio. La prima cosa da fare è verificare che sia trascorso il tempo totale desiderato, questo avviene quando si producono 20 interrupt del temporizzatore. Nel caso in cui la condizione non sia soddisfatta, ridurremo il contatore e attenderemo un nuovo interrupt; se sono stati generati 20 interrupt cambieremo lo stato dell'uscita e reinizializzeremo i valori del conteggio. Gli organigrammi presentati rispondono al funzionamento spiegato.



Per compilare i vostri codici dovrete utilizzare MPLAB e creare un progetto.



Videata tipica di simulazione con MPLAB.



Risultato della compilazione.

Codice che risolve l'enunciato 2

Dato che in questo esercizio dobbiamo utilizzare una variabile come contatore per i 20 interrupt, definiremo quest'ultima all'inizio del programma, riservando a essa un indirizzo nella memoria. Nelle direttive in cui definiamo dove si trova il codice del programma, dobbiamo inserire la routine di interrupt (ORG 4).

Nelle immagini della pagina precedente possiamo vedere come rimane l'intestazione del programma. Avendo ben chiare queste particolarità, lo sviluppo del codice non risulta molto complicato. Nel programma principale si configurano i dispositivi da utilizzare e se ne definiscono i valori, sia del predivisore che del registro di configurazione degli interrupt, per poter così abilitare ciò che ci interessa, ovvero il TMR0. Inizializzeremo i valori di conteggio e attenderemo in un ciclo infinito senza fare nulla. Nel programma per il trattamento degli interrupt si contano questi ultimi e se si arriva a 20, si cambia il valore dell'uscita mediante l'istruzione XOR, reinizializzando successivamente i valori del conteggio. Se non si è ancora arrivati a 20 caricheremo nuovamente il temporizzatore con il suo valore di conteggio e attenderemo che finisca e provochi un nuovo interrupt.

Compilazione

Per compilare i programmi presentati o quelli fatti da voi stessi utilizzeremo MPLAB. Ricordate che è necessario creare un progetto, ad esempio con il nome di pratica1, a cui aggiungere in seguito il codice in assembler associato. Aprite il file che avete associato al progetto per poterlo visualizzare e selezionate Build All dal menù Project o premete Ctrl+F10. In questo modo verrà compilato il codice e potrete verificare se è stato commesso qualche errore durante il confezionamento del programma. Eseguite questi passi con i due programmi realizzati e verificate che non contengano errori. Nel caso in cui la compilazione non avvenga con successo, tenendo aperto il codice sul display, fate un doppio click con il mouse sulla linea che indica l'errore nella compilazione, e automaticamente il software si posizionerà sulla linea di codice che contiene l'errore.

Quando la compilazione viene eseguita con successo si creano i file in codice macchina (.hex) che successivamente si scriveranno sul microcontroller.

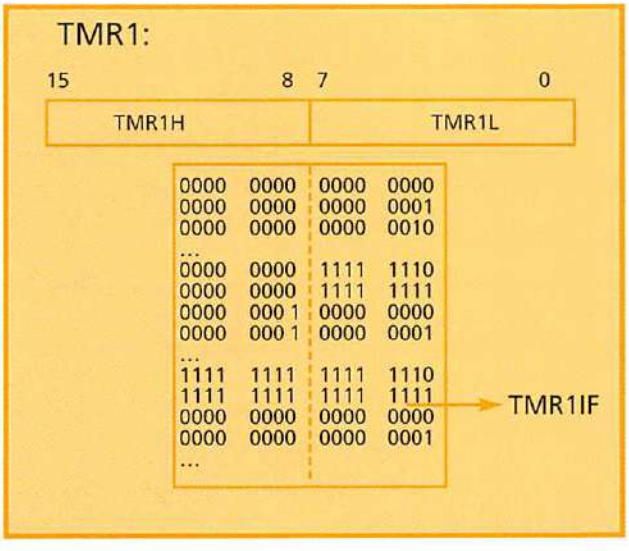
Simulazione

Come abbiamo già visto negli esercizi precedenti la simulazione risulta molto comoda per verificare il corretto funzionamento dei programmi. Il fatto che un programma non generi errori di compilazione non significa che risponda correttamente ai requisiti del funzionamento. I due programmi presentati non sono facili da simulare, dato che lavorano con routine di temporizzazione o con interrupt. Dovremo eseguire delle forzature per verificare che rispondano in modo adeguato.

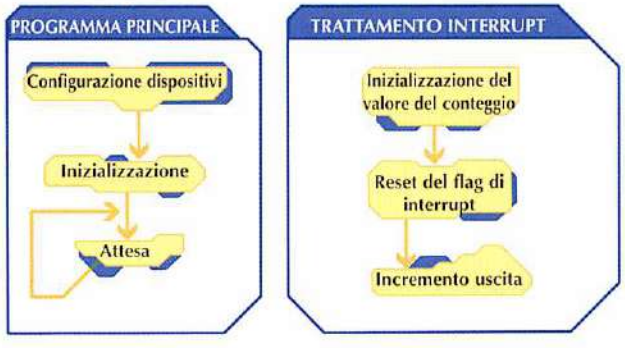
Dopo aver simulato il buon funzionamento del programma non ci rimarrà che scriverlo sul microcontroller e realizzare il montaggio del circuito per verificare che il funzionamento sia realmente quello desiderato.

Esercizio 9: il timer 1 (TMR1), il programma

In questo esercizio lavoreremo con il secondo temporizzatore, il TMR1. Tramite gli esercizi vogliamo farvi applicare le conoscenze acquisite e che impariate a maneggiare i potenti dispositivi del microcontroller. Sul secondo CD, il nome del file su cui si trova la soluzione di questo esercizio è "ese9.asm".



Il TMR1 è un temporizzatore/contatore da 16 bit che quando funziona come temporizzatore incrementa il suo valore ogni ciclo di istruzione (Fosc/4).



Organigramma delle applicazioni.

Introduzione

Il TMR1 può funzionare in tre modi diversi: temporizzatore, contatore sincrono e contatore asincrono. Dato che come contatore è necessario il collegamento di un oscillatore o di un clock esterno (secondo che sia sincrono o asincrono) proponiamo un esercizio in cui il TMR1 funziona come temporizzatore. In questo modo il suo funzionamento è molto simile a quello del TMR0, con la differenza che il range del predivisore in questo caso è: 1/1, 1/2, 1/4, 1/8.

Enunciato

Si vuole accendere sequenzialmente i LED di uno dei display a 7 segmenti collegato alla porta B ogni mezzo secondo.

Per realizzare questo esercizio abbiamo bisogno di tutte le linee della porta B per utilizzarle come uscite e anche di ripassare i nostri concetti riguardo al temporizzatore TMR1 e, nuovamente, quelli degli interrupt.

Gli organigrammi che rispondono a quanto richiesto nell'enunciato sono riportati nell'immagine a fianco. In uno si risolve il programma principale, in cui si configurano i dispositivi che vogliamo utilizzare: porta B, interrupt e temporizzatore, e successivamente entreremo in un ciclo senza fare nulla in attesa che si generi l'interrupt. Nella subroutine di trattamento degli interrupt caricheremo il valore del conteggio del temporizzatore, resettere-

```
                        ESERCIZIO: TMR1
                        ---------------
;
;Il programma muove in modo sequenziale i LED del 7 segmenti ogni mezzo secondo (500 ms).

;Si suppone che il TMR1 lavori con un prescaler di 1/8, a 4 MHz.
;Calcoliamo il valore che dobbiamo caricare sul TMR1:
;       500ms=4*(1/4MHz)*valore*8 -->   valore(dec)=62500
;       valore(bin)=11110100 00100100
                    -------- --------
                     TMR1H    TMR1L
```

Nel confezionamento di un programma sono necessari i commenti.

```
LIST    P=16F870          ;Definiamo il nostro PIC
INCLUDE "P16F870.INC"     ;File dei registri interni


ORG     0
GOTO    INIZIO
ORG     4
GOTO    INT
ORG     5
```

Inizieremo a sviluppare il codice con l'intestazione a cui siamo abituati.

```
;Programma principale.

INIZIO  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        clrf    TRISB           ;Porta B uscite
        bsf     PIE1,TMR1IE     ;Abilitiamo l'interrupt del TMR1 (TMR1IE)
        bcf     STATUS,RP0      ;Torniamo al banco 0
        movlw   b'00110001'
        movwf   T1CON           ;Attiviamo il TMR1 e impostiamo il divisore da 1/8
        movlw   b'11000000'
        movwf   INTCON          ;Abilitiamo gli interrupt: globali e per il PEIE

NULLA   clrwdt                  ;Ciclo che non fa nulla
        goto    NULLA
```

Codice del programma principale.

```
;Programma di trattamento dell'interrupt

INT     movlw   b'11011100'     ;c2(00100100)
        movwf   TMR1L
        movlw   b'00001100'     ;c2(11110100)
        movwf   TMR1H           ;Carichiamo il TMR1 con il complemento di 31250 in hex.
        bcf     PIR1,TMR1IF     ;Impostiamo a zero il flag TMR1IF
        incf    PORTB,f         ;Incrementiamo il valore della porta B che riflette il
                                ;risultato sul display a 7 segmenti
        retfie                  ;Ritorno da Interrupt
```

Codice della subroutine di trattamento dell'interrupt.

mo il flag di indicazione di interrupt e incrementeremo l'uscita per accendere così i LED del display.

Codice

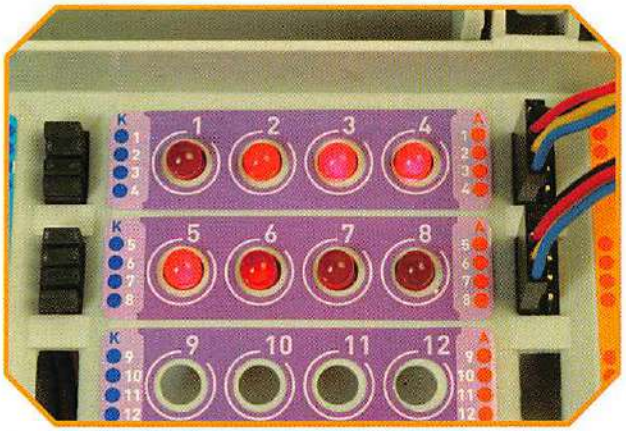
Il grado di complessità di questo esercizio non è molto elevato. L'unica difficoltà risiede nella gestione degli interrupt e del temporizzatore.

Inizio

Per iniziare con il programma scrivete i commenti che definiscono la funzionalità dello stesso. Vi consigliamo di scrivere tutto ciò che è necessario a far sì che in seguito risulti più facile risolvere il problema o interpretare il programma voi stessi o un futuro fruitore. Siamo pronti per iniziare il programma e, come sempre, inizieremo dalla definizione del microcontroller, dalla libreria che contiene la definizione dei dispositivi interni e dalle direttive del programma in cui definiremo gli indirizzi della memoria dove dovrà risiedere il nostro programma.

Programma principale

Secondo quanto previsto nell'organigramma, nel programma principale dovremo configurare i dispositivi. La porta B, quindi, si configura come uscita, abiliteremo gli interrupt del temporizzatore TMR1 impostando a 1 il bit TMR1IE del registro PIE1, abiliteremo questo temporizzatore indicando il predivisore di frequenza da utilizzare e abiliteremo gli interrupt globali necessari per qualsiasi interrupt si generi. Con questo passaggio abbiamo realizzato la configurazione di cui abbiamo bisogno per risolvere l'esercizio. L'ordine dei pas-



È molto facile imitare alcuni effetti che tempo fa risultavano sorprendenti, come il frontale di KIT (Supercar).

saggi non è determinante, ma dovrà essere sempre eseguito in modo da non richiedere inutili cambi di banco (in questo modo risparmieremo istruzioni), e dovrà avere una forma ordinata. All'inizio di questo blocco inseriremo un'istruzione di cancellazione della porta di uscita per assicurarci che quando caricheremo il programma sul microcontroller parta con tutti i LED spenti.

Infine entreremo in un ciclo infinito senza fare nulla attendendo che si generi un interrupt del temporizzatore.

Trattamento dell'interrupt

È in questa subroutine dove si risolve realmente il programma e dove il progettista può giocare con le diverse soluzioni o inserire le eventuali aggiunte. La prima cosa da fare è caricare il valore di conteggio del temporizzatore, il quale si ottiene in funzione del tempo che vogliamo controllare. Dobbiamo anche resettare o impostare a 0 il bit del flag di interrupt TMR1IF. In ultimo, definiremo ciò che vogliamo fare dopo che è trascorso il tempo, in questo caso incrementeremo il valore della porta B, in modo che questo valore si rifletta sui LED del display. A questo scopo potremo utilizzare diverse istruzioni e osservare i risultati relativi. Se vogliamo far accendere solamente un LED alla volta invece di accenderli uno dopo l'altro in sequenza, dovremo caricare un 1 e utilizzare le istruzioni di rotazione. Potremo utilizzare i LED della matrice invece di quelli del display, e simulare l'effetto di "Supercar", in modo che quando l'ultimo LED sarà acceso si realizzi la sequenza in senso inverso.

Non bisogna dimenticare di terminare il programma con la direttiva END. Nell'immagine in basso possiamo vedere il codice completo del programma.

Compilazione

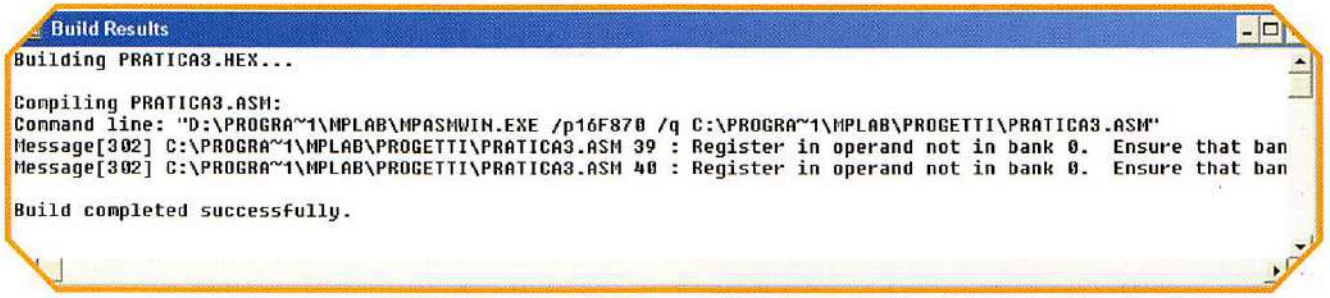
Realizzato il programma, dobbiamo compilare il nostro codice per poter così correggere gli errori sintattici che potremmo aver commesso. È normale che, dopo aver progettato il codice, si trovi qualche errore durante la compilazione, a questo scopo MPLAB ci offre un potente compilatore.

Apriremo MPLAB, creeremo un nuovo progetto a cui aggiungeremo il nostro codice che sarà stato precedentemente salvato nella directory dei progetti di MPLAB. Se selezioneremo Project → Build All otterremo il risultato riportato nell'immagine della figura della pagina successiva. Il nostro codice non riporta nessun errore, questo significa che voi non dovrete far altro che correggere eventualmente qualche piccolo errore del programma che avrete realizzato. Il fatto di dover correggere qualche piccolo errore è positivo perché aiuta a mettere a punto e a chiarire concetti che potrebbero non essere stati bene assimilati.

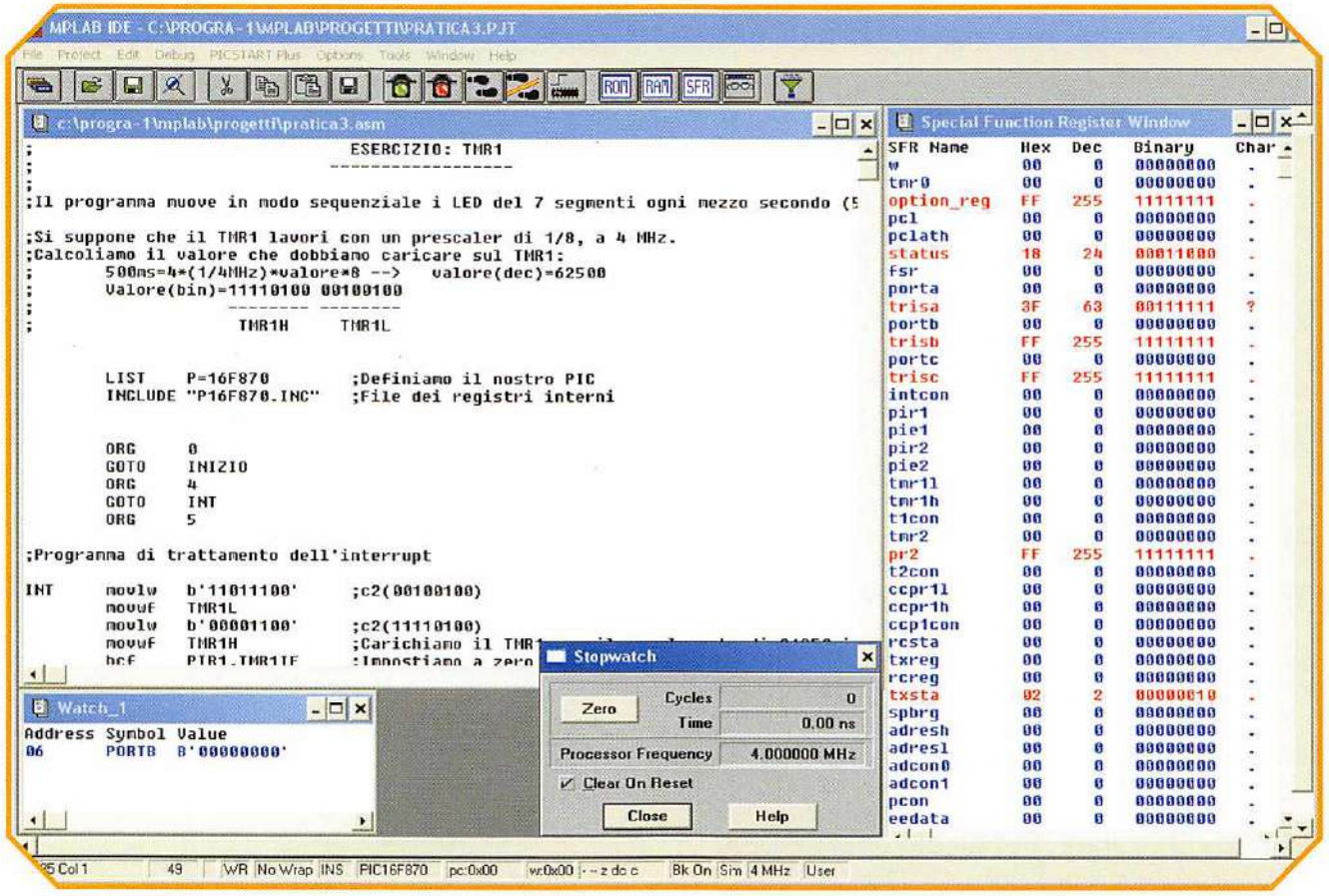
Dopo aver ripulito il programma dagli errori e aver generato il file in codice macchina necessario per caricare il PIC, potremo simulare l'applicazione. La simulazione in un programma che lavora con interrupt non è molto semplice, ma vi consigliamo comunque di giocare

```
                        ESERCIZIO: TMR1
                        ---------------
;Il programma muove in modo sequenziale 1 LED dei 7 segmenti ogni mezzo secondo (500 ms).

;Si suppone che il TMR1 lavori con un prescaler di 1/8, a 4 MHz.
;Calcoliamo il valore che dobbiamo caricare sul TMR1:
;       500ms=4*(1/8MHz)*valore*8 -->   valore(dec)=62500
;       valore(bin)=11110100 00100100
;                   -------- --------
;                    TMR1H    TMR1L

        LIST    P=16F870        ;definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni


        ORG     0
        GOTO    INIZIO
        ORG     4
        GOTO    INT
        ORG     5

;Programma di trattamento dell'interrupt.

INT     movlw   b'11011100'     ;c2(00100100)
        movwf   TMR1L
        movlw   b'00001100'     ;c2(11110100)
        movwf   TMR1H           ;Carichiamo il TMR1 con il complemento di 31250 in hex.
        bcf     PIR1,TMR1IF     ;Impostiamo a zero il flag TMR1IF
        incf    PORTB,F         ;Incrementiamo il valore della porta B che riflette il
                                ;risultato sul display a 7 segmenti
        retfie                  ;Ritorno da interrupt

;Programma principale.

INIZIO  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        clrf    TRISB           ;Porta B uscite
        bsf     PIE1,TMR1IE     ;Abilitiamo l'interruot del TMR1 (TMR1IE)
        bcf     STATUS,RP0      ;Torniamo al banco 0
        movlw   b'00110001'
        movwf   T1CON           ;Attiviamo il TMR1 e impostiamo il divisore da 1/8
        movlw   b'11000000'
        movwf   INTCON          ;Abilitiamo gli interrupt: globali e per il PEIE

NULLA   clrwdt                  ;Ciclo che non fa nulla
        goto    NULLA

        END
```

Codice del programma completo.



Risultato della compilazione.



Videata di simulazione di MPLAB.

con il programma e di provare le diverse opzioni viste quando lo abbiamo studiato.

Conclusioni

Il programma è stato predisposto per essere scritto sul microcontroller.

Questo programma, i precedenti e alcuni che vedremo lungo il corso dell'opera possono essere utilizzati per fare pratica con altre istruzioni e nuovi dispositivi. Infatti in questo programma potrete utilizzare istruzioni di rotazione, lo potrete ampliare in modo che ripeta il ciclo per un numero determinato di volte, lo potrete combinare con altri (ad esempio accendendo i LED in modo sequenziale e quando si attiva un pulsante fornire un numero casuale), ecc. Per diventare un esperto programmatore è necessario dedicare un pò di tempo al software, avere il coraggio di uscire dal copione e improvvisare programmi nuovi.

Esercizio 10: il TMR2, il programma

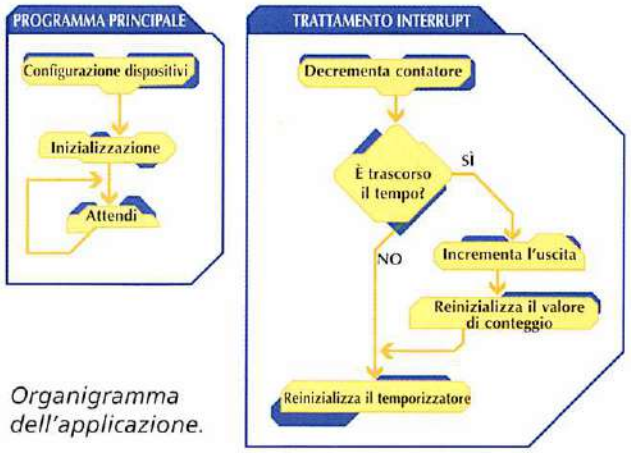
Continuiamo con gli esercizi lavorando, in questo caso, con il terzo temporizzatore, il TMR2. Questo temporizzatore, a differenza del TMR1, è da 8 bit e, come è stato spiegato nella teoria, è molto utilizzato per implementare alcune funzioni speciali che coinvolgono i moduli CCP e la porta seriale sincrona (SSP). L'obiettivo di questo esercizio è che impariate a lavorare con il temporizzatore e che continuiate ad acquisire esperienza nella programmazione.

Enunciato

Si vuole realizzare un programma che ogni 1,3 s provochi l'accensione successiva, come un contatore binario, della barra dei LED mediante l'utilizzo degli interrupt.

Utilizzeremo la porta B per fornire l'uscita ai diodi LED, e per ottenere il tempo specificato dovremo moltiplicare 65 ms (valore ottenuto con il predivisore e con il postdivisore) per 20 (numero di interrupt che si devono generare per fare in modo che si accenda un LED). Per ottenere i 65 ms supponiamo che i valori del predivisore e del postdivisore siano di 1/16 e che la frequenza del PIC sia di 4 MHz.

Nel programma principale dobbiamo configurare i dispositivi che vogliamo utilizzare. Programmeremo la porta B, i registri associati al Timer2 e gli interrupt. Nella routine dedicata agli interrupt svilupperemo il programma. Predisporremo in questa routine ciò che vogliamo far eseguire al programma ogni volta che passa un determinato periodo di tempo. Nell'organigramma si struttura la soluzione all'enunciato.



Organigramma dell'applicazione.

Codice

Il codice deve essere intestato come per gli esercizi precedenti, utilizzando i commenti per scrivere ciò che si vuole risolvere con il pro-

```
;                       ESERCIZIO: TMR2
;                      ---------------------

;Programma che ogni 1,3 s genera l'accensione sequenziale, come un contatore binario,
;della barra dei LED, utilizzando gli interrupt del TMR2.
;Il valore 1,3 s deriva dalla moltiplicazione di 65 ms (valore ottenuto con il prescaler e il postscaler)
;per 20 (n° di interrupt che si devono generare per accendere un LED).
;Per ottenere i 65 ms utilizzeremo dei valori di prescaler e di
;postscaler di 1/16 e la frequenza del PIC sarà 4 MHz.

;Calcolo dei tempi:
;       4Tosc=1 µs (Il TMR2 impiega questo tempo ad aggiornarsi)
;       Con i due divisori il range totale è 1/256
;       Quindi il valore da caricare sul PR2 sarà 255 ;(255*256µs=65 ms)



                LIST    P=16F870        ;Definiamo il nostro PIC
                INCLUDE "P16F870.INC"   ;File dei registri interni


                ORG     0
                goto    INIZIO
                ORG     4
                goto    INT
                ORG     5
```

L'nizio del programma è simile in tutte le applicazioni.

```
;Programma principale.

INIZIO          clrf    PORTB
                bsf     STATUS,RP0      ;Passiamo al banco 1
                clrf    TRISB           ;Porta B uscita
                movlw   b'11001111'
                movwf   OPTION_REG      ;WDT
                movlw   .255
                movwf   PR2             ;Carichiamo il PR2 con 255
                bsf     PIE1,TMR2IE     ;Abilitiamo l'interrupt del TMR2
                bcf     STATUS,RP0      ;Ritorniamo al banco 0
                movlw   b'01111111'
                movwf   T2CON           ;Attiviamo il TMR2 e configuriamo il prescaler
                                        ;e il postscaler con 1/16
                clrf    TMR2            ;Resettiamo il TMR2
                movlw   .20
                movwf   Cont1           ;Inizializziamo il contatore (ogni 20 int.)
                movlw   b'11000000'
                movwf   INTCON          ;Abilitiamo gli interrupt

NIENTE          clrwdt
                goto    NIENTE          ;Ciclo che non fa niente
                                        ;sino a quando non si produce
                                        ;l'interrupt
                END
```

Codice del programma principale.

```
                LIST    P=16F870        ;Definiamo il nostro PIC
                INCLUDE "P16F870.INC"   ;File dei registri interni

;Definizione delle variabili
Cont1           equ     0x20


                ORG     0
                goto    INIZIO
                ORG     4
                goto    INT
                ORG     5
```

Definizione della variabile che utilizzeremo per il contatore.

gramma. È sempre consigliabile fare questo per avere sempre presente ciò che esige l'enunciato. Continuiamo con le direttive di inizio del programma e procediamo allo sviluppo di ciò che sarà il programma principale.

Iniziamo quest'ultimo resettando qualsiasi valore residuo che ci potrebbe essere sulla porta che vogliamo utilizzare come uscita. Configuriamo la porta in modo che tutti i suoi terminali siano uscite digitali impostando il valore 0 al registro TRISB (clrf TRISB). Continueremo caricando il registro delle opzioni allo scopo di programmare il Watchdog (WDT). Caricheremo il PR2 con il periodo desiderato per il TMR2, che nel nostro caso è 255. Dovremo anche abilitare il temporizzatore e configurare i suoi divisori con il range previsto in precedenza. Questo viene fatto programmando il registro T2CON. Dato che il programma dipende dagli interrupt è necessario abilitarli, sia i globali che gli specifici del temporizzatore e, in ultimo, dobbiamo inizializzare il valore del conteggio. Con questo abbiamo configurato tutti i dispositivi che vogliamo utilizzare ma dobbiamo ancora decidere che cosa far fare al programma quando non ci sono gli interrupt del Timer. Nella nostra applicazione non è specificata nessuna funzione in più di quella di accendere un LED come un contatore ogni determinato periodo di tempo, quindi possiamo lasciare il programma principale in un ciclo di attesa nel quale non esegue nessuna azione.

Fatto questo il programma principale avrà una forma simile a quella della figura della pagina successiva.

Come abbiamo visto, per ottenere il tempo richiesto dobbiamo attendere che si generino 20 interrupt e per poter tenere conto di questo conteggio abbiamo bisogno di una variabile. La variabile si inizializza nel programma principale però è necessario definirla e assegnarle un indirizzo di memoria all'inizio del programma. Inserite la definizione nella posizione indicata, come mostrato in figura.

```
;Routine di trattamento dell'interrupt
INT             decfsz  Cont1,F         ;Verifichiamo se il tempo è passato
                goto    CONTINUA        ;Non ancora
                incf    PORTB,F         ;Incrementa il contatore binario della barra dei LED
                movlw   .20
                movwf   Cont1           ;Reinizializziamo il Cont1

CONTINUA        bcf     PIR1,TMR2IF     ;Resettiamo il flag del TMR2
                clrf    TMR2            ;Resettiamo il TMR2
                retfie
```

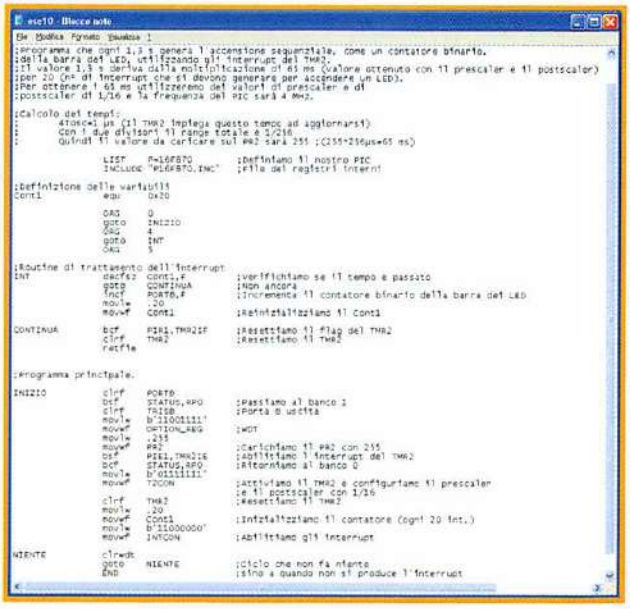
Codice della routine di interrupt.

Routine di servizio all'interrupt

In questa routine si deve risolvere l'applicazione. Definiamo che cosa deve succedere ogni volta che il temporizzatore termina il suo compito. Dato che vogliamo intervenire solamente quando si sono generati 20 interrupt, decrementeremo il contatore ogni volta che se ne produce uno e compareremo il suo valore con 0. Se avremo ottenuto i 20 interrupt (variabile = 0), agiremo sull'uscita incrementandola di una unità e reinizializzando il valore del contatore e il temporizzatore.

In caso contrario, aggiorneremo solamente il temporizzatore. Nella figura possiamo vedere un codice di esempio che risolve ciò che abbiamo previsto.

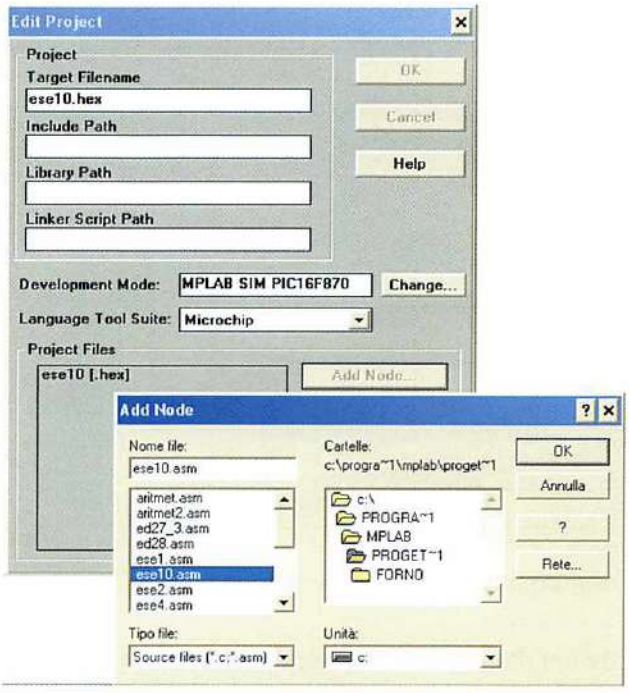
Non ci resta che assemblare il codice realizzato e aggiungere i commenti nel caso non l'avessimo ancora fatto. Il programma completo avrà un aspetto simile a quello della figura.



Aspetto del programma completo.

Compilazione

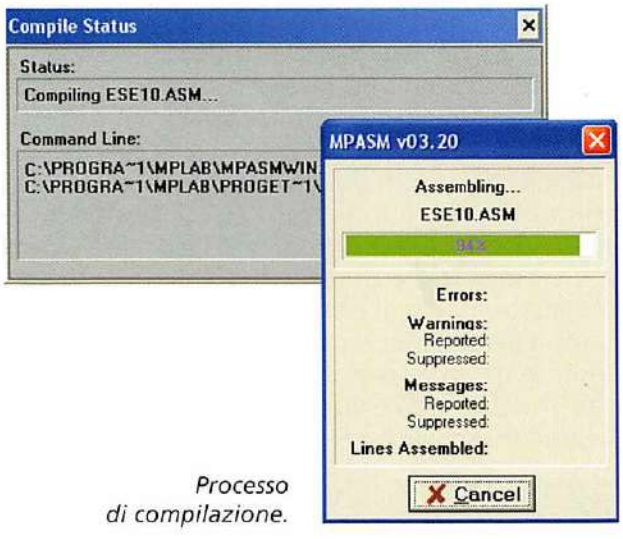
Abbiamo progettato un codice che risolve l'esercizio e ora dobbiamo verificare di averlo scritto bene e di aver utilizzato il repertorio delle istruzioni in modo corretto. Facciamo partire MPLAB e creiamo un nuovo progetto. Ricordate che per evitare possibili confusioni è consigliabile dare lo stesso nome sia al progetto che al codice in assembler. Creiamo un oggetto e, editandolo, aggiungiamo a esso il nostro codice. Nel menù **File** selezioniamo l'opzione **Open** per visualizzare così sul monitor il nostro codice. Fatto questo, il passo successivo è quello di compilare il codice per generare il file corrispondente in codice macchina (.hex), sempre che il nostro codice non contenga errori, successivamente scriveremo questo file sul microcon-



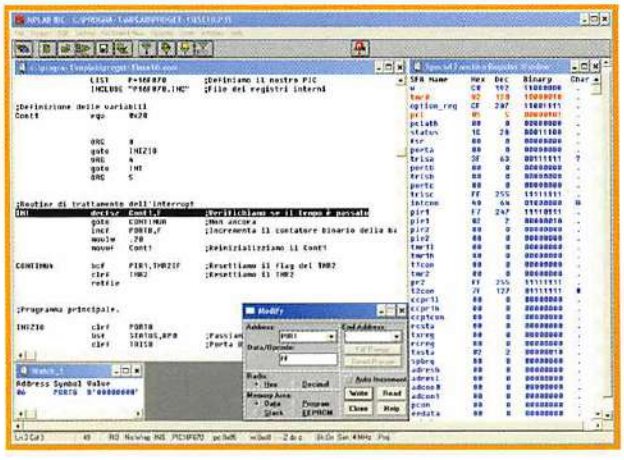
Finestra di edizione del progetto nella quale associamo il nostro codice.

```
Build Results
Building ESE10.HEX...

Compiling ESE10.ASM:
Command line: "D:\PROGRA~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\PROGRA~1\MPLAB\PROGET~1\ESE1
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE10.ASM 48 : Register in operand not in bank 0.
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE10.ASM 50 : Register in operand not in bank 0.
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE10.ASM 52 : Register in operand not in bank 0.
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE10.ASM 53 : Register in operand not in bank 0.

Build completed successfully.
```

Risultato della compilazione.



Processo di compilazione.



Aspetto finale della videata di simulazione.

troller. Per compilare ricordate che potete farlo mediante il menù **Project → Build All**, premendo Control+F10 oppure mediante l'icona corrispondente sulla barra degli strumenti.

Il codice predisposto come esempio, che potrete trovare sul secondo CD fornito, non presenta errori ma solamente i messaggi a cui siamo già abituati inerenti alla posizione dei registri sui banchi di memoria. Nella figura è riportato il risultato ottenuto dalla compilazione eseguita con successo.

Simulazione

Dobbiamo cercare, per quanto possibile, di verificare che il programma risponda a quanto specificato nell'enunciato prima di scriverlo sul PIC. A questo scopo eseguiamo la simulazione passo a passo e osserviamo che ogni linea di codice risponde a ciò che vogliamo che faccia. Mediante il menù **Debug → Run → Step** simuliamo il codice linea a linea, anche se un modo più semplice per realizzare questo tipo di simulazione è quello di premere F7. Apriremo la finestra dei Registri con funzioni speciali e quella dedicata unicamente all'osservazione dei parametri di uscita. Potremo eseguire tutte le linee di codice del programma principale, però arrivati al ciclo di attesa in cui il programma entra senza fare nulla aspettando che si generino gli interrupt, il simulatore si fermerà senza poter entrare nella routine di interrupt. Il registro TMR2 si incrementerà fino al suo valore limite di 255.

Per entrare nella routine di interrupt dobbiamo simulare che quest'ultimo si sia verificato e questo si può fare modificando il valore del registro PIR1. Forzeremo in questo modo l'interrupt (possiamo impostare un valore FF sul registro, unicamente per la simulazione) entreremo nella routine e simuleremo il suo corretto funzionamento. Per modificare il valore del registro faremo uso della finestra **Modify**. Nella figura è riportato l'aspetto che dovrebbe avere la videata di MPLAB durante la simulazione.

Esercizio 11: modulo CCP in modo comparazione, il programma

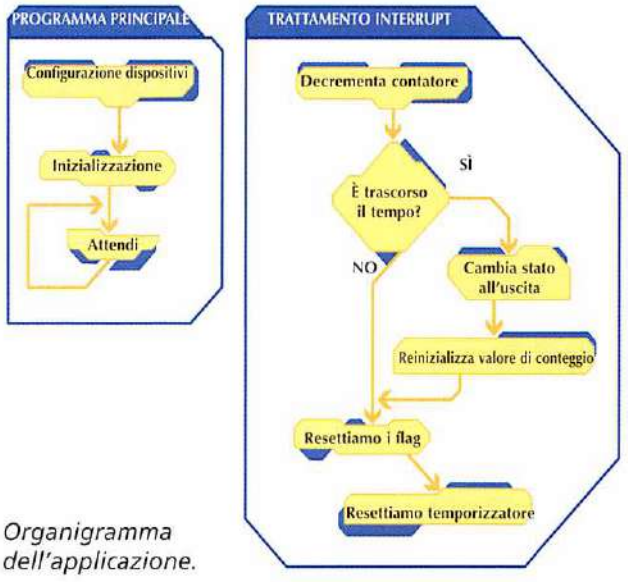
I moduli CCP possono lavorare in tre forme differenti:
- Acquisizione del valore del TMR1 quando si genera un evento su RC2/CCP1.
- Comparazione del valore del TMR1 con il valore caricato su CCP1H-L; se sono uguali si genera un evento sul pin RC2/CCP1.
- Modulatore di impulsi.
In questo esercizio lavoreremo con questo modulo nel modo comparazione. Questo dispositivo comporta implicitamente di lavorare con il TMR1, utilizzeremo inoltre gli interrupt.

Enunciato

Vogliamo sviluppare un programma che accenda e spenga il LED corrispondente a RB0 quando il valore del temporizzatore TMR1 coincide con un valore prefissato. Questo valore verrà scritto sui registri CCP1H/L e utilizzeremo come esempio: 10101010 10101010. Per poter osservare il lampeggio del LED faremo in modo che si accenda o si spenga ogni 20 interrupt.

Lavoreremo con la porta B per fornire l'uscita al LED, e anche con i dispositivi CCP, TMR1 e interrupt.

Questi esercizi hanno un livello di difficoltà medio, dato che utilizzano diversi dispositivi, ma sono esercizi di base per imparare a gestire questi dispositivi e, in seguito, saperli utilizzare in applicazioni più complesse.



Organigramma dell'applicazione.

Organigramma

Utilizziamo gli organigrammi per fare una pianificazione generale di come risolvere l'esercizio. Questo caso è simile a quelli presentati in precedenza nel senso che dovremo prevedere un programma principale in cui configurare tutti i dispositivi, e una subroutine di trattamento degli interrupt in cui eseguiremo le azioni finali.

Registri da configurare	Registri da inizializzare
TRISB	PORTB
OPTION REG	CCPR1H
PIE1	CCPR1L
CCP1CON	TMR1H
INTCON	TMR1L
T1CON	CONT
	PIR1

Registri con cui vogliamo lavorare.

Nel programma principale configureremo la porta di uscita resettandola per cancellare eventuali valori residui, configureremo il predivisore per il WDT, abiliteremo gli interrupt, configureremo il modulo CCP, il TMR1 e inizializzeremo tutti i registri con cui vogliamo lavorare. Fatto questo non ci resta che attendere all'interno di un ciclo infinito che si producano gli interrupt dovuti al modulo CCP.

Nella subroutine dell'interrupt eseguiamo le azioni sull'uscita solamente se si sono generati 20 interrupt.

```
;       ESERCIZIO: MODULI CCP in modo comparazione
;      ---------------------------------------------

;Programma che accende e spegne il LED corrispondente a RB0 quando il valore TMR1
;coincide con il valore che abbiamo preimpostato sui registri CCP1H-L, ad esempio,
;10101010 10101010. Per poter osservare il lampeggio faremo accendere e spegnere
;il LED ogni 20 interrupt.


        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File di definizione dei registri interni

Cont1           equ     0x20
Cont2           equ     0x21


                ORG     0
                GOTO    INIZIO
                ORG     4
                GOTO    INT
                ORG     5
```

Intestazione del codice.

```
;Programma principale.

INIZIO          clrf    PORTB
                clrf    PORTC           ;Disattiviamo gli I/O
                bsf     STATUS,RP0      ;Passiamo al banco 1
                clrf    TRISB           ;Porta B uscita
                movlw   b'11101111'
                movwf   OPTION_REG      ;Prescaler da 128 per il WDT
                bsf     PIE1,CCP1IE     ;Abilitiamo gli interrupt tramite CCP1
                bcf     STATUS,RP0      ;Torniamo al banco 0
                movlw   b'00001011'
                movwf   CCP1CON         ;Configuriamo il CCP1 in modo comparazione con attivazione
                                        ;speciale (resetta il TMR1) quando coincide con il
                                        ;valore su CCPR1H e CCPR1L

                movlw   b'10101010'
                movwf   CCPR1H
                movlw   b'10101010'
                movwf   CCPR1L          ;Carichiamo i registri CCP1H-L con un valore (43690)
                clrf    TMR1L
                clrf    TMR1H           ;Resettiamo il TMR1
                movlw   b'11000000'
                movwf   INTCON          ;Abilitiamo gli interrupt
                movlw   b'00000001'
                movwf   T1CON           ;Attiviamo il TMR1 e configuriamo il prescaler con 1/1
                movlw   .20
                movwf   Cont1
                bcf     PIR1,TMR1IF     ;Resettiamo il flag del TMR1
                bcf     PIR1,CCP1IF     ;Resettiamo il flag del modulo CCP1

NIENTE          clrwdt
                goto    NIENTE          ;Ciclo che non fa niente sino a quando
                                        ;non si produce l'interrupt
                END
```

Esempio di codice che formerà il programma principale.

Codice

Inizieremo a sviluppare il codice intestando il programma con i commenti. Definiremo il microcontroller e la libreria dei registri e, mediante le direttive ORG, le zone di memoria che occuperà il programma.

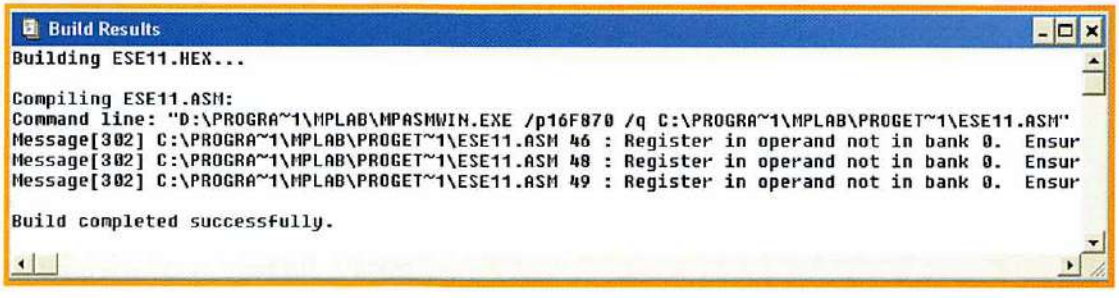
Anche se è consigliabile definire le variabili man mano che si rendono necessarie, lo si può fare all'inizio del programma facendo una stima di ciò di cui avremo bisogno. Sappiamo che ne utilizzeremo un paio per il contatore, ma ancora non sappiamo se avremo bisogno di altre variabili, quindi ne creeremo due. Si possono creare quante variabili si vuole, ma bisogna tener presente che occupano memoria e le risorse utilizzate devono essere sempre ottimizzate. Nella figura possiamo vedere la forma che sta assumendo il codice e come abbiamo definito due variabili, anche se per ora non sappiamo se le utilizzeremo.

Programma principale

Mettiamo l'etichetta "Inizio" e iniziamo a programmare. Cancelliamo eventuali valori residui che ci potrebbero essere sulle porte (sarebbe consigliabile farlo solamente sulle porte che desideriamo utilizzare). Configuriamo la porta B come uscita cancellando il registro TRISB. Configuriamo il Watchdog modificando il registro OPTION_REG. Abilitiamo gli interrupt dovuti al modulo CCP e configuriamo quest'ultimo per farlo lavorare nel modo comparazione e che imposti a 0 il temporizzatore se è stato raggiunto il valore desiderato. Il registro CCPCON permette di configurare il modulo CCP. Dobbiamo inizializzare tutti i re-

```
;Routine di trattamento dell'interrupt.

INT             decfsz  Cont1,F         ;Verifichiamo se il tempo è trascorso
                goto    CONTINUA        ;Ancora no
                movfw   PORTB           ;Se è spento si accende
                xorlw   b'00000001'     ;Altrimenti si spegne
                movwf   PORTB
                movlw   .20
                movwf   Cont1           ;Reinizializziamo il Cont1
CONTINUA        bcf     PIR1,TMR1IF     ;Azzeriamo il flag del TMR1
                bcf     PIR1,CCP1IF     ;Azzeriamo il flag del modulo CCP1
                clrf    TMR1L
                clrf    TMR1H           ;Resettiamo il TMR1
                retfie
```

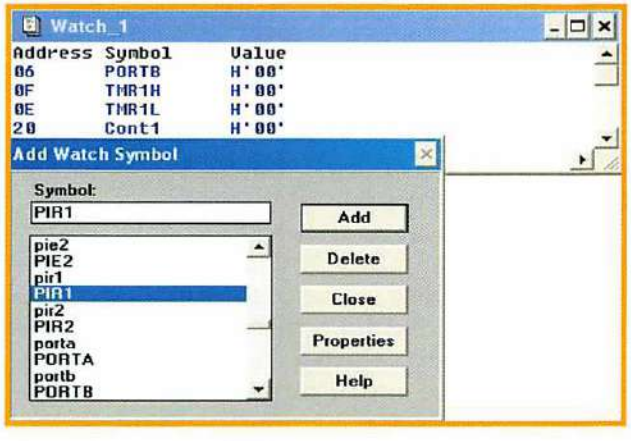
Codice della subroutine dedicata all'interrupt.



Risultato della compilazione.

gistri, quindi su CCPR1H/L caricheremo il valore desiderato, resetteremo i registri TMR1H/L, inizializzeremo la variabile che utilizzeremo come contatore e resetteremo i flag di segnalazione riferiti ai dispositivi con cui vogliamo lavorare. In ultimo configureremo il registro INTCON per abilitare gli interrupt e il registro T1CON del temporizzatore.

Ora non ci resta che attendere all'interno di un ciclo che si generino gli interrupt. Per maggior sicurezza, per evitare che il programma possa rimanere bloccato, resettiamo il WDT (Watchdog) all'interno del ciclo.



Per aggiungere registri da visualizzare selezioneremo Add.

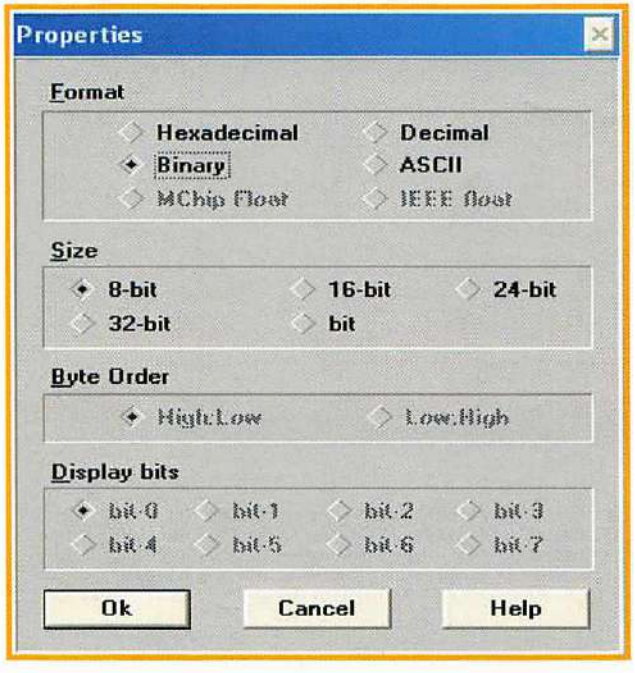
Routine di servizio dell'interrupt

Questa routine è simile a quella progettata negli esercizi precedenti. Quando si genera un interrupt decrementiamo il contatore e, se questi è arrivato a 0, agiamo sull'uscita, inizializziamo il valore del contatore e quello del temporizzatore, e resettiamo i flag di indicazione. Nel caso in cui il contatore non sia ancora a 0 resetteremo unicamente i flag e inizializzeremo il valore del temporizzatore.

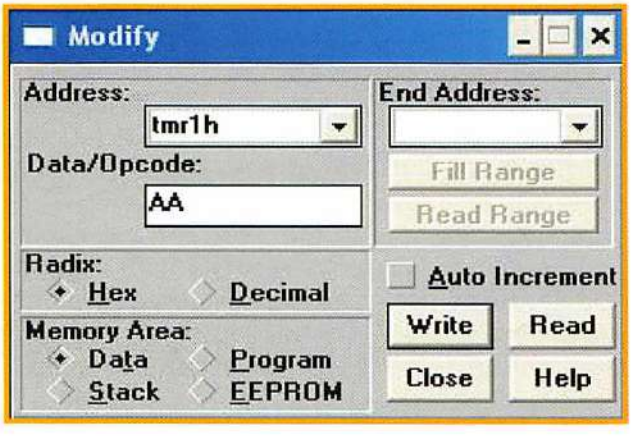
Compilazione

Abbiamo progettato il nostro codice e lo dobbiamo compilare. Creiamo un progetto, associamo il codice creato a quest'ultimo e selezioniamo le opzioni di compilazione (Build All). Per l'esempio che abbiamo presentato il risultato di questo processo di compilazione è quello mostrato nella figura.

Questo esempio lo potrete trovare sul secondo CD-ROM con il nome "ese11.asm", vi consigliamo comunque di provare a realizzare voi stessi i programmi, e di tenere questi esempi solamente come fonte di consultazione.



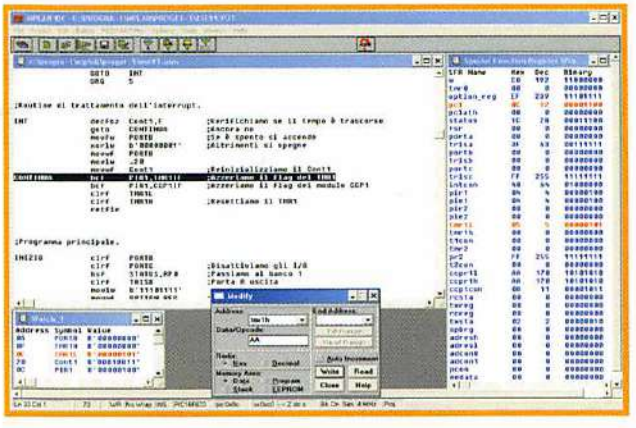
Cambiamo il formato dei dati presentandoli in binario.



Forziamo il valore del registro TMR1H.

Simulazione

Per simulare il corretto funzionamento del programma apriremo le finestre abituali, una per i registri delle funzioni speciali e l'altra finestra dedicata ai registri che vogliamo vedere in modo indipendente. In questa finestra selezioneremo i registri: PORTB, TMR1H, TMR1L, CONT1 e PIR1, presenteremo i dati in formato binario. Ricordate che per fare questo è necessario cliccare con il pulsante sinistro del mouse sul simbolo situato nell'angolo superiore sinistro della finestra e selezionare Edit Watch. A questo



Aspetto generale di MPLAB durante la simulazione.

punto si apre una finestra dove possiamo vedere i registri che vogliamo visualizzare, questi possono essere modificati selezionandoli uno per uno premendo Properties. Apparirà una nuova finestra come quella riportata nella figura a fianco, in cui sceglieremo l'opzione Binary nella sezione Format.

Quando eseguiamo passo a passo la simulazione vediamo come si configurano correttamente i dispositivi nel programma principale, come al momento in cui arriva l'attivazione dell'interrupt entri nella subroutine e terminata questa ritorni al programma principale, continuando con la configurazione e inizializzazione dei registri. All'interno del ciclo infinito, premendo F7 (simulazione passo a passo) incrementeremo di una unità per volta i registri del temporizzatore, quindi per non perdere troppo tempo e arrivare al valore desiderato, forzeremo il valore del registro TMR1H a 10101010. Faremo questo con la finestra Modify, inserendo il valore nel suo equivalente esadecimale (10101010=AA). Continuiamo a eseguire passo a passo, fino a quando il temporizzatore raggiungerà il valore desiderato, momento in cui si genera un interrupt, cambia lo stato del registro PIR1 e saltiamo alla routine di servizio per l'interrupt.

Nella subroutine si eseguono tutti i passi correttamente, ma se vogliamo vedere come funziona il codice sull'uscita dobbiamo forzare anche il valore del registro CONT1. Se scriviamo su questo registro il valore 01h la prossima volta che entreremo nella subroutine agiremo sull'uscita. In questo modo il programma rimane simulato in modo soddisfacente, ed è pronto per essere scritto sul PIC.

Esercizio 12: moduli di modulazione di ampiezza degli impulsi PWM, il programma

Un'altra delle funzioni del modulo CCP è quella della modulazione di ampiezza degli impulsi PWM. Nell'esercizio che proponiamo è presentato un semplice esempio in cui si sviluppa questa funzione.

Introduzione

La modulazione di ampiezza degli impulsi ha svariate applicazioni. Si tratta di poter controllare l'ampiezza di un impulso di un'onda quadra. In questo modo, conoscendo la frequenza dell'onda, possiamo grazie a essa trasportare informazioni (FM frequenza modulata) o esercitare un determinato controllo su di un dispositivo, come ad esempio un motore.

Nell'immagine della figura possiamo vedere come partendo da un'onda quadra con ampiezza di impulso costante (onda portante), quando la si modula usando un'onda sinusoidale, otteniamo un'onda quadra modulata in cui l'ampiezza dell'impulso è variabile in funzione dell'onda sinusoidale. Se compariamo l'onda risultante con la portante, possiamo vedere le differenze fra la durata degli impulsi e interpretarle, ottenendo un'informazione.

Enunciato

Vogliamo generare un segnale ad onda quadra sulla linea R2/CCP1 il cui periodo si possa modificare, così come l'ampiezza del circuito (Duty Cycle).

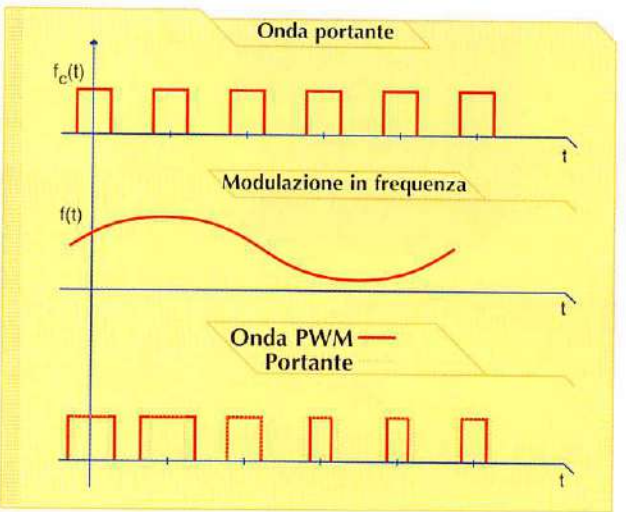
Il periodo (T) e la durata dell'impulso "Duty Cycle" (d) si determinano seguendo le formule della figura.

L'esercizio che viene presentato come esempio di soluzione e che nel secondo CD ha il nome di "ese12.asm", utilizza il modulo CCP1 con uscita del segnale sulla linea RC2/CCP1. Il segnale di uscita ha un periodo di 4 ms, dato che viene fissato il predivisore a 16 e il registro dei periodi si carica con il valore 250. L'ampiezza del ciclo "Duty" è variabile, e si determina in base al valore binario degli interruttori RB0-RB7.

$$T = (PR2 + 1) \cdot 4 \cdot T_{OSC} \cdot \text{TMR2 Predivisore}$$

$$d = (CCPR1L : CCPCON1 < 5 : 4 >) \cdot T_{OSC} \cdot \text{TMR2 Predivisore}$$

Formule per calcolare il periodo e la durata dell'impulso.



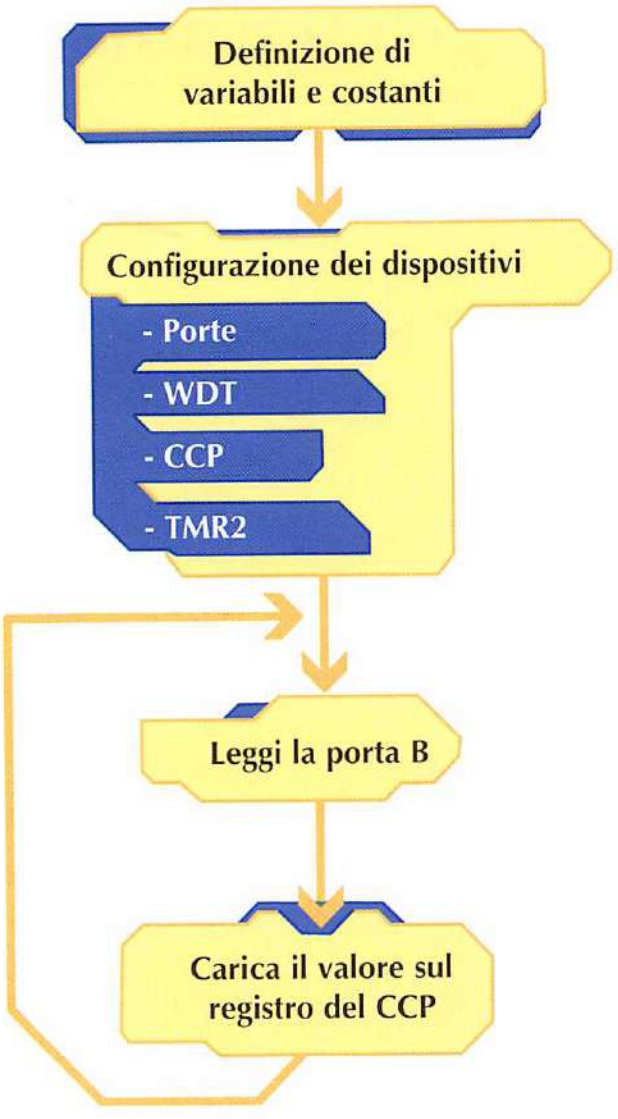
Modulazione PWM.

Organigramma

Per risolvere l'enunciato dobbiamo ricorre alla teoria e tener presente quali sono i registri che intervengono nel modo PWM e come lavorare con essi. Prima di iniziare a programmare, dobbiamo plasmare la forma che avrà il codice in un organigramma.

Codice

Utilizzando come base l'organigramma della figura inizieremo a progettare il codice e il miglior mo-



Organigramma dell'applicazione.

do di iniziare è inserire l'enunciato dell'applicazione sotto forma di commenti, nel file ".asm". In questo modo risponderemo a ciò che esegue il programma. Continuiamo intestando il programma con la definizione del processore, la libreria dei registri e le direttive di organizzazione della memoria.

Inizieremo il programma nell'etichetta Inizio, pulendo i possibili valori residui sulle porte che utilizzeremo come uscita. Configureremo le porte che useremo: Porta B come ingresso e pin RC2 come uscita. Configureremo il WDT nel registro OPTION_REG. Dobbiamo ricordare che WDT va utilizzato ogni volta che il programma comprende un ciclo infinito, per evitare che il microprocessore si inceppi.

Poiché nell'enunciato si chiede che il periodo si possa scegliere, definiremo una costante con il nome Periodo, che nell'esempio prende il valore di 250. Questa costante verrà scritta nel registro dei periodi PR2.

In ultimo, per quanto riguarda le configurazioni, programmeremo il modulo CCP in modo PWM e attiveremo il TMR2, che è il temporizzatore associato a questo modo.

A questo punto abbiamo tutti i dispositivi debitamente configurati, non ci rimane quindi che leggere il valore degli ingressi e passare quest'ultimo al registro che determina l'ampiezza dell'impulso (CCPR1L). Questo viene eseguito in modo ciclico, all'interno di un ciclo, in modo che si possa modificare l'ampiezza di un impulso in qualsiasi momento.

Utilizzeremo una variabile chiamata Tem-

```
;       ESERCIZIO: Moduli CCPx in modo PWM. Modulazione dell'ampiezza degli impulsi.
;       ------------------------------------------------------------------------
;
;consiste nel generare un segnale ad onda quadra sulla linea RC2/CCP1 il cui periodo può
;essere modificato, così come l'ampiezza dell'impulso (Duty Cycle). Il periodo si determina con la
;formula T=(PR2+1)*4*Tosc*TMR2 prescaler. La durata dell'impulso, o "Duty Cycle" (d), si deter-
;mina con d=(CCPR1L:CCPCON1<5:4>)*Tosc*TMR2 prescaler.
;
;L'esempio utilizza il modulo CCP1, l'uscita del segnale è mandata sulla linea RC2/CCP1 e il prescaler
;vale 16. Il segnale di uscita ha un periodo di 4 ms. L'ampiezza del ciclo "Duty"  è variabile
;e si determina in base al valore binario degli interruttori RB0-RB7.
;

                List    p=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizione dei registri interni

                org     0x00            ;Vector di Reset
                goto    Inizio

                org     0x05            ;Salva il vector di interrupt
```

Intestazione del programma.

```
;Programma principale

Inizio          clrf    PORTC           ;Cancella le uscite
                bsf     STATUS,RP0      ;Seleziona il banco 1
                movlw   b'00000110'
                movwf   ADCON1          ;Porta A I/O digitali
                movlw   b'11111111'
                movwf   TRISB           ;Porta B si configura come ingresso
                movlw   b'11111011'
                movwf   TRISC           ;RC2 uscita
                movlw   b'11101111'
                movwf   OPTION_REG      ;Prescaler di 128 associato al WDT
                movlw   Periodo-1
                movwf   PR2             ;Carica il registro dei periodi
                bcf     STATUS,RP0      ;Seleziona banco 0

;Il modulo CCP1 funziona in modo PWM con uscita del segnale su RC2/CCP1

                movlw   b'00001100'
                movwf   CCP1CON

;Il TMR2 lavora con un prescaler 1:16 quindi a una frequenza di 4 MHz evolve
;ogni 16 µS ((4*Tosc)*16)

                movlw   b'00000111'
                movwf   T2CON           ;T2 a On
```

Codice in cui si configurano tutti i dispositivi.

porale, come semplice contenitore temporale del valore degli ingressi.

Abbiamo utilizzato una costante e una variabile, le dobbiamo quindi definire all'inizio del programma, prima delle direttive ORG. Nelle figure possiamo verificare come si sviluppano tutte le fasi del codice.

Compilazione

Dobbiamo verificare che il codice che abbiamo confezionato non abbia errori, a questo scopo lo do-

```
Loop            clrwdt                  ;Aggiorna il WDT
                movf    PORTB,W
                movwf   Temporale       ;Carica il valore determinato da RB0-RB7
                movwf   CCPR1L          ;Carica l'ampiezza dell'impulso (n°Prescaler di 16)
                goto    Loop            ;Ciclo infinito

                end                     ;Fine del programma sorgente
```

Ciclo infinito in cui leggiamo il valore degli ingressi.

```
                List    p=16F870        ;Tipo di processore
    Variabile   include "P16F870.INC"   ;Definizione dei registri interni
Temporale       equ     0x20            ;Variabile temporale
Periodo         equ     .250            ;Periodo de 250*Prescaler di 16 (4 mS)
    Costante    org     0x00            ;Vector di Reset
                goto    Inizio

                org     0x05            ;Salva il vector di interrupt
```

Definizione delle costanti e delle variabili utilizzate.

```
Build Results
Building ESE12.HEX...

Compiling ESE12.ASM:
Command line: "D:\PROGRA~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM"
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM 31 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM 33 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM 35 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM 37 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGET~1\ESE12.ASM 39 : Register in operand not in bank 0.  Ensure

Build completed successfully.
```

Risultato della compilazione.

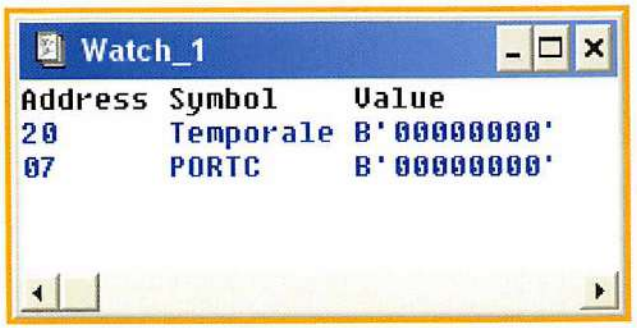
vremo compilare mediante MPLAB. Creeremo un progetto che editeremo per poter includere al suo interno, il nostro file di codice e selezioneremo l'opzione di assemblaggio e compilazione: Build All.

Il risultato della compilazione è visualizzato nell'immagine della figura.

Come potete verificare il compilatore ci dà i tipici messaggi di posizione riferiti ai banchi di memoria dei registri, però questi messaggi non indicano errori all'interno del programma. Nel caso in cui la compilazione rilevi degli errori, li dovremo risolvere uno ad uno con l'aiuto offerto dal compilatore, dato che verrà indicata sia la linea di errore che la sua natura.

Simulazione

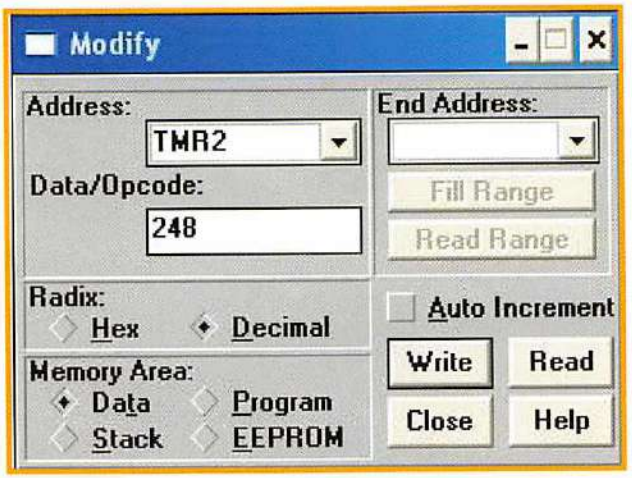
Dopo aver compilato il programma con successo passeremo a simularne il funzionamento. Dobbiamo aprire la finestra dei Registri delle Funzioni Speciali e una finestra dove po-



Forziamo il valore del registro TMR2 per rendere più agile la simulazione.

Finestra di visualizzazione dei registri più importanti.

ter avere un rapido accesso visivo ai registri di uscita e a quelli di ingresso. Il miglior modo di visualizzare questi registri sarà in binario, quindi dovremo cambiare il modo di rappre-



Possiamo simulare gli ingressi mediante il simulatore di stimoli.

sentazione nelle proprietà dei registri. Per simulare gli ingressi che determineranno l'ampiezza degli impulsi dobbiamo utilizzare il simulatore di stimoli (Debug → Simulator Stimulus → Asynchronous Stimulus) in modo da assegnare a ogni pin della porta B uno stimolo. Realizzeremo questo, cliccando con il pulsante destro del mouse su ogni pulsante e scegliendo l'opzione Assign Pin.

Inizieremo a simulare passo a passo premendo il pulsante F7. Quando arriveremo al ciclo finale potremo vedere come cambia l'uscita, ma per realizzare una corretta simulazione dovremo cambiare i valori di ingresso passando alcuni stimoli a High. Dovremo cliccare sul pulsante destro sul pin che vogliamo modificare e scegliere l'opzione High, in modo che quando si desidera cambiare valore, cioè fare in modo che lo stimolo diventi effettivo, sarà sufficiente cliccare sul pulsante sinistro di questo pin.

Dato che per poter visualizzare una variazione sull'uscita dovremo attendere che il temporizzatore TMR2 raggiunga il suo valore di conteggio, potremo forzare quest'ultimo mediante la finestra Modify. In questo modo renderemo più agile il processo di simulazione. Nell'immagine della figura possiamo vedere un esempio di come forzare il valore del temporizzatore.

Possiamo verificare come la simulazione del codice presentata come esempio risulti soddisfacente, quindi siamo pronti per la fase di scrittura e montaggio.

Esercizio 13: convertitore A/D, il programma

Nell'esercizio che vogliamo eseguire lavoreremo con uno dei dispositivi più utilizzati del microcontroller, il convertitore A/D. Molte delle grandezze o dei segnali che esistono nell'ambiente sono di natura analogica e per essere trattate da un microcontroller o da un microprocessore devono essere convertite in digitale. Da qui l'importanza dei convertitori A/D.

Enunciato

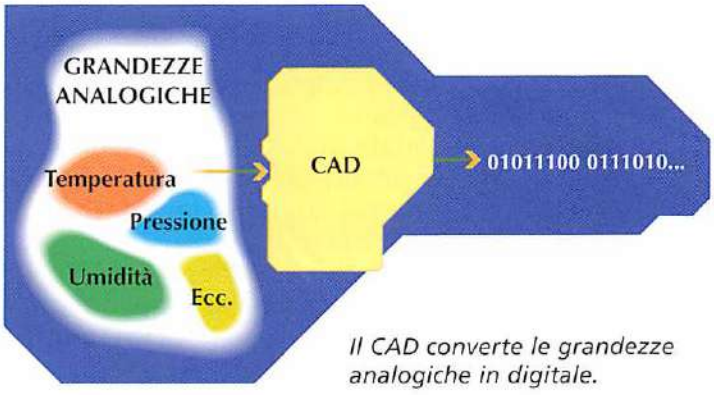
Si tratta di sviluppare un programma che realizzi la conversione di un segnale analogico in uno digitale e lo rappresenti in binario mediante i LED della matrice dei LED.

Il segnale analogico entrerà sul terminale RA2/AN2 e utilizzeremo una frequenza di lavoro Fosc/8.

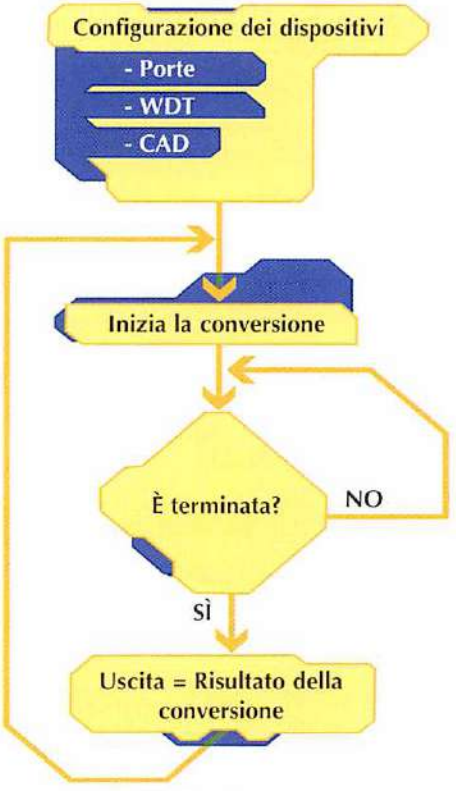
Questo esercizio è un esempio molto semplice di utilizzo del convertitore, e dopo averlo capito ci servirà per risolvere applicazioni più complesse.

Organigramma

Ogni dispositivo con cui lavoriamo ha le sue particolarità. In questo caso il CAD impiega un tempo per realizzare la conversione, tempo che dobbiamo tenere in considerazione al momento di sviluppare il programma. Seguendo l'organigramma della figura, la prima cosa che dobbiamo fare, tralasciando l'intestazione, è configurare tutti i dispositivi con cui vogliamo lavorare. Dopodiché acquisiremo il valore analogico dell'ingresso e inizieremo la conversione. Trascorso il tempo necessario per eseguire quest'ultima, forniremo sull'uscita il corrispondente valore in binario.



Il CAD converte le grandezze analogiche in digitale.



Organigramma dell'applicazione.

Codice

Nel secondo CD allegato all'opera, troverete un esempio di codice, con il nome di "ese13.asm", che risolve questo enunciato. Noi cercheremo di fare da soli il nostro programma, anche se utilizzeremo questo esempio come base per l'applicazione.

Inizieremo, come in tutti i programmi, specificando la funzione del codice che si vuole sviluppare, rispondendo alla domanda: che cosa fa?

Intesteremo il programma con le funzioni e le direttive abituali, e ci preparere-

```
;       ESERCIZIO: CAD (CONVERTITORE ANALOGICO DIGITALE)
;      ------------------------------------------------

;Programma che realizza la conversione di un segnale analogico che arriva su AN2 in uno digitale
;rappresentato in binario mediante i LED della barra dei LED.
;Utilizzeremo una frequenza di lavoro Fosc/8.


        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni


        ORG     0
        GOTO    INIZIO
        ORG     5
```

Intestazione del programma.

```
;Programma principale.

INIZIO  clrf    PORTA
        clrf    PORTB
        movlw   b'01010001'
        movwf   ADCON0          ;Configura CAD, sceglie canale AN2 e abilita l'utilizzo del convertitore
        bsf     STATUS,RP0      ;Passiamo al banco 1
        movlw   b'00000000'
        movwf   ADCON1          ;Configuriamo la Porta A come I/O analogiche
        clrf    TRISB           ;Porta B uscita
        movlw   b'11001111'
        movwf   OPTION_REG      ;WDT
        movlw   b'11111111'
        movwf   TRISA
        bcf     STATUS,RP0      ;Torniamo al banco 0
```

Inizio del programma principale in cui si configurano i dispositivi.

```
CICLO   clrwdt
        clrf    PORTB
        bcf     PIR1,ADIF       ;Resettiamo il flag dell'interrupt del CAD
        bsf     ADCON0,GO       ;Comando di inizio della conversione

ATTENDERE btfsc   ADCON0,2      ;Verifichiamo se la conversione è terminata testando
                                ;lo stato del bit GO/DONE#
        goto    ATTENDERE
        call    LED             ;Visualizziamo il risultato sulla barra dei LED
        goto    CICLO

        END
```

Ciclo con il codice che esegue tutte le azioni per risolvere l'applicazione.

mo alla configurazione dei dispositivi con cui vogliamo lavorare. Molte volte è necessario ricorrere alla teoria, per poter gestire i registri associati ai dispositivi con cui si vuole lavorare nell'applicazione. Il programma principale, a partire dall'etichetta "Inizio", comincia cancellando le porte, pulendole da qualsiasi valore residuo. Di seguito si configurano tutti i dispositivi. Nell'esempio di questo esercizio il primo dispositivo che si configura è il convertitore A/D. Programmiamo il registro ADCON0 in modo da abilitare il convertitore e determiniamo come canale di ingresso AN2. Poi programmiamo le porte. Quindi la porta A sarà una porta di ingressi analogici (configurare il TRISA e ADCON1) e la porta B di uscite digitali (caricare il valore corrispondente sul TRISB). Dato che questo programma avrà anche un ciclo infinito dovremo abilitare il WDT e questo lo faremo nel registro OPTION_REG.

```
;Routine per il risultato della conversione.

LED     movf    ADRESH,W
        movwf   PORTB           ;Spostiamo il contenuto di ADRESH (risultato della
                                ;conversione) sulla barra dei LED
        return
```

Subroutine di accensione dei LED.

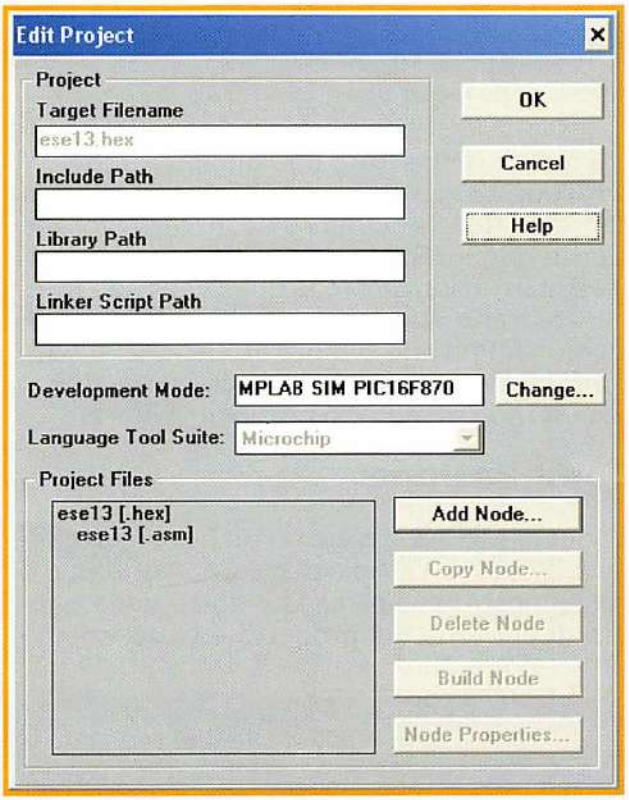
Dopo la configurazione

Configurati tutti i dispositivi, dobbiamo configurare il codice che risolva l'applicazione. Si vuole acquisire il valore dell'ingresso, convertirlo e successivamente passarlo sulla porta di uscita, il tutto ciclicamente. Creiamo un ciclo mettendo un'etichetta a cui poterci successivamente indirizzare, e inizieremo il ciclo resettando il WDT e cancellando i valori che ci potrebbero essere sulla porta di uscita. Dobbiamo anche resettare il flag o indicatore di interrupt ogni volta, perché ogni volta acquisiremo un nuovo valore e realizzeremo una nuova conversione.

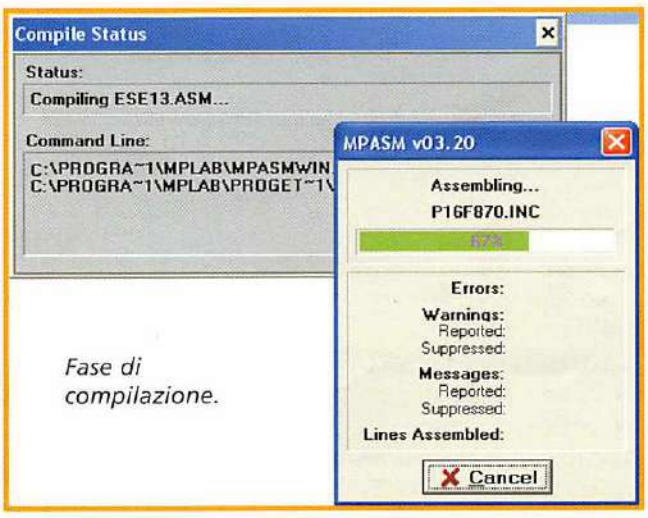
Ordineremo l'inizio della conversione, impostando a 1 il bit GO del registro ADCON0, però dobbiamo essere coscienti che la stessa impiega un certo tempo a essere eseguita, quindi ciò che dobbiamo fare è aspettare che finisca all'interno di un nuovo ciclo. Quando la conversione è terminata il flag GO/DONE# passerà a 1 e solo in quel momento usciremo dal ciclo di attesa. Ora non ci rimane che agire sull'uscita ed eseguire un salto incondizionato all'inizio del ciclo. Nell'immagine della figura è riportato il codice di esempio che corrisponde a quanto abbiamo appena detto.

Come potrete vedere, al momento di passare il valore risultato dalla conversione sulla porta di uscita, si esegue una chiamata a una subroutine (LED). Nei programmi lunghi, con molte linee di codice, conviene separare le diverse parti del programma, in subroutine, per facilitarne la comprensione e la messa a punto. Noi dobbiamo separare le azioni che riguardano l'uscita in una subroutine chiamata LED. In essa l'unica cosa che dobbiamo fare è spostare il valore risultante dalla conversione, contenuto in ADRESH, sulla porta B di uscita.

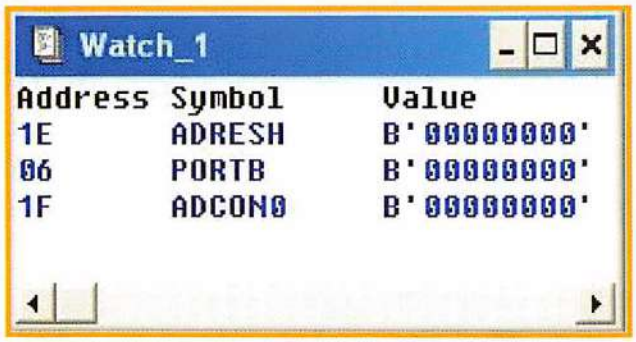
Invece di utilizzare una subroutine si poteva fare questo in due linee di codice, sostituendo la chiamata alla subroutine, e il pro-



Creiamo un progetto a cui associamo il nostro codice in assembler.



Fase di compilazione.



Finestra in cui realizzeremo i registri più importanti.

gramma sarebbe stato ugualmente valido.

In questo caso non dobbiamo cambiare banco, dato che i registri con cui lavorano queste due interazioni sono sullo stesso banco, e le istruzioni entro le quali vengono inseriti, lavorano anch'esse sullo stesso banco di memoria. Prestate sempre attenzione ai banchi di memoria, infatti risulta un errore tipico, che essendo di metodo possiamo evitare.

Compilazione

Apriamo MPLAB e creiamo un nuovo progetto nella nostra cartella di lavoro. Associamo a esso il nostro codice in assembler, nella finestra di edizione del progetto e visualizziamo nella finestra il nostro file, aprendolo nel menù File.

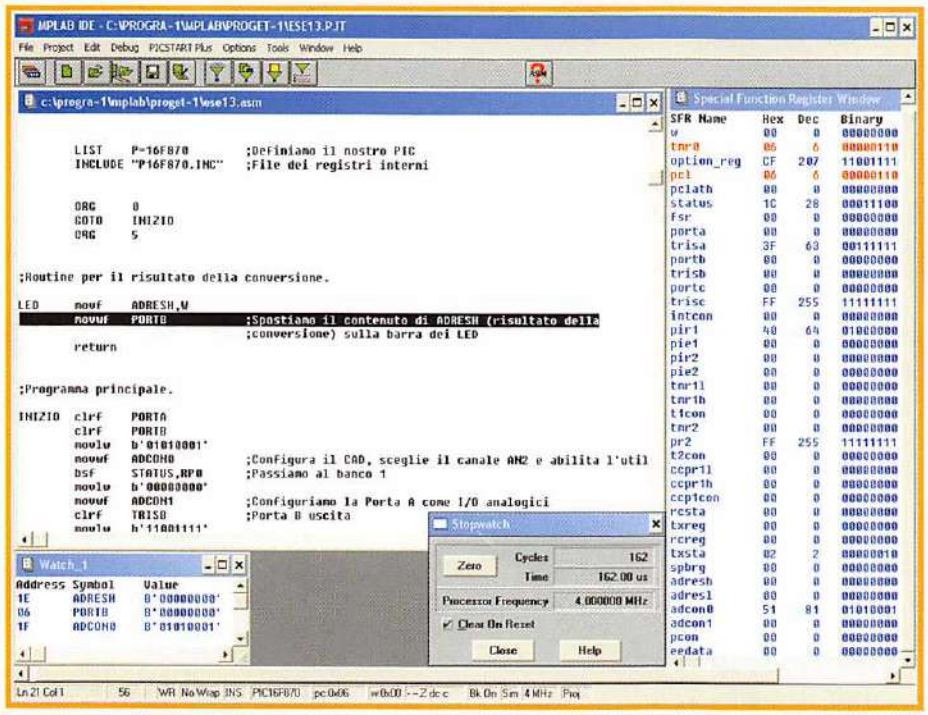
Compiliamo il codice e se tutto è stato fatto bene, sarà il file in codice macchina ".hex" da scrivere sul microcontroller.

Simulazione

Visualizzate la finestra dei registri delle funzioni speciali e quella specifica con i registri che vogliamo vedere in modo indipendente (ADRESH, PORTB, ADCON0), i valori di quest'ultima finestra li visualizzeremo in formato binario. Non possiamo simulare un valore analogico, però possiamo trovare che per un determinato valore iniziale otterremo l'uscita corrispondente. Se eseguiamo il programma passo a passo senza forzare nessun valore, verificheremo che al termine della conversione il risultato di quest'ultima è 0, dato che il valore di ingresso è anch'esso 0.

Pur non potendo provare in dettaglio il programma, possiamo vedere che le istruzioni che abbiamo utilizzato rispondono esattamente alle nostre aspettative, possiamo quindi supporre che ciò che non possiamo simulare, risponderà in modo soddisfacente al momento della prova con il circuito elettronico montato sul laboratorio.

Provate a compilare e simulare diverse alternative al programma, come ad esempio quella di non utilizzare una subroutine per le operazioni sull'uscita, o togliere l'istruzione clrf PORTB dall'inizio del programma principale. In questo modo verificherete che sono molte le possibilità che si hanno nel progettare un codice e acquisirete dimestichezza nella programmazione.



Aspetto di MPLAB durante la simulazione.

Esercizio 14: la memoria EEPROM, il programma

Nell'esercizio che vi presentiamo si lavora con la memoria EEPROM. Verrà sviluppato un programma che ci servirà per imparare a lavorare con questa memoria e poterla così utilizzare in applicazioni più complesse.

La memoria EEPROM

La memoria dei dati è formata da una memoria RAM e una memoria EEPROM. Nella RAM si scrivono i dati di utilizzo generale, che sono gestiti nel programma e i registri specifici i cui bit controlleranno il funzionamento del processore e i dispositivi interni. La memoria EEPROM (non volatile) si utilizza per scrivere i dati i cui valori devono essere mantenuti anche quando si toglie l'alimentazione. Questa memoria è molto utilizzata per applicazioni in cui è richiesta una chiave di sicurezza, come ad esempio un controllo di accessi.

Enunciato

Si tratta di sviluppare un programma che ottenga il valore dello stato degli interruttori RA4-RA0, lo scriva nella EEPROM quando RA4 è attivo e visualizzi il contenuto della EEPROM sulla matrice dei LED che sarà collegata alla porta B.

Come si può vedere l'esercizio previsto è un semplice programma, la cui unica difficoltà consiste nella gestione della memoria EEPROM.

Organigramma

Per sviluppare il codice che risolve l'applicazione, dobbiamo tenere presente che lavoreremo con ingressi e uscite digitali, e che utilizzeremo la EEPROM, in modo da poter scrivere su di essa e poterne leggere il contenuto. Per questa ragione separeremo il codice in tre parti differenti:

– Programma principale: configureremo i dispositivi comuni, che in questo caso saranno le porte di ingresso e di uscita.

– Scrittura: programmeremo la scrittura nella memoria EEPROM.

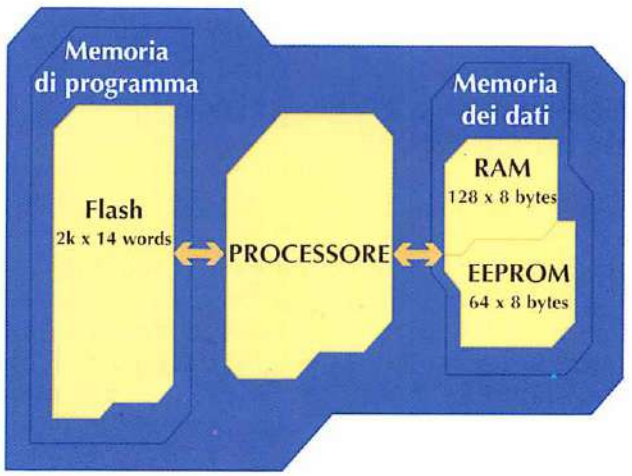
– Lettura: otterremo il valore desiderato di questa memoria e agiremo sulle uscite.

Nella figura in basso, si può vedere l'organigramma che corrisponde al proponimento appena esposto.

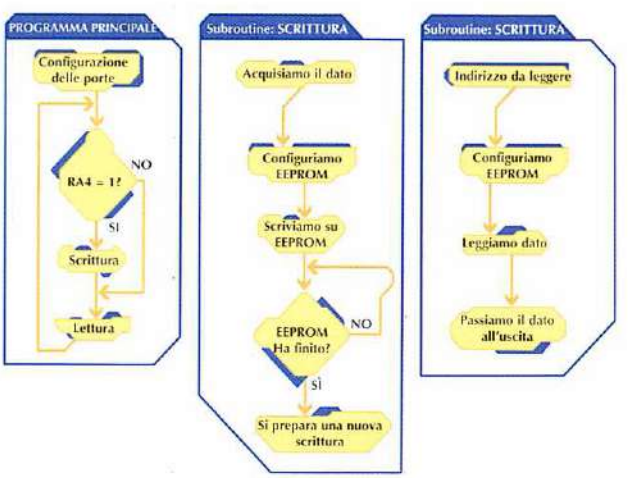
Codice

Programma principale

Cominceremo a programmare con i commenti dell'intestazione del programma, nel quale spiegheremo la funzionalità di quest'ultimo. Proseguiremo con la definizione del PIC, inclu-



IL PIC16F870 ha tre tipi di memoria.



Organigramma dell'applicazione.

Codice del programma principale.

```
;                       ESERCIZIO: EEPROM
;                       -----------------------

;Programma che rileva il valore dello stato degli interruttori
;RA4-RA0, lo scrive nella EEPROM quando si attiva RA4 e visualizza
;il contenuto della EEPROM sulla barra dei LED della porta B.


        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni


        ORG     0
        GOTO    INIZIO
        ORG     5

INIZIO  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        movlw   b'00000110'
        movwf   ADCON1          ;Configuriamo la Porta A come I/O digitali
        clrf    TRISB           ;Porta B uscita
        movlw   b'00011111'
        movwf   TRISA           ;RA0-RA4 ingressi
        bcf     STATUS,RP0      ;Passiamo nuovamente al banco 0

CICLO   btfsc   PORTA,4         ;Se l'interruttore è attivo memorizziamo
        call    SCRITTURA       ;Salta alla routine di scrittura
        call    LETTURA         ;Salta alla routine di lettura
        goto    CICLO           ;Ciclo infinito
```

Definiamo le due variabili all'inizio del codice.

```
;il contenuto della EEPROM sulla barra dei LED della porta B.


        LIST    P=16F870        ;Definiamo il nostro PIC
        INCLUDE "P16F870.INC"   ;File dei registri interni

EE_Dir  equ     0x20
EE_Dato equ     0x21

        ORG     0
        GOTO    INIZIO
        ORG     5
```

dendo la libreria che contiene la definizione dei registri. Mediante le direttive ORG organizzeremo la memoria di programma e a partire dall'etichetta Inizio, inizieremo a configurare i dispositivi.

Useremo la porta B come uscita, elimineremo quindi qualsiasi valore residuo che ci potrebbe essere mediante l'istruzione clrf PORTB, e tramite il registro TRISB indicheremo che i suoi pin saranno utilizzati come uscite.

La porta A lavorerà come ingresso, quindi configureremo il registro TRISA e definiremo il tipo di ingresso mediante il registro ADCON1. Creeremo un ciclo in cui in funzione del valore del pin RA4, si esegue la chiamata alla subroutine di scrittura e a quella di lettura, o solamente a quella di lettura.

Il programma principale terminerà con questo codice (vedi immagine in alto), ripetendo continuamente il ciclo finale.

Subroutine di scrittura nella EEPROM

Quando RA4 è a 1, livello alto, dobbiamo passare il valore presente sull'ingresso alla memoria EEPROM. Creeremo a questo scopo due variabili, una che contiene il dato da scrivere e l'altra per l'indirizzo dove verrà scritto. Il contenuto di queste due variabili si deve passare ai corrispondenti registri della EEPROM, EEDATA e EEADR. Non bisogna dimenticare che ogni volta che si completa una scrittura su questa memoria, un bit di flag si attiva, per indicarci

```
SCRITTURA
        movf    PORTA,W
        movwf   EE_Dato         ;Acquisiamo lo stato di RA4-0
                                ;e lo passiamo a EE_Dato (registro di appoggio)
        bcf     PIR2,EEIF       ;Resettiamo il flag della EEPROM
        bsf     STATUS,RP1      ;Passiamo al banco 2
        bcf     STATUS,RP0
        movlw   EE_Dir
        movwf   EEADR           ;Indirizzo della EEPROM in cui scrivere
        movf    EE_Dato,W
        movwf   EEDATA
        bsf     STATUS,RP0      ;Passiamo al banco 3
        bcf     EECON1,EEPGD    ;Selezioniamo la EEPROM dei dati
        bsf     EECON1,WREN     ;Abilitiamo la scrittura della EEPROM
        movlw   0x55
        movwf   EECON2
        movlw   0xaa
        movwf   EECON2          ;Sequenza obbligatoria
        bsf     EECON1,WR       ;Iniziamo la scrittura
        bcf     STATUS,RP0
        bcf     STATUS,RP1      ;Torniamo al banco 0
ATTENDI  btfss   PIR2,EEIF      ;Attendiamo il termine della scrittura
         goto    ATTENDI
         return
```

Codice della subroutine di scrittura nella EEPROM.

Codice della subroutine di lettura.

```
LETTURA bsf     STATUS,RP1      ;Passiamo al banco 2
        bcf     STATUS,RP0
        movlw   EE_Dir
        movwf   EEADR           ;Scrive l'indirizzo
        bsf     STATUS,RP0      ;Passiamo al banco 3
        bcf     EECON1,EEPGD    ;Selezioniamo la EEPROM dei dati
        bsf     EECON1,RD       ;Abilitiamo la lettura della EEPROM
        bcf     STATUS,RP0      ;Torniamo al banco 2
        movf    EEDATA,W        ;Leggiamo il dato
        bcf     STATUS,RP1      ;Torniamo al banco 0
        movwf   PORTB           ;Visualizziamo il dato letto sui LED
        return
```

che il processo è terminato. Questo bit deve essere resettato cioè impostato a zero, ogni volta che entriamo in questa subroutine.

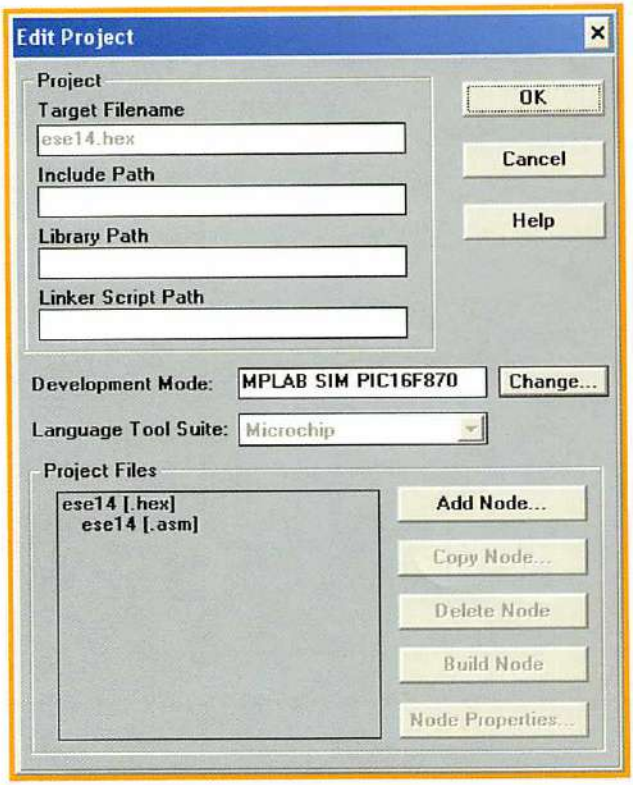
Configureremo la memoria mediante il registro EECON1, per selezionare la EEPROM dei dati e abilitare la scrittura in essa, e in seguito configureremo il registro EECON2 con la sequenza di istruzioni da utilizzare per lavorare con questa memoria in modo scrittura. Questa sequenza sarà sempre la stessa e dovrà essere utilizzata sempre in questo modo.

Siamo pronti per scrivere, inizieremo quindi impostando a 1 il bit WR del registro EECON1. Dato che questa operazione impiega un certo tempo, resteremo all'interno di un ciclo in attesa che il flag che indica il termine di questo processo passi a 1.

Subroutine di lettura

Capita la subroutine precedente, che comprende la lettura del dato nella EEPROM e le operazioni sull'uscita, risulta molto più semplice. Per leggere un dato della memoria la prima cosa da sapere è la posizione dove si trova questo dato. Sul registro EEADR caricheremo l'indirizzo che si trova nella variabile creata espressamente per questo scopo. Configureremo la memoria come nella subroutine precedente, ma con la differenza che ora vogliamo leggere in essa, dopo attiveremo il bit RD del registro EECON1. Il dato si troverà sul registro EEDATA, quindi lo sposteremo sul registro di lavoro W.

Fatto questo abbiamo già a disposizione il



Raccomandiamo di dare al progetto lo stesso nome del codice in assembler.

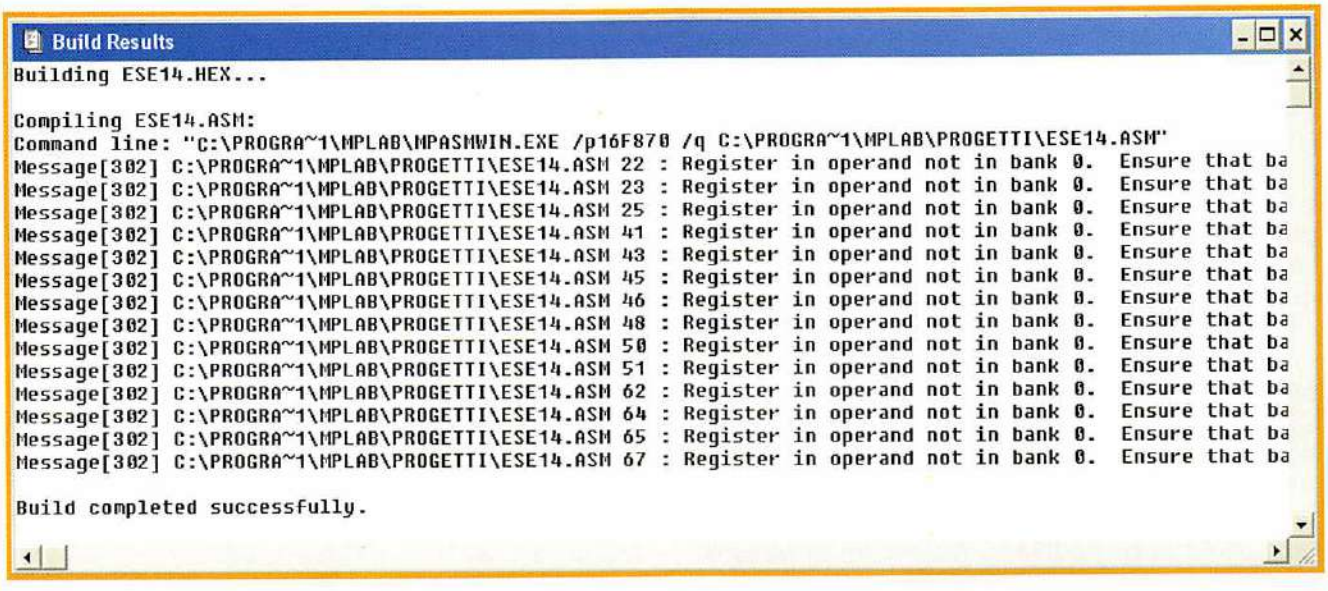
dato, ora lo possiamo utilizzare come desideriamo. Nel nostro caso lo passeremo semplicemente sulla porta di uscita, la porta B.

Il codice non è ancora terminato, infatti bisogna ancora indicare il punto in cui deve finire. Inseriamo la direttiva END al termine del codice per poterlo dichiarare terminato.

Compilazione

Apriamo MPLAB e inseriamo il nostro codice in un progetto. È sempre consigliabile che il nome del progetto coincida con quello del codice, in questo modo eviteremo delle confusioni. Compiliamo l'insieme per poter ottenere il file in codice macchina necessario per scrivere il PIC. Se il programma non ha errori verrà creato il file ".hex" desiderato, nel caso ce ne fossero sarà necessario mettere a punto il programma risolvendoli uno per uno. Come potete vedere nella figura in basso, il codice si assembla e si compila con successo. Il compilatore ci informa, tramite dei messaggi, che i banchi di memoria in cui abbiamo localizzato parte dei registri, non concordano con quelli di default. Questo perché il compilatore utilizza la struttura di memoria del PIC 16F84, e non coincide con quella del nostro (il PIC 16F84 ha solamente due banchi di memoria).

Questi messaggi sono unicamente a scopo informativo e non impediscono che la compilazione avvenga con successo, ma devono essere presi in considerazione se per qualche motivo, legato ad altri aspetti della programmazione, dovessimo fare riferimento alla localizzazione dei registri sui banchi di memoria.



Risultato della compilazione.

Esercizio 16: l'importanza di SLEEP e del WDT, il programma

Facciamo un piccolo salto negli esercizi per presentarne uno molto semplice che illustra in modo eccezionale l'importanza del Watch-Dog Timer (WDT) combinato con l'istruzione Sleep. Questo esercizio si trova risolto nel secondo CD allegato all'opera, con il nome "ese16.asm".

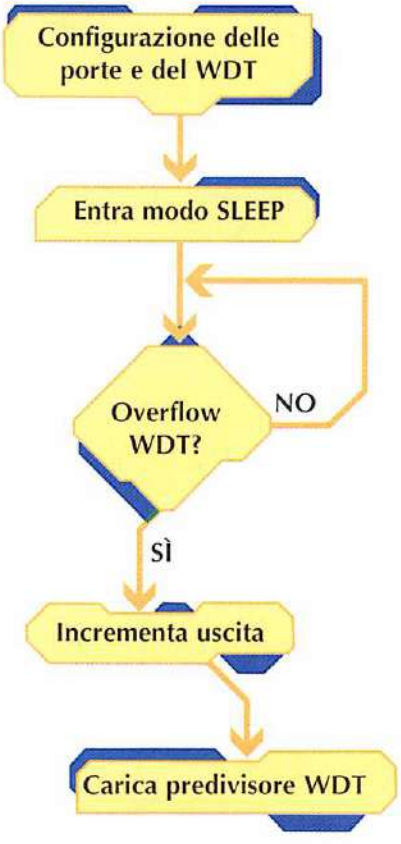
Enunciato

Mediante l'istruzione SLEEP vogliamo impostare il PIC in modo standby di basso consumo, dal quale uscirà ogni volta che il WDT va in overflow. In questo momento si genera un incremento del valore della porta B che funzionerà come contatore binario e nuovamente tornerà nella situazione standbay.

Il predivisore che si associa al WDT sarà compreso fra 1 e 128, in base allo stato logico degli interruttori RA0-RA2. Il valore nominale del WDT è di 18 ms. Cioè, con un predivisore di 1, il PIC si risveglierà ogni 18 ms, con un predivisore di 128 lo farà ogni 2,3 secondi.

Organigramma

L'enunciato specifica chiaramente ciò che si vuole dal programma, quindi risulta molto semplice predisporre una soluzione generale dell'esercizio. Nell'immagine si può vedere l'organigramma che corrisponde all'enunciato.



Organigramma dell'applicazione.

Codice

Iniziamo, come d'abitudine, con i commenti che spiegano il funzionamento del program-

```
;ESERCIZIO: Il modo "sleep" e il "wake-up" (risveglio) mediante il watch-dog Timer (WDT)
;-----------------------------------------------------------------------------
;
;Questo esempio vuole illustrare l'utilizzo dell'istruzione SLEEP per impostare il PIC in
;modo standby a basso consumo. La riattivazione del medesimo avviene ogni volta che il WDT
;va in overflow. In quel momento si produce un incremento di valore della porta B che funziona
;come un contatore binario dopodiché il PIC tornerà nuovamente alla situazione di standby.
;
;Il prescaler si associerà al WDT e sarà compreso fra 1 e 128, in base allo stato
;logico degli interruttori RA0-RA2.
;
;Il valore nominale del WDT è di 18 ms. Quindi con un prescaler di 1, il PIC si "risveglierà"
;ogni 18 ms, con un prescaler da 128, lo farà ogni 2,3 secondi.

            List    p=16F870        ;Tipo di processore
            include "P16F870.INC"   ;Definizione dei registri interni

            org     0x00            ;Vector di Reset
            goto    Inizio
            org     0x05            ;Salva il vector di interrupt
```

Inizieremo tutti i programmi con dei commenti, le definizioni del PIC e le direttive di organizzazione.

Dispositivi	Registri da configurare
Porta A	TRISA, ADCON1
Porta B	TRISB
Watch-Dog timer	OPTION_REG

Dispositivi con cui lavoreremo.

ma. Per questi esercizi, vi consigliamo di copiare l'enunciato così com'è all'interno del file di testo in cui sviluppiamo il nostro codice. Questi non saranno gli unici commenti all'interno del programma, dato che un buon progettista utilizza i commenti praticamente su tutte le linee di codice, per spiegarne in dettaglio il funzionamento.

Definiamo il microcontroller e la libreria che contiene la definizione dei registri e mediante le direttive ORG organizziamo all'interno della memoria il codice che scriveremo.

La prima parte del codice, a partire dall'etichetta Inizio, contiene la configurazione dei dispositivi che vogliamo utilizzare. Impostiamo a 0 tutti i bit del registro TRISB; per configurare la porta B come uscita, configureremo la porta A come ingresso, impostando a 1 i cinque bit meno significativi del registro TRISA, configureremo gli ingressi come digitali mediante il registro ADCON1 e mediante il registro OPTION_REG configureremo il WDT.

Modo SLEEP o di basso consumo

Inizieremo il ciclo impostando un'etichetta a cui poterci indirizzare al termine di questo blocco di programmazione. La prima istruzione del ciclo sarà l'istruzione SLEEP. Questa istruzione manda il microcontroller in uno stato di riposo o di basso consumo. Si disattivano parte dei dispositivi in modo che il PIC rimanga con il minimo necessario per potersi risvegliare. Si tratta di uno stato simile a quello di ibernazione o sospensione in cui entra un PC e dal quale esce solamente se si produce un evento esterno. Nel PIC, per uscire dal modo riposo, in altre parole per risvegliarlo, possiamo utilizzare il WDT, oppure un reset o un interrupt esterno. In molte applicazioni non è necessario che il PIC funzioni costantemente poiché rimane in attesa di un evento esterno (come ad esempio l'inserimento di un codice per accedere a una zona determinata). In queste applicazioni è molto importante utilizzare correttamente il modo SLEEP.

PS2:PS0	Range del WDT
000	1:1
001	1:2
010	1:4
011	1:8
100	1:16
101	1:32
110	1:64
111	1:128

Range del predivisore del WDT in funzione dei bit OPTION_REG <2:0>.

Programmazione del ciclo di SLEEP

Dopo aver configurato tutti i dispositivi, creiamo un ciclo assegnandogli un'etichetta a cui poter saltare in seguito. Il ciclo inizia con l'istruzione SLEEP, con la quale mandiamo il PIC in modo di basso consumo; in questo modo il microocontroller non fa nulla, si trova addormentato, e uscirà dal suo letargo solamente quando il WDT va in overflow. In questo mo-

```
Inizio          clrf    PORTB           ;Cancella i latch di uscita
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00011111'
                movwf   TRISA           ;RA0-RA4 ingressi
                movlw   b'00000110'
                movwf   ADCON1
                movlw   b'00001000'
                movwf   OPTION_REG      ;Prescaler da 1 per il WDT
                bcf     STATUS,RP0      ;Seleziona il banco 0
```

Codice di configurazione dei dispositivi.

```
Loop        sleep                   ;Modo Standby

            incf    PORTB,F         ;Incrementa il contatore binario sulla porta B
            movf    PORTA,W
            andlw   b'00000111'     ;Legge lo stato degli interruttori RA0-RA2
            iorlw   b'00001000'
            bsf     STATUS,RP0      ;Seleziona il banco 1
            movwf   OPTION_REG      ;Regola il valore del prescaler
            bcf     STATUS,RP0      ;Seleziona il banco 1
            goto    Loop            ;Torna al modo Standby

            end                     ;Fine del programma sorgente
```

Codice del ciclo di SLEEP.

Selezionando New Project creiamo un nuovo progetto.

mento dobbiamo incrementare il valore da visualizzare tramite i LED collegati alla porta B, che non è altro che il numero di volte che il PIC è uscito dal modo riposo. Per preparare un nuovo SLEEP prendiamo il valore del predivisore del WDT, che viene caricato sulla porta A e lo scriviamo sul registro OPTION_REG, saltando poi all'inizio del ciclo per ripetere tutto il processo.

Compilazione

Abbiamo terminato quello che pensiamo sia il codice dell'applicazione, però per accertarci che quello che abbiamo progettato risponda all'enunciato, dobbiamo passare alla fase di compilazione, simulazione e montaggio in modo soddisfacente. Faremo partire MPLAB e creeremo un nuovo progetto. Ricordate che è conveniente utilizzare lo stesso nome sia per il progetto che per il codice, o dare un nome al progetto che indichi in modo esplicito ciò che esso sviluppa. Nel nostro caso possiamo chiamare il progetto "ese16.pjt" o "sleep.pjt". Creando un nuovo progetto si apre la finestra di edizione di quest'ultimo e tramite questa finestra assoceremo il file che contiene il codice.

Apriamo il file per visualizzare il codice sul monitor e compiliamo il progetto selezionando l'opzione Build All. Se la compilazione avviene correttamente, apparirà la videata riportata nella pagina successiva in cui veniamo informati che la compilazione è stata eseguita con successo (Build completed successfully).

Simulazione

Per simulare questa applicazione apriremo le finestre dei Registri delle Funzioni Speciali, una finestra con i registri più importanti (nel nostro caso sarà sufficiente visualizzare la por-

```
Build Results
Building ESE16.HEX...

Compiling ESE16.ASM:
Command line: "D:\PROGRA~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM"
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM 25 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM 27 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM 29 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM 31 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\ESE16.ASM 42 : Register in operand not in bank 0.  Ensure

Build completed successfully.
```

Risultato della compilazione.

ta di uscita PORTB) e la finestra della memoria di programma (Program Memory).

Quando inizieremo la simulazione passo a passo (premeremo F7 e simuleremo linea a linea) il codice si eseguirà correttamente fino ad arrivare all'istruzione SLEEP. A questo punto si ferma l'esecuzione, dato che il microcontroller entrerà in modo riposo, infatti anche se adesso premiamo molte volte F7 continuerà a "dormire".

Possiamo vedere che quando simuliamo la linea di esecuzione viene evidenziata con una striscia nera, e questo si verifica sia nella finestra dei file in assembler che nella finestra di memoria del programma. Per uscire dallo SLEEP dopo esservi entrati con la simulazione, faremo un doppio clic con il pulsante sinistro del mouse sulla linea 17 della memoria di programma, la linea che si trova evidenziata col colore rosso e la cui istruzione è "incf 0x6".

Con questo doppio clic, la linea di stato di MPLAB passerà a colore giallo (linea inferiore della finestra), per alcuni secondi. Automaticamente sparisce il colore giallo e premendo F7 potremo continuare con la simulazione.

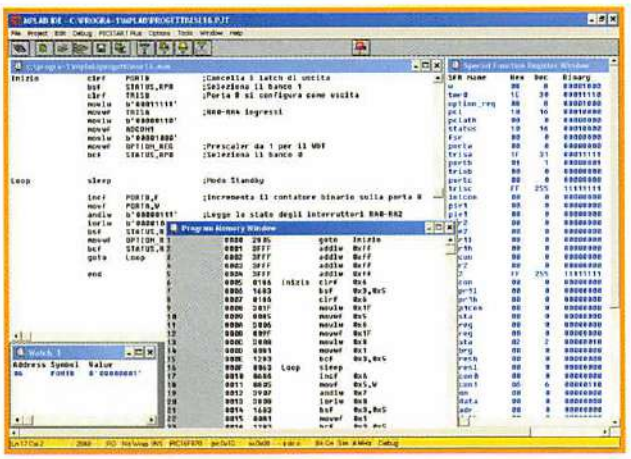
Nel caso rimanessimo bloccati, potremo cliccare con il mouse sulla linea della memoria di programma che desideriamo, per fare un salto nella simulazione.

La porta B si incrementa correttamente ogni volta che usciamo dallo SLEEP, quindi la simulazione può essere considerata corretta e terminata.

Program Memory Window

1		0000	2805		goto	Inizio
2		0001	3FFF		addlw	0xff
3		0002	3FFF		addlw	0xff
4		0003	3FFF		addlw	0xff
5		0004	3FFF		addlw	0xff
6		0005	0186	Inizio	clrf	0x6
7		0006	1683		bsf	0x3,0x5
8		0007	0186		clrf	0x6
9		0008	301F		movlw	0x1F
10		0009	0085		movwf	0x5
11		000A	3006		movlw	0x6
12		000B	009F		movwf	0x1F
13		000C	3008		movlw	0x8
14		000D	0081		movwf	0x1
15		000E	1283		bcf	0x3,0x5
16		000F	0063	Loop	sleep	
17	B...	0010	0A86		incf	0x6
18	B...	0011	0805		movf	0x5,W
19		0012	3907		andlw	0x7
20	B...	0013	3808		iorlw	0x8
21		0014	1683		bsf	0x3,0x5
22	B...	0015	0081		movwf	0x1
23		0016	1283		bcf	0x3,0x5
24		0017	280F		goto	Loop
25		0018	3FFF		addlw	0xff
26		0019	3FFF		addlw	0xff
27		001A	3FFF		addlw	0xff
28		001B	3FFF		addlw	0xff
29		001C	3FFF		addlw	0xff
30		001D	3FFF		addlw	0xff

Finestra della memoria di programma.



La linea di stato cambia nel colore giallo.

Esercizio 15: la USART, il programma

L'ultimo degli esercizi previsti per consolidare le conoscenze della teoria, tratta la comunicazione seriale con il modulo USART.
Questo dispositivo ci permette di comunicare con altri dispositivi, ed è molto utilizzato per far comunicare il PIC con un PC.

Comunicazione PIC-PC

In molte applicazioni si utilizza un PC per eseguire un controllo o la supervisione dell'applicazione, però è il PIC che ha il compito di agire sul sistema. Per poter mettere in comunicazione il PIC con il PC utilizzeremo la comunicazione seriale RS 232 e a questo scopo il PIC deve utilizzare il suo modulo di comunicazione seriale USART nel modo asincrono. Stabilita la comunicazione potremo trasferire informazioni fra entrambe le parti, programmare il PIC e anche aggiornare un programma senza doverlo togliere dal circuito dove si trova (quest'ultima operazione grazie a programmi residenti come Bootloader o Uploader che spiegheremo più avanti).

Enunciato

Si tratta di progettare un programma che legga lo stato logico di interruttori corrispondenti a RA4-RA0 e lo trasmetta via seriale a Hyper Terminal di Windows.

Nel caso in cui l'applicazione richieda comunicazione con un PC utilizzeremo il modulo USART nel suo modo di trasmissione asincrono.

Conoscendo la velocità a cui dobbiamo stabilire la comunicazione (9.600 **Baud**) possiamo configurare la trasmissione secondo la formula riportata nella figura.

Organigramma

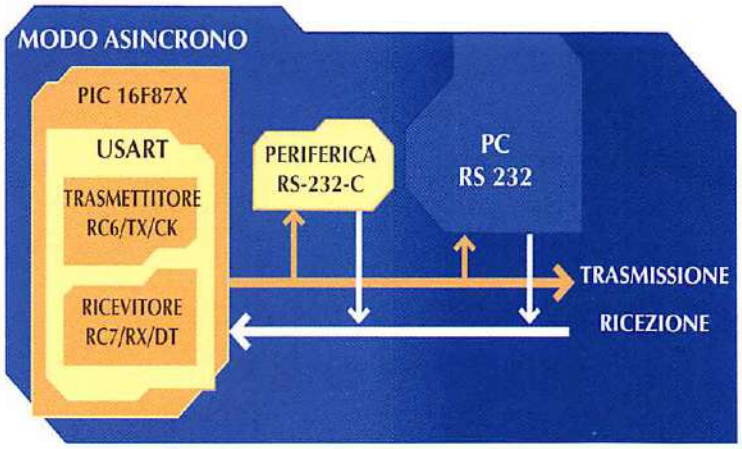
In un programma con queste caratteristiche conviene separare la parte del codice in subroutine, dato che esisteranno alcuni blocchi di codice che dovranno realizzare delle funzioni specifiche. Vi consigliamo di separare questi blocchi per svilupparli in modo indipendente.

Nell'organigramma della figura è rappresentata una soluzione generale all'esercizio, che ovviamente dovremo sviluppare con attenzione al momento di progettare il codice.

Codice

Intestazione e configurazione dei dispositivi
I commenti ci permetteranno di definire la funzionalità del programma, il PIC, la libreria delle definizione dei registri e grazie alle direttive ORG, la zona di memoria dove risiede il codice di programma.

Il programma principale inizia con l'etichetta Inizio configurando i dispositivi con cui vogliamo lavorare. La porta A sarà di ingressi digitali,



Schema di comunicazione del modulo USART modo asincrono.

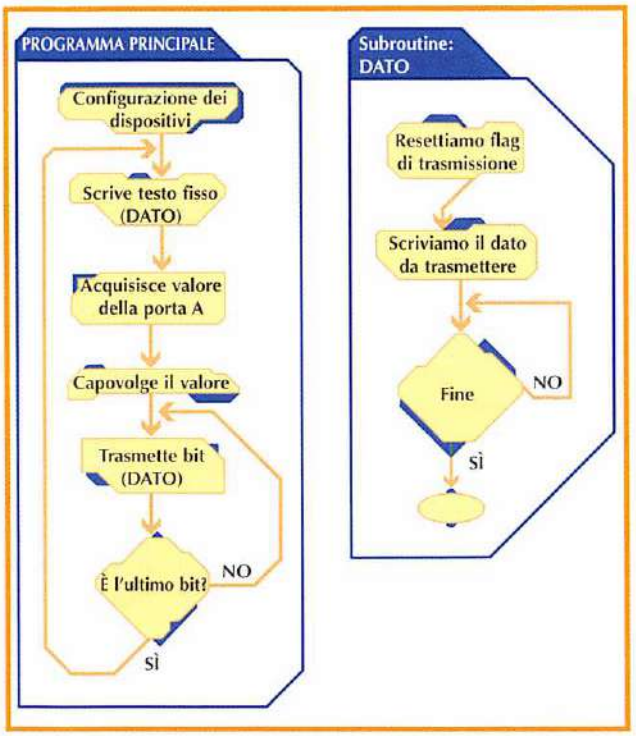
$$Frec = \frac{Fosc}{k\cdot(x+1)}, \text{ dove } K \text{ vale } 16 \text{ o } 64$$

Per calcolare il valore di x, sappiamo che la frequenza di comunicazione sarà Frec = 9600 Baud e che la frequenza di oscillazione del PIC è Foxc = $4\cdot10^6$ Hz

$$x = \left(\frac{Fosc}{k\cdot Frec}\right) - 1 = \left(\frac{4\cdot10^6}{16\cdot9600}\right) - 1 = 26 - 1 = 25$$

questo è il valore che verrà caricato sul registro SPBRG di configurazione.

Formule per configurare la comunicazione.



Organigramma dell'applicazione.

quindi dovremo configurare i registri TRISA e ADCON1. Nella porta C troviamo le linee di trasmissione RC6/Tx e RC7/Rx, essendo quest'ultima un ingresso configureremo il registro TRISC tenendo presente quest'ultima caratteristica. Configureremo il WDT mediante il registro OPTION_REG, infine determineremo la trasmissione. La prima cosa che dobbiamo fare è delimitare il tipo di trasmissione (modo asincrono ad alta velocità e da 8 bit) scrivendo il registro TXSTA, definire la velocità di trasmissione (9.600 **Baud**), caricando il valore ottenuto dalle formule sul registro SPBRG, e abilitare la trasmissione seriale impostando a 1 il bit SPEN del registro RCSTA.

Subroutine DATO di trasmissione dei caratteri

La trasmissione seriale si esegue byte a byte, quindi se vorremo inviare un testo lo dovremo fare inviando carattere per carattere. Dato che nella nostra applicazione vogliamo visualizzare su Hyper Terminal lo stato dei pin RA4:RA0 dovremo inviare come minimo 5 byte, uno per ogni carattere corrispondente allo stato di ogni pin di ingresso (carattere '0' o '1'). Sarà possibile anche migliorare la presentazione scrivendo un testo che preceda il dato in questione (ad esempio: porta A: xxxxxx) quindi sarà necessario trasmettere più byte dei minimi richiesti. Dato che dovremo ripetere il processo per ogni carattere da inviare, cioè molte volte, lo separeremo in una subroutine.

Ogni volta che entreremo nella subroutine dovremo resettare il flag o l'indicatore della trasmissione precedente, che viene attivato al termine di ogni trasmissione. Dovremo quindi passare il file da trasmettere al registro TXREG, e in seguito attendere che la trasmissione termini verificando il bit TRMT del registro TXSTA. Nella figura è possibile vedere il codice che corrisponde a questa subroutine.

Continuiamo con il programma principale

Abbiamo configurato i dispositivi e creato una subroutine di trasmissione, quindi ora dobbia-

```
;
;                          ESERCIZIO: LA USART
;                          -------------------
;
;Il programma sarà un'applicazione del modo di trasmissione asincrono (RS-232).
;Programma che legge lo stato logico degli interruttori corrispondenti a RA4-RA0,
;trasmette questo stato logico via seriale al software hiperterminal di windows.


        LIST    P=16F870        ;Definiamo il nostro microcontroller
        INCLUDE "P16F870.INC"   ;File dei registri interni

        ORG     0
        GOTO    INIZIO
        ORG     5
```

Intestazione del programma.

```
;Programma principale.

INIZIO          clrf    PORTC           ;Resettiamo la Porta C
                bsf     STATUS,RP0      ;Passiamo al banco 1
                movlw   b'00011111'
                movwf   TRISA
                movlw   b'00000110'
                movwf   ADCON1
                movlw   b'10111111'
                movwf   TRISC           ;RC6/Tx come uscita e RC7/Rx come ingresso
                movlw   b'11001111'
                movwf   OPTION_REG      ;WDT
                movlw   b'00100100'
                movwf   TXSTA           ;Attiviamo la trasmissione in modo
                                        ;asincrono a 8 bit ad alta velocità
                movlw   .25
                movwf   SPBRG           ;Carichiamo il registro generatore della velocità
                                        ;in baude in modo che si configuri con 9600
                bcf     STATUS,RP0      ;Torniamo al banco 0
                bsf     RCSTA,SPEN      ;Abilitiamo la porta seriale
```

Configurazione dei dispositivi.

```
;Trattamento della routine di trasmissione del dato.

DATO            bcf     PIR1,TXIF       ;Resettiamo il flag del trasmettitore
                movwf   TXREG           ;Scriviamo il dato
                bsf     STATUS,RP0      ;Passiamo al banco 1
CICLO1          btfss   TXSTA,TRMT      ;Verifichiamo se la trasmissione è terminata
                goto    CICLO1          ;No, non è ancora terminata
                bcf     STATUS,RP0      ;Torniamo al banco 0
                return
```

Inviamo un testo tramite la USART.

```
        LIST    P=16F870        ;Definiamo il nostro microcontroller
        INCLUDE "P16F870.INC"   ;File dei registri interni

aux1    equ     0x20
aux2    equ     0x21
aux3    equ     0x22

        ORG     0
        GOTO    INIZIO
        ORG     5
```

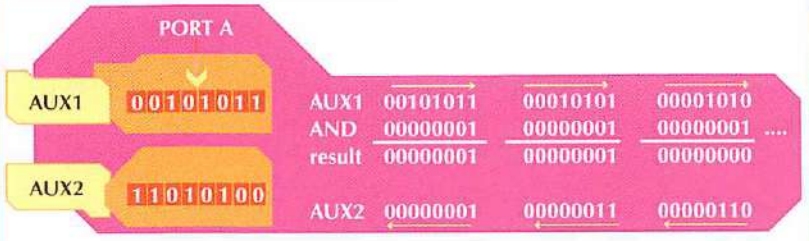
Definiamo tre variabili.

mo definire ciò che vogliamo trasmettere e come farlo.

La prima cosa dao fare è inviare il testo "Porta A:" e come abbiamo appena visto lo dobbiamo fare carattere dopo carattere. Esistono una serie di comandi speciali che Hyper Terminal riconosce come ordini, quindi '/r' è un salto di linea (Return), '/t' è una tabulazione, ecc. Dobbiamo includere alcuni di questi comandi per migliorare la visualizzazione. Passeremo ogni carattere o comando da inviare sul registro di lavoro e chiameremo la subroutine di trasmissione, come si può vedere dal codice della figura. Acquisiremo lo stato degli ingressi e scriveremo il loro valore su una variabile ausiliaria. Dobbiamo trasferire i 5 bit meno significativi su RA4:RA0 uno per volta, iniziando da quello che corrisponderà a RA4 fino ad arrivare a RA0. Prima di mandarli sarà quindi necessario riordinare i bit.

Progettiamo ora una nuova subroutine che isoli il bit e lo porti riordinato su una nuova variabile ausiliaria. In questa subroutine entreremo cinque volte per ogni acquisizione della

```
                movlw   ':'
                call    DATO
                movlw   ' '
                call    DATO

                movfw   PORTA
                movwf   aux1

                clrf    aux2
                CALL    TRANSFER

                movlw   .4
                movwf   aux3
TRANS           rlf     aux2,f
                rrf     aux1,f
                CALL    TRANSFER
                decfsz  aux3,f
                goto    TRANS
```



Procedimento di riordinazione

```
;Routine per isolare il bit nel processo di riordino

TRANSFER        movfw   aux1
                andlw   1               ;Isolo il bit
                bcf     aux2,0          ;Imposto a 0
                iorwf   aux2,f          ;Sommo
                return
```

Codice per riordinare i bit.

```
                decfsz  aux3,f
                goto    TRANS

                movlw   .5
                movwf   aux3
RIPETE
                movfw   aux2
                call    BINTOASCII      ;Convertiamo il bit 0 in codice ASCII
                call    DATO            ;Inviamo il dato
                rrf     aux2,f          ;Bit successivo
                decfsz  aux3,f
                goto    RIPETE
                goto    CICLO
```

```
;Routine per convertire il valore del bit in ASCII

BINTOASCII      andlw   1               ;Isoliamo il bit 0
                addlw   '0'             ;Sommiamo il codice ASCII del carattere '0'
                return
```

Codice per l'invio del valore dell'ingresso.

porta, quindi creeremo una nuova variabile che funzioni da contatore (aux3). Definiamo queste variabili all'inizio del programma.

Per isolare i bit e riordinarli dobbiamo utilizzare le istruzioni di rotazione, in modo che il bit da isolare sia quello meno significativo, moltiplicare la variabile per b'00000001', passare il risultato alla variabile che contiene il valore ordinato sommandola a questa, e ruotare a sinistra il suo contenuto in modo che ripetendo l'operazione sia possibile alloggiare un nuovo bit nella posizione desiderata (vedere il grafico della figura). Tutto questo si traduce in codice nel modo indicato dalle immagini.

Dopo aver ordinato il valore acquisito lo dovremo trasferire bit a bit e a questo scopo ruoteremo il registro a destra isolando ogni bit. Il bit isolato avrà valore 0 o 1, ma dato che ciò che vogliamo trasferire è un carattere, ne cambieremo il valore in ASCII mediante una subroutine, in modo che possa essere inviato. Ruoteremo il registro 5 volte, dato che 5 sono i bit da trasmettere e ripeteremo il processo dalla trasmissione del testo iniziale.

Nella figura possiamo vedere il codice che corrisponde a quest'ultima parte del programma e la subroutine per convertire il bit in ASCII. Non dimenticate di terminare il programma con la direttiva END.

Il programma completo si trova sul secondo CD allegato all'opera con il nome di "ese15.asm".

Esercizio: controllo di un forno

Presentiamo ora un progetto reale in cui sono utilizzati molti dei dispositivi del PIC16F870, come ad esempio le memorie FLASH e EEPROM interne, il convertitore A/D, interrupt, timer e le porte di comunicazione seriale (USART). Utilizzando i microcontroller potremo risolvere e controllare una moltitudine di processi industriali in modo semplice, pratico ed economico.

Controllo della temperatura per il modellamento dei pezzi

Vogliamo realizzare il controllo di un processo in cui si riscaldano i pezzi in un forno per il loro successivo modellamento.

Lavoreremo con due tipi di pezzi:

– Pezzi A: questi pezzi potranno essere modellati se rimarranno nel forno per 5 secondi a una temperatura superiore ai 125° (Forno 75-125.asm).

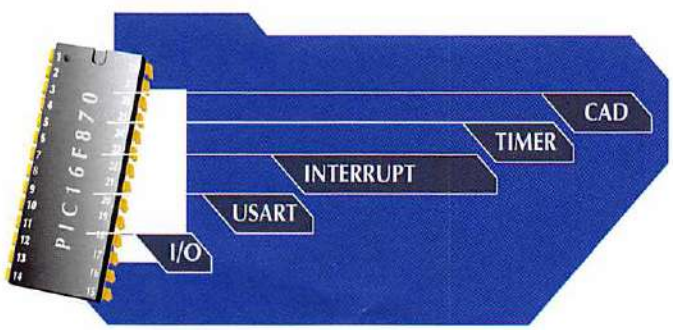
– Pezzi B: questi pezzi potranno essere modellati se rimarranno nel forno per 5 secondi a una temperatura superiore a 150° (Forno 100-150.asm).

In base ai pezzi che si trovano sulla catena dobbiamo eseguire un controllo oppure l'altro.

Per i pezzi A il nostro forno ci avviserà che ci stiamo avvicinando alla temperatura specificata (125°) dopo aver superato i 75° C. In questo modo l'addetto potrà essere avvisato e preparare quanto serve per il processo successivo.

L'allarme sparisce quando la temperatura non è superiore ai 75°. Se al contrario la temperatura ha raggiunto quella specificata, sparisce l'allarme e viene indicato che la temperatura è quella corretta. Se questo stato si mantiene per 5 secondi il pezzo verrà considerato riscaldato e si introdurrà all'interno del forno il pezzo successivo.

Per i pezzi B il processo è identico ma cambiano le temperature. Essendo di materiale diverso, per il loro successivo modellamento, hanno bisogno di rimanere nel forno per 5 secondi a temperatura superiore a 150°, generando l'allarme di avviso all'addetto quando la temperatura del forno supera i 100°.



Dispositivi utilizzati dal PIC.

Esecuzione del progetto

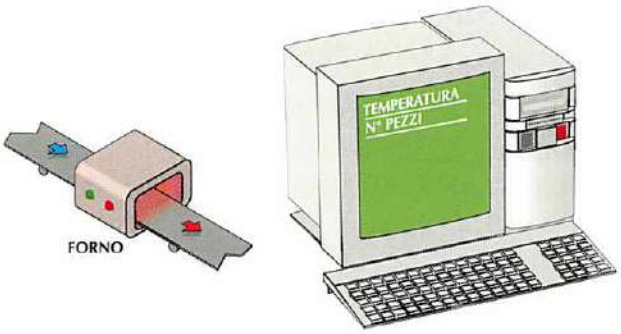
Per la realizzazione di questo progetto abbiamo a disposizione una sonda attiva di temperatura all'interno del forno. Il forno avrà un range di temperatura fra 0° e 255° C, e l'uscita proporzionale che otterremo dalla sonda sarà compresa tra 0 e 5 Volt.

Questa sonda potrà essere simulata con un potenziometro.

L'uscita di questa sonda sarà interpretata dal microcontroller che convertirà questo valore in binario, per poterlo elaborare (convertitore A/D). Ogni volta che eseguiamo una conversione genereremo un interrupt in cui verificheremo il valore ottenuto.

Se il valore è compreso all'interno del range di allarme, indicheremo questa situazione facendo lampeggiare un diodo LED. Otterremo questo utilizzando il Timer0. Se il valore è compreso all'interno del range corretto di riscaldamento del pezzo, utilizzeremo lo stesso indicatore ma questa volta fornendo una luce fissa.

Verificheremo che il pezzo rimanga in quella situazione per 5 secondi e dopo aver soddisfatto questa condizione inseriremo un nuovo pezzo nel forno. Il movimento del nastro trasportatore lo indicheremo con un altro diodo



Processo industriale di modellamento dei pezzi.

Visualizzazione sul PC delle variabili da controllare.

LED e sarà prodotto da un motore collegato indirettamente (tramite una circuiteria addizionale) a una uscita del microcontroller.

Per simulare il montaggio collegheremo solamente il LED all'uscita.

Dovremo inoltre indicare il numero di pezzi eseguiti e la temperatura di lavoro attuale del forno. Mediante la comunicazione seriale del microcontroller (modulo USART) invieremo al terminale di un PC questi valori. In questo modo potremo osservare il funzionamento del processo in tempo reale tramite una postazione remota.

Utilizzando il Bootloader, file che si trova nelle directory (Bootloader.ascu) che porta questo nome inserita nel primo CD-R, potremo, tramite lo stesso programma e utilizzando lo stesso tipo di comunicazione, selezionare il processo da realizzare. Potremo scrivere il nuovo programma da eseguire per il microcontroller, in base ai pezzi che si desidera automatizzare. Fermando il processo e facendolo ripartire tenendo premuto il reset, il microcontroller si ferma e attende che venga caricato un nuovo programma, spegnendo e alimentando nuovamente il circuito manderemo in funzione il programma dell'utente per il processo scelto.

Il Bootloader e le relative modifiche ai programmi per poter lavorare con esso, verranno presentate nei prossimi fascicoli.

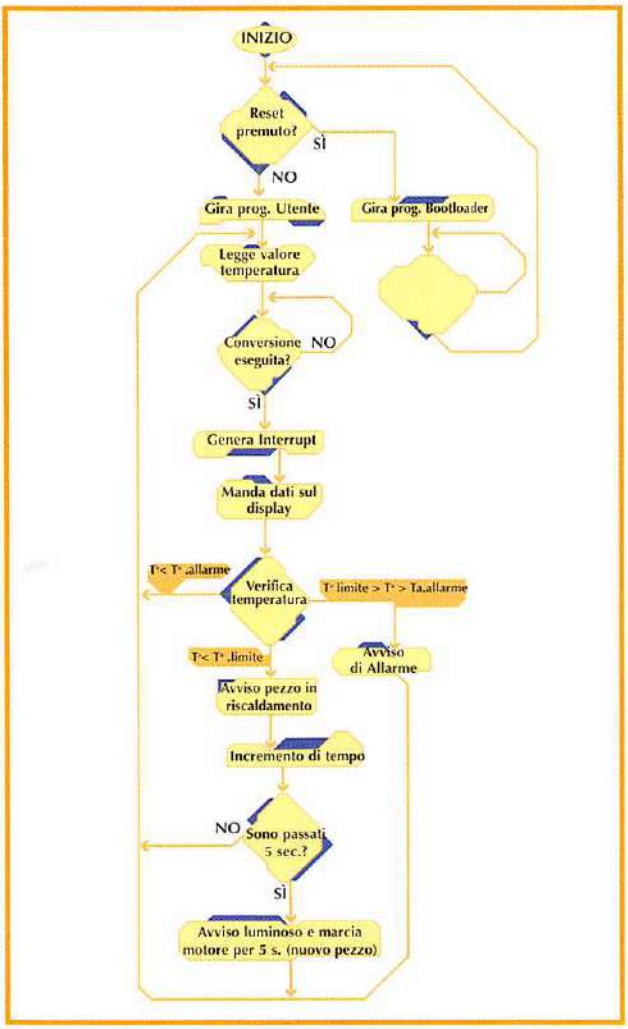
Organigramma

Mediante un organigramma pianificheremo una soluzione generale al progetto e partendo da essa svilupperemo il codice più facilmente.

Codice

Per spiegare come è stata risolta l'applicazione ci concentreremo su uno dei programmi, dato che l'altro è molto simile, con l'unica differenza dei valori di temperatura. Risolveremo quindi il programma per i pezzi A, e spiegheremo i cambiamenti da eseguire, partendo da questo, per fare il programma corrispondente ai pezzi di tipo B.

Il codice inizia con i commenti relativi al programma, le definizioni del PIC e la libreria dei registri. Possiamo definire le costanti per i valori di temperatura. Così 125 sarà '01111101' in binario e 75 sarà '01001011'. Seguiranno la disposizione per gli indirizzi della memoria di programma e il vector di interrupt mediante le direttive ORG.



Organigramma del progetto.

```
;       ESERCIZIO: Forno 75-125
;      -------------------------
;
        LIST P=16F870
        include "P16f870.INC"

volt3           EQU     b'01111101'     ;Raggiunti i 125° fissa l'uscita e inizia la sequenza
volt2           EQU     b'01001011'     ;Raggiunti i 75° ci avvisa della temperatura


;Inizio del programma
        org 0
        GOTO INIZIO

        org 4
        goto INT

        org 5
```

Intestazione del programma.

```
INIZIO  clrf    PORTA
        clrf    PORTB
        clrf    PORTC           ;Resettiamo la Porta C
        clrf    n_pezzi
        ;Configuriamo ADCON con: RC, canale2, Attivazione del "modulo del convertitore"
        ;Se non lo programmiamo con RC non lo potremo risvegliare dallo SLEEP!
        movlw   b'11000001'
        movwf   ADCON0
        ;Configuriamo ADCON con giustificazione a sinistra->lo carichiamo in ADRESH+tutto analogico;
        banksel ADCON1
        movlw   b'00001110'
        movwf   ADCON1
        clrf    TRISB           ;Configuriamo la Porta B come uscita digitale
        movlw   b'11001111'     ;Sul registro OPTION WDT=1/128
        movwf   OPTION_REG
        movlw   b'00000001'     ;Porta A: RA0 ingresso analogico, gli altri uscite
        movwf   TRISA
        movlw   b'10111111'
        movwf   TRISC           ;Impostiamo RC6/Tx come uscita e RC7/Rx come ingresso
        movlw   b'00100100'
        movwf   TXSTA           ;Attiviamo la trasmissione in modo asincrono, a 8 bit ad alta velocità
        movlw   .25
        movwf   SPBRG           ;Carichiamo il reg. generatore della velocità in baude in modo che si configuri con 9600
        banksel RCSTA
        bsf     RCSTA,SPEN      ;Abilitiamo la porta seriale
        movlw   .325000         ;Con questo valore otterremo i 5 sec.
        movwf   cont
        movwf   cont1

;*****Configurazione dell'interrupt per il salto quando si attiva la conversione
        banksel PIE1
        movlw   b'01000000'
        movwf   PIE1
        banksel INTCON
        movlw   b'11000000'
        movwf   INTCON
conver  bsf     ADCON0,GO
        sleep
        goto conver
```

Configurazione dei dispositivi.

```
;*****************Routine di interrupt*****************

        org 50
        banksel PORTB
INT     goto    CICLO
eti     btfsc   PORTB,5
        goto    temp
        movf    ADRESH,w
        sublw   volt3
        banksel STATUS
        btfss   STATUS,0        ;Se il carry è a zero:
        goto    ALLARME
        movlw   .325000         ;Con questo valore otterremo i 5 sec.
        movwf   cont
        movf    ADRESH,w
        sublw   volt2
        banksel STATUS
        btfss   STATUS,0        ;Se il carry è a zero:
        call    lampeg
        banksel PORTB
        clrf    PORTB
        banksel PIR1
REINIT  bcf     PIR1,6          ;Reinizializziamo il bit dell'interrupt
        retfie

;*******************************************************************************
;Subroutine che mantiene l'uscita RB4 costantemente accesa
ALLARME nop
        banksel PORTB
        bsf     PORTB,4
        decfsz  cont,F
        goto    eti1
        call    motore
temp    decfsz  cont1,F
        goto    eti1
        bcf     PORTB,5
eti1    nop
        banksel PIR1
        goto    REINIT
```

Routine di servizio dedicata all'interrupt.

A partire dall'etichetta Inizio resetteremo le porte e configureremo i dispositivi, le tre porte, il CAD, il WDT, la USART e gli interrupt. Ci fermeremo in modalità SLEEP entrando in un ciclo fino a quando non si esegue la conversione. Definiremo tre nuove variabili all'inizio del programma.

Terminata la conversione si genera un interrupt e si esegue la routine dedicata a quest'ultima. In essa si analizza il valore acquisito, attivando l'uscita nel caso in cui la temperatura sia adeguata, avvertendo con il lampeggio dell'allarme il fun-

```
;*****************************************************
;Subroutine che provoca il lampeggio dell'uscita RB4

lampeg  clrf    PORTB
        bsf     STATUS,RP0      ;Passiamo al banco 1
        clrf    TRISB           ;Porta B uscite
        movlw   b'11010111'     ;divisore 256
        movwf   OPTION_REG
        bcf     STATUS,RP0      ;Torniamo al banco 0

LAMPEGG bcf     PORTB,4         ;Spegniamo il LED (RB4)
        call    RITARDO         ;Saltiamo alla routine di ritardo
        bsf     PORTB,4         ;Accendiamo il LED
        call    RITARDO
        return


RITARDO movlw   b'00000001'     ;valore da contare 255
        movwf   TMR0            ;(complemento a 2)
        btfss   INTCON,T0IF
        goto    $-1             ;Va all'istruzione precedente
        bcf     INTCON,T0IF
        return

;*********************************************************************
;Subroutine che mette in marcia il motore
motore  movlw   .325000
        movwf   cont
        movwf   cont1
        bsf     PORTB,5
        incf    n_pezzi,1
        return
;*********************************************************************
;Subroutine della usart: visualizziamo sul display la temperatura e il numero dei pezzi

VISUAL  bcf     PIR1,TXIF       ;Resettiamo il flag del trasmettitore
        movwf   TXREG           ;Scriviamo il dato
        bsf     STATUS,RP0      ;Passiamo al banco 1
CICL01  btfss   TXSTA,TRMT      ;Verifichiamo se la trasmissione è terminata
        goto    CICL01          ;Non è ancora terminata
        bcf     STATUS,RP0      ;Torniamo al banco 0
        return
```

Subroutine con funzioni diverse.

terno dei limiti dell'allarme, utilizzeremo una subroutine, ma ne prepareremo anche un'altra per attivare l'uscita che mette in marcia il motore, e un'altra per configurare la trasmissione tramite USART.

Ricordatevi di definire le variabili necessarie all'inizio del programma.

All'interno del programma principale dovremo inviare i caratteri da visualizzare sul PC, quali la lettura della temperatura e il numero di pezzi prodotti. Eseguiremo una chiamata alla subroutine di trasmissione per ogni carattere. Per trasmettere i dati delle variabili è necessario convertirne il valore da binario a decimale, a questo scopo creeremo una subroutine per eseguire questa operazione.

Potete seguire il codice tramite le immagini di queste pagine oppure utilizzare il file che si trova sul secondo CD-R fornito con l'opera (Forno 75-125.asm). Ricordatevi di dichiarare sempre le variabili che mano a mano si vanno generando. Per i pezzi di tipo B cambierà solamente il valore delle costanti di temperatura, invece di 75 e 125 dovranno essere 100 e 150.

zionamento sull'uscita. Per semplificare il programma lo separeremo in pezzi di codice raggruppati in subroutine. Per il lampeggio del LED, nel caso in cui il valore acquisito sia all'in-

```
CICL0   clrwdt
        movlw   '\r'
        call    VISUAL
        movlw   'T'
        call    VISUAL
        movlw   'e'
        call    VISUAL
        movlw   'm'
        call    VISUAL
        movlw   'p'
        call    VISUAL
        movlw   ':'
        call    VISUAL
        movlw   ' '
        call    VISUAL

        movfw   ADRESH
        movwf   aux11
        call    bin2dec
        movfw   aux21
        addlw   '0'
        call    VISUAL
        movfw   aux31
        addlw   '0'
        call    VISUAL
        movfw   aux41
        addlw   '0'
        call    VISUAL
        clrwdt
        movlw   ' '
        call    VISUAL
        movlw   ' '
        call    VISUAL
        movlw   'P'
        call    VISUAL
        movlw   'e'
        call    VISUAL
        movlw   'z'
        call    VISUAL
        movlw   'z'
        call    VISUAL
        movlw   'i'
        call    VISUAL
        movlw   ':'
        call    VISUAL
        movlw   ' '
        call    VISUAL

        movfw   n_pezzi
        movwf   aux11
        call    bin2dec
        movfw   aux21
        addlw   '0'
        call    VISUAL
        movfw   aux31
        addlw   '0'
        call    VISUAL
        movfw   aux41
        addlw   '0'
        call    VISUAL
        movlw   .10
        movwf   aux51
        movlw   ' '
tdesp   call    VISUAL
        decfsz  aux51,F
        goto    tdesp
        banksel PORTB
        goto    et1
```

Programma principale.

```
;*****************************************  conversione da bin a dec  ****************
bin2dec movwf   aux11
        clrf    aux21

        goto    salto
da100   incf    aux21
        movlw   .100
        subwf   aux11
salto   btfsc   aux11,7
        goto    da100
        btfss   aux11,6
        goto    no100
        btfss   aux11,5
        goto    no100
        btfsc   aux11,4
        goto    da100
        btfsc   aux11,3
        goto    da100
        btfsc   aux11,2
        goto    da100
no100   clrf    aux31
        goto    salto1
da10    incf    aux31
        movlw   .10
        subwf   aux11
salto1  btfsc   aux11,6
        goto    da10
        btfsc   aux11,5
        goto    da10
        btfsc   aux11,4
        goto    da10
        btfss   aux11,3
        goto    no10
        btfsc   aux11,2
        goto    da10
        btfsc   aux11,1
        goto    da10
no10    movfw   aux11
        movwf   aux41
        return

        end
```

Subroutine per convertire un numero binario in decimale.

Le schede Smart Card

Presenteremo ora uno degli elementi più interessanti del nostro laboratorio, le schede Smart Card, anche chiamate schede chip o schede intelligenti. Costruite nell'anno 1983 sono composte da un chip integrato in un rettangolo di PVC di dimensioni standard.

Cosa sono le schede Smart Card?

Una scheda intelligente (o Smart Card) è un dispositivo di sicurezza della dimensione di una carta di credito che offre funzioni quali l'immagazzinamento sicuro di informazioni e l'elaborazione delle stesse in base a tecnologia VLSI. Il chip contenuto nella scheda dispone di alcuni contatti esterni che permettono la comunicazione per accedere all'informazione in esso contenuta o per scrivere una nuova informazione.

I contatti sono laminati in oro per poter avere una maggior resistenza e poter permettere un utilizzo senza problemi della scheda in qualsiasi ambiente.

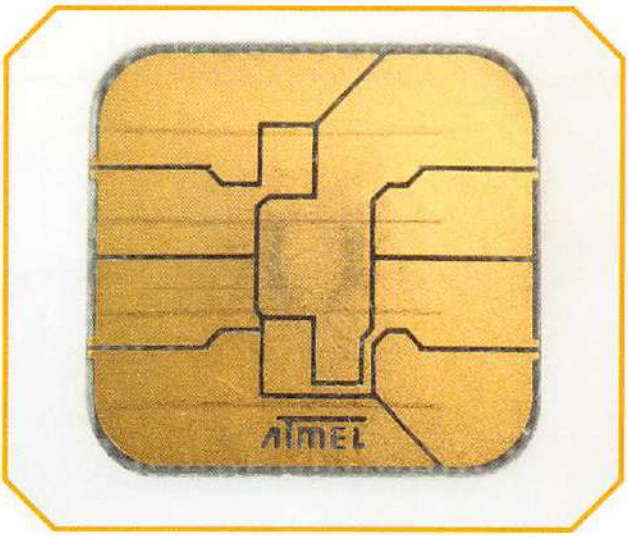
Funzionalità

Le schede Smart Card sono state sviluppate come sistema di immagazzinamento di informazione intelligente e interattivo. Il loro uso pertanto va dai semplici sistemi di moneta elettronica, fino ai sistemi di identificazione associati all'immagazzinamento di informazioni degli elementi da identificare. Il loro piccolo formato le rende ideali anche come sistemi di identificazione personale. Grazie alla capacità di poterne modificare il contenuto senza la necessità di uno scrittore particolarmente costoso e alla possibilità di eseguire molte registrazioni senza il rischio di perdere l'informazione, stanno sostituendo le tradizionali schede a banda magnetica. Inoltre le Smart Card con microprocessore permettono di avere un controllo molto più sicuro sull'identificazione, infatti dopo diversi accordi internazionali fra i costruttori esistono degli identificativi diversi per tutte le schede che circolano nel mondo.

Le misure di una scheda Smart Card sono definite da standard, ma si possono realizzare schede di qualsiasi dimensione. Se avete un cellulare vedrete che al suo interno esiste una scheda Smart Card di dimensioni molto ridotte, ma le prestazioni e gli utilizzi sono gli stessi di qualsiasi altra scheda.

Pensando a quest'ultimo esempio, quello dei cellulari, vi potete fare un'idea della diffusione mondiale e dell'espansione di questo tipo di schede sul mercato.

Schede Smart Card.



Contatti per alimentare e comunicare con il chip.

Schede Contlact less Card a radiofrequenza.

Tipi di schede

Le schede Smart Card si possono classificare in due grandi gruppi: le schede con microprocessore e le schede di memoria.

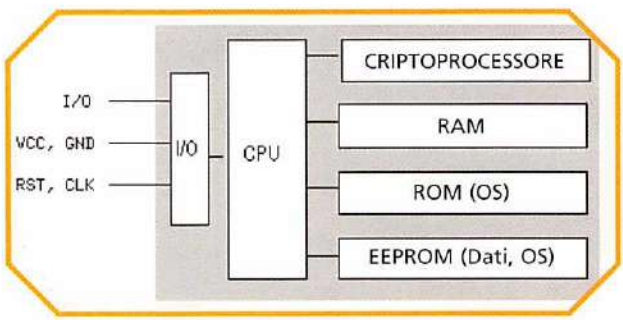
Schede Smart Card con microprocessore

Nello schema a blocchi della figura sottostante è riportatala la struttura generale di una scheda intelligente basata su un microprocessore. In essa possiamo vedere che l'accesso alle aree di memoria è unicamente possibile tramite l'unità di ingresso/uscita e di una CPU (tipicamente da 8 bit), cosa che chiaramente aumenta la sicurezza del dispositivo.

Esiste anche un sistema operativo integrato nella scheda, generalmente contenuto in una ROM, è inoltre possibile estendere le funzioni all'interno di una EEPROM, il cui compito principale è eseguire funzioni criptografiche. Il criptoprocessore supporta questi lavori offrendo operazioni RSA con chiavi da 512 a 1.024 bit. Un esempio di implementazione reale di questo schema è costituito dalla scheda intelligente CERES, dell'Istituto Spagnolo equivalente alla nostra Zecca di Stato. In essa oltre ai dispositivi già citati è contenuto anche un generatore di numeri casuali, insieme ai meccanismi di protezione interni della scheda.

Le Smart Card con microprocessore, trova-

Schede Smart Card per telefoni cellulari.



Schema generale di una scheda con microprocessore.

Scheda Smart Card del laboratorio.

no il loro principale utilizzo nel settore dei sistemi di conteggio (carte di credito, schede per telefonia, ecc.) e di identificazione di alta sicurezza.

L'evoluzione di questo tipo di schede è stata tale che esistono già sul mercato le schede "Contactless Card" (schede senza contatto), il cui funzionamento è basato su un'interfaccia via radio. Per questo tipo di Card è sufficiente essere vicini al lettore (circa 10 cm). Entrambi (lettore e scheda) hanno un'antenna e la comunicazione si genera tramite radiofrequenza.

Sono ideali per le applicazioni che prevedono alte densità di passaggio e che richiedono velocità di elaborazione.

Scheda di memoria

Sostituiscono la complessità del sistema di sicurezza con una maggior capacità di immagazzinare i dati. Attualmente se ne costruiscono con tagli da 64 Kb di memoria. La scheda che viene fornita per il laboratorio è una scheda di memoria con una capacità di contenimento di 1K. Possiede una memoria EEPROM con 24C16 al suo interno.

Applicazioni

La realizzazione di software associato a questo nuovo ambiente permette diverse applicazioni commerciali per le schede Smart Card:

– Controllo di accesso e di presenza: limitano e controllano l'accesso a aree ristrette, edifici, officine, computer...

– Pagamenti elettronici: offre una soluzione ideale per applicazioni relative a carte di credito, schede telefoniche, macchine distributrici, acquisti elettronici, ecc..

– Trasporti: mezzo di pagamento sicuro e facile da utilizzare per trasporti pubblici, biglietteria d'aerei, parchimetri, pedaggi autostradali, ecc..

– Identificazione, autenticazione e firma digitale: controllo di accesso a computer, terminali, reti, applicazioni software, data base, directory, file confidenziali, firme digitali, ecc..

– Sanità: memorizzazione di dati del paziente, come ad esempio la sua cartella clinica.

– Processi industriali: controllo di accesso in processi di produzione, misura del tempo, sicurezza industriale, ecc..

All'interno del laboratorio utilizzeremo la Smart Card per memorizzare programmi che verranno successivamente fatti girare sul microcontroller. Quando inseriremo la scheda sul lettore, il programma contenuto in essa si caricherà sul microcontroller se questo sarà stato precedentemente programmato con il programma residente Bootloader o Uploader.

La scheda Smart Card nel laboratorio

Per lavorare con la scheda sul laboratorio dobbiamo inserire quest'ultima nella posizione

Diverse applicazioni delle schede Smart Card.

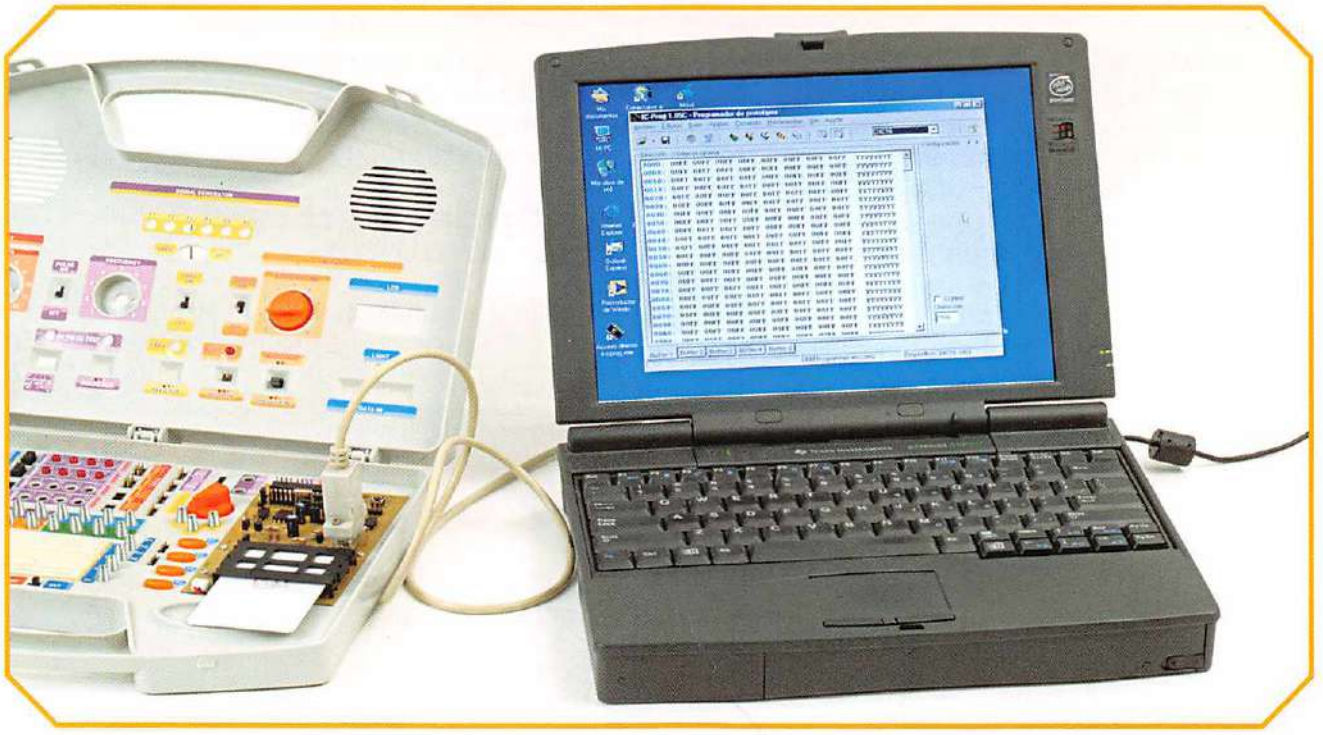
La scheda Smart Card sul nostro laboratorio.

Possiamo avere una scheda per ogni programma che vogliamo caricare.

corretta, con i contatti rivolti verso il basso fino a farle toccare il fondo completamente. Nel suo zoccolo la scheda potrà essere scritta e potremo anche caricare su di essa il programma contenuto sul PIC, sempre che su quest'ultimo sia presente il programma Bootloader. Questo programma permetterà di scaricare automaticamente nella memoria del PIC il codice scritto sulla Smart Card.

La scheda permette di eseguire fino a 1.000.000 di cicli di cancellazione e scrittura, quindi potremo registrare fino a un milione di programmi diversi. È possibile inoltre disporre di diverse Smart Card e caricarle con un programma diverso per ogni scheda, potremo quindi cambiare scheda in funzione del programma che desideriamo caricare sul PIC.

Per scrivere la scheda è sufficiente collegare il cavo di trasferimento tra il laboratorio e il PC e configurare IC-Prog in modo adeguato.



Scriveremo la scheda utilizzando IC-Prog.

La scheda Smart Card e il bootloader

Abbiamo visto la scheda Smart Card ma non sappiamo ancora come utilizzarla sul nostro laboratorio.
Tramite la scheda Smart Card caricheremo i nuovi programmi sul PIC, sempre che su di esso si trovi registrato il programma residente Bootloader e se il laboratorio è stato configurato correttamente.

Scrittura della Smart Card

Per scrivere la scheda Smart Card utilizzeremo il software IC-Prog, lo stesso che abbiamo usato per scrivere il PIC. Il processo di scrittura è praticamente lo stesso con alcune differenze che ora vi spiegheremo.

Configurazione hardware del laboratorio

La scheda di comunicazione DG07 ha alcuni connettori specifici per il lavoro con la Smart Card: JP4, JP5, JP6 e JP7.

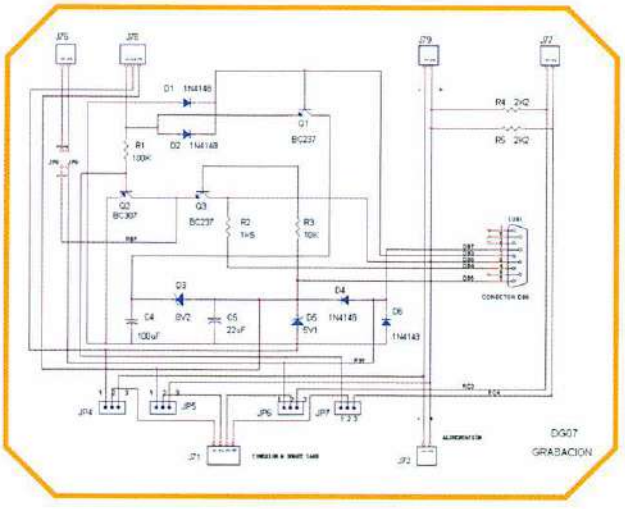
I primi due selezionano l'alimentazione con cui lavorerà la scheda, JP4 per il negativo e JP5 per il positivo. Quando questi due connettori hanno i ponticelli sulle posizioni 1 e 2, l'alimentazione che riceve la scheda è quella che arriva dalla porta seriale del computer (processo di trasferimento con PC), tuttavia se il ponticello unisce i terminali 2 e 3 di ogni connettore, l'alimentazione ricevuta è quella del laboratorio (quando vogliamo caricare un programma sul PIC tramite la scheda).

Gli altri due connettori JP6 e JP7, agiscono sulle linee di comunicazione con la scheda. Quindi se i ponticelli sono sulle posizioni 1 e 2 potremo trasferire alla scheda i dati che arrivano dalla porta seriale del PC, cioè registriamo la scheda. Se i ponticelli sono sulle posizioni 2 e 3 la scheda comunicherà con il PIC tramite i terminali RC3 e RC4, trasferendo i dati in essa contenuti.

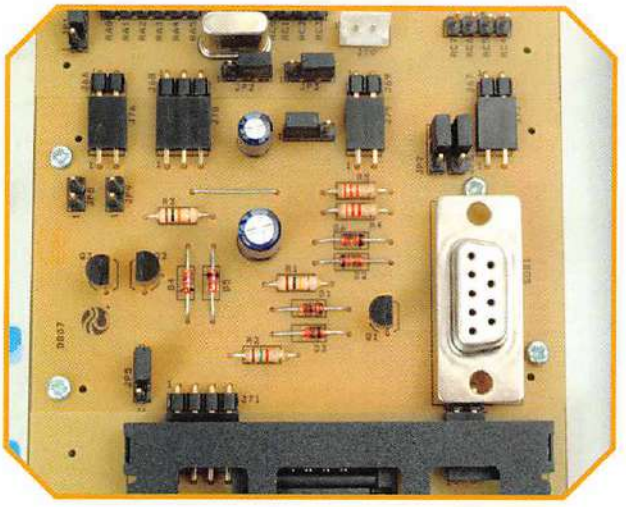
Per precauzione dobbiamo togliere, nel caso fossero inseriti, i ponticelli dei connettori JP8 e JP9, dato che questi si utilizzano solamente per scrivere il PIC. La configurazione dei ponticelli della scheda DG06 sarà quella del lavoro normale.

Configurazione di IC-Prog e scrittura

Con il laboratorio configurato correttamente per la scrittura della Smart Card collegheremo



Circuito elettrico della scheda DG07.



Configuriamo la scheda DG07 per lavorare con la Smart Card.

Connettori	Ponticelli per la scrittura della scheda	Ponticelli per scrivere il PIC con la scheda
JP4	Entre 1 y 2	Entre 2 y 3
JP5	Entre 1 y 2	Entre 2 y 3
JP6	Entre 1 y 2	Entre 2 y 3
JP7	Entre 1 y 2	Entre 2 y 3
I ponticelli dei connettori JP8 e JP9 devono essere inseriti.		

Configurazione dei ponticelli della scheda DG07.

il cavo di comunicazione fra questi e il PC, e inseriremo la scheda sul suo zoccolo con i connettori verso il basso. Apriremo IC-Prog e selezioneremo il tipo di dispositivo che vogliamo programmare. La memoria EEPROM interna che possiede la scheda e del modello 24C16, quindi sul menù a tendina nella parte superiore a destra del monitor selezioneremo questo dispositivo. Fatto questo selezioneremo il file che vogliamo trasferire sulla scheda. Come per il PIC i file che devono essere scritti sulla scheda, sono file in codice macchina, cioè con estensione ".hex". Dopo aver aperto il file lo possiamo scrivere direttamente sulla scheda, dato che questa non richiede una cancellazione preventiva, anche se è consigliabile eseguire una verifica successiva all'operazione per vedere se è stata eseguita correttamente.

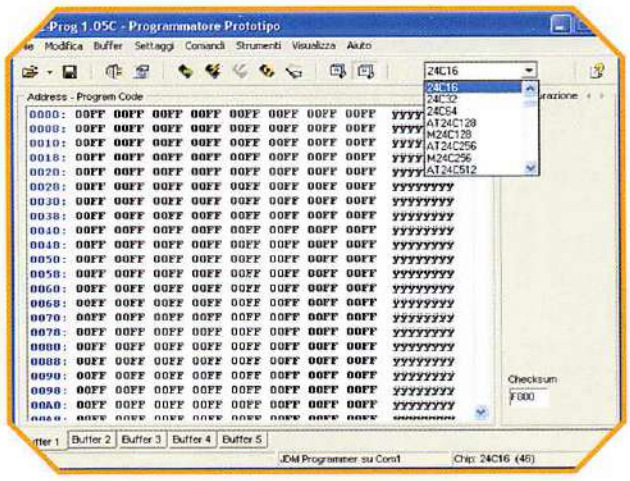
A titolo di esempio carichiamo il programma "forno 75-125.hex" e selezioniamo l'opzione Programma Tutto. Come succedeva quando scrivevamo il PIC, una finestra ci indica lo stato del processo. La scheda si programma correttamente quindi verificheremo il suo contenuto mediante l'opzione Leggi Tutto.

Caricare un programma della Smart Card sul PIC

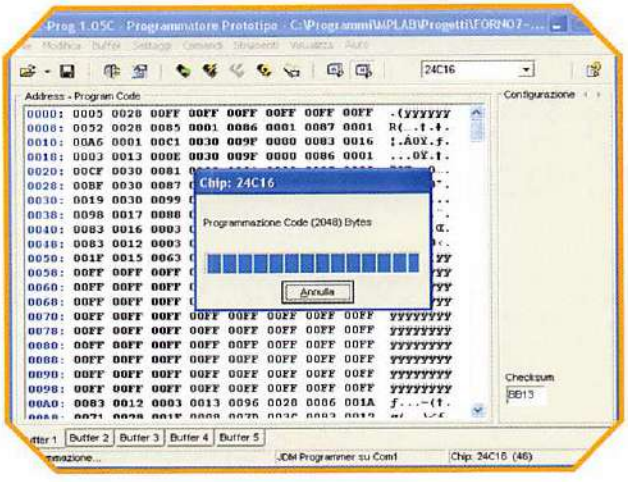
Per caricare un programma della Smart Card sul PIC è fondamentale che su quest'ultimo si trovi il programma residente Bootloader. Questo programma stabilirà la comunicazione con la scheda, copierà il contenuto di questa sulla memoria del PIC e lo eseguirà. Quando vogliamo caricare sul PIC un nuovo codice, utilizzando la Smart Card, dobbiamo verificare che il laboratorio sia correttamente configurato: sulla scheda DG06 i ponticelli devono essere inseriti in modo lavoro e non in modo scrittura, e sulla scheda DG07 i ponticelli avranno la configurazione che possiamo vedere nella tabella. In questo modo, quando inseriamo la scheda sullo zoccolo, il PIC rimarrà automaticamente programmato con il nostro codice. Questa operazione richiede un certo tempo, quindi prima di togliere la scheda aspetteremo almeno cinque secondi per assicurarci che il processo sia stato completato.

Modifiche dei programmi che girano con Bootloader

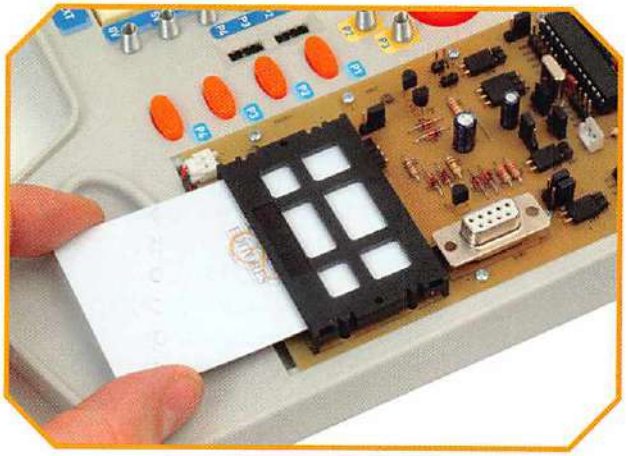
Se abbiamo un programma che sta girando sul PIC e ne vogliamo caricare uno nuovo compatibile con quello residente, quest'ultimo dovrà avere la memoria organizzata in modo che quando funzionano entrambi e



Selezione del dispositivo da programmare.



Scrittura del programma sulla scheda.



Processo di caricamento di un nuovo codice sul PIC.

contemporaneamente non interferiscano fra di loro. Mediante le direttive ORG si organizzano le porzioni di memoria che occupa un programma. I programmi con cui abbiamo lavorato fino a ora occupano l'indirizzo 0 con un' istruzione di salto all'indirizzo 5, a partire dal quale trova posto il programma. Questi indirizzi ora sono occupati dal Bootloader, i nuovi programmi che caricheremo, quindi, non potranno occupare la memoria che è già

Senza il Bootloader		***Con il Bootloader***
ORG 0	°ORG	0xB0
goto Inizio		
ORG	5	Inizio: ...
Inizio: ...		

Riorganizzazione della memoria per i programmi che devono girare con Bootloader.

utilizzata da questo programma residente. I programmi che scriveremo sulla Smart Card avranno un codice che partirà dall'indirizzo 0xB0.

Nel primo CD fornito con l'opera, gli esercizi in esso contenuti, avevano la configurazione che vi abbiamo spiegato. Se vogliamo lavorare in questo modo con qualsiasi altro programma lo dovremo adattare.

Il Bootloader

I microcontroller PIC hanno nel loro repertorio istruzioni capaci di scrivere la memoria di

ese3 - Blocco note

File Modifica Formato Visualizza ?

```
;
;Programma somma
;
;Al valore che si inserisce tramite gli interruttori RC0-RC2 dalla porta C, si somma una
;costante (in questo esempio il valore 3). Il risultato si visualizza sui LED RB0-RB7
;collegati alla porta B
;
                List    p=16F870        ;Processore
                include "P16F870.INC"   ;Definizione dei registri interni

                ORG     0xB0

Inizio          clrf    PORTB           ;Azzera eventuali valori residui
                bsf     STATUS,RP0      ;Seleziona il banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00000111'
                movwf   TRISC           ;3 bit della porta C si configurano come ingresso
                bcf     STATUS,RP0      ;Seleziona il banco 0

Loop:           clrwdt                  ;Azzera il WDT
                movf    PORTC,w         ;Carica lo stato degli ingressi RC0-RC2
                addlw   .3              ;Si somma 3
                movwf   PORTB           ;Visualizza il risultato su RB0-RB7
                goto    Loop

                end                     ;Fine del programma
```

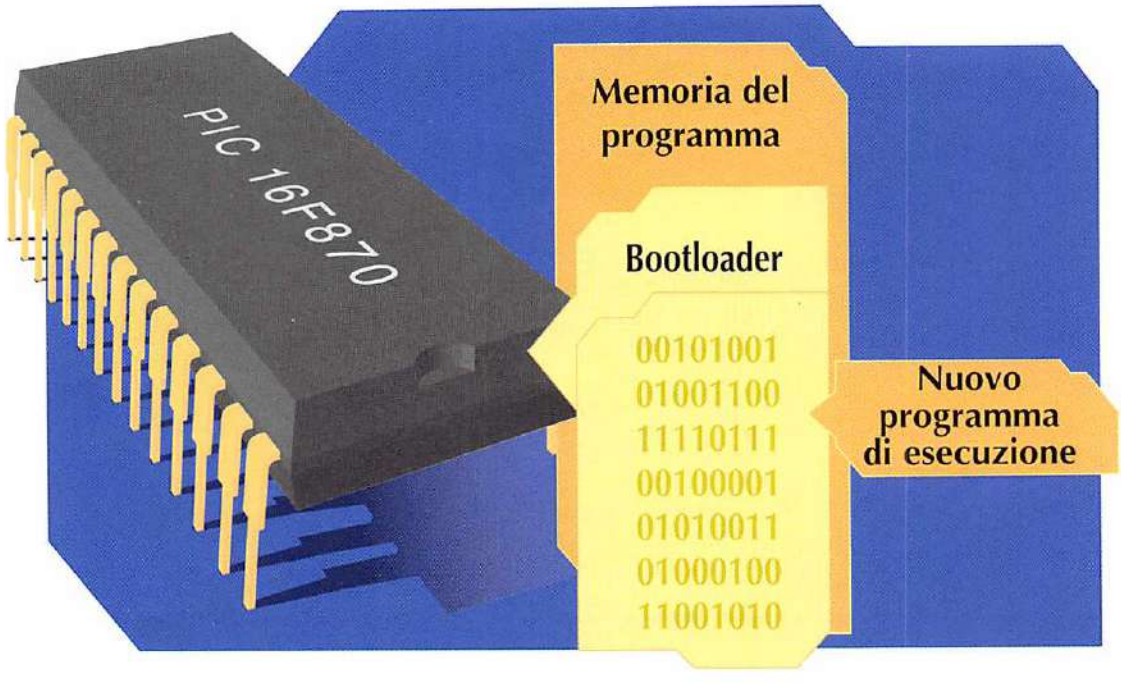
Diverse applicazioni della scheda Smart Card.

programma all'interno del programma stesso. Questo tipo di istruzioni sono utili nella programmazione per fare in modo che il nostro codice si possa autoriconfigurare, e possa quindi accettare programmi di esecuzione senza dover togliere il microcontroller dal circuito in cui si trova inserito.

Siamo pronti per lavorare con la Smart Card.

Il Bootloader è un programma che permette di caricare un nuovo codice sul PIC senza la necessità di smontare il circuito o togliere il PIC per scriverlo utilizzando un hardware diverso. Vediamo ora il funzionamento di questo programma: dopo il reset del micro, la prima cosa che viene eseguita è il Bootloader, il quale inizializza la porta seriale del micro e se riceve l'ordine adeguato, utilizza le istruzioni prima menzionate per scrivere all'interno della memoria di programma i dati che riceve tramite la porta, a partire da un indirizzo predeterminato della memoria di programma.

Dopo aver scritto sul micro il nostro nuovo codice, ogni volta che si resetterà, entrerà nuovamente nel ciclo di boot, ma nel caso in cui inizializzi la porta seriale e non ottenga risposta, si dirigerà all'indirizzo di memoria a partire dal quale inizia il nostro programma.



Il programma viene scritto nelle posizioni libere della memoria.

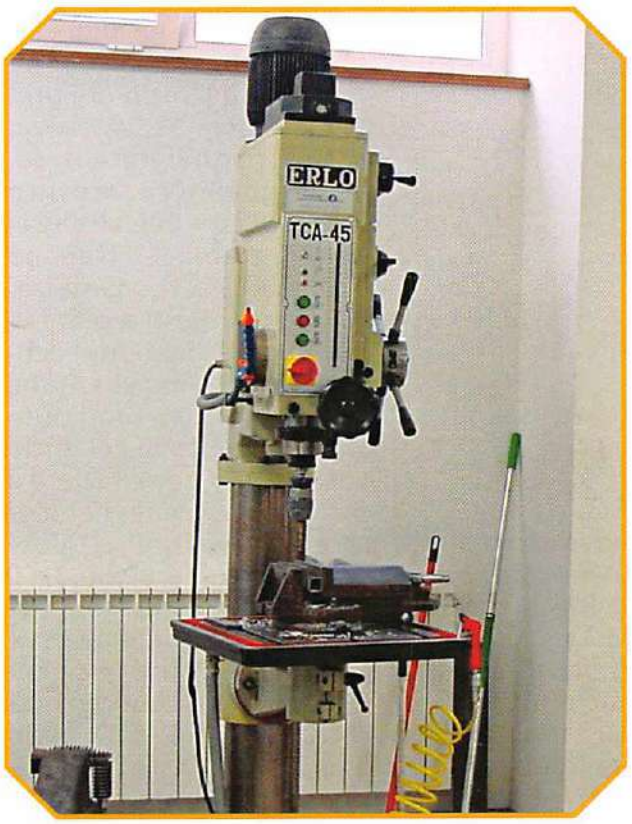
Controllo dei processi industriali: il trapano

L'impiego di microcontroller per il controllo di processi industriali è molto diffuso. La semplicità, la versatilità e il costo ridotto sono le caratteristiche che permettono alle applicazioni basate su microcontroller di essere la soluzione migliore per il controllo di molti processi industriali.

Macchina di foratura

L'applicazione che svilupperemo si prefigge di automatizzare una macchina di foratura. Questi tipi di processi normalmente richiedono la presenza di un operaio per l'esecuzione, sono monotoni e a volte anche pericolosi. Mediante l'integrazione di un semplice hardware basato su un microcontroller si semplifica il processo che viene eseguito automaticamente, ottimizzando tempi, qualità e costi.

Il processo che presenteremo di seguito è un processo reale di controllo di una macchina per foratura. Per forare un pezzo l'operaio deve avvicinare il trapano allo stesso, attivarlo, forare con la precisione necessaria e toglierlo, considerando così il pezzo terminato.



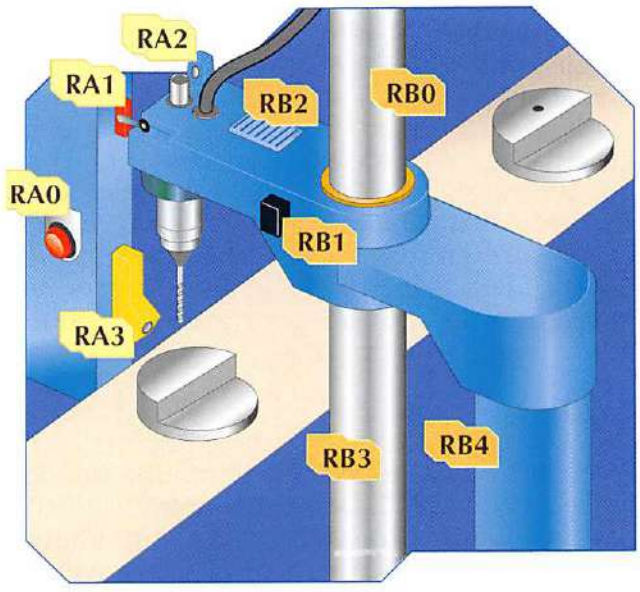
Processo industriale da automatizzare.

Sviluppo

Per automatizzare il processo posizioneremo due sensori, uno per rilevare l'inizio della foratura e l'altro per avvisare quando è stato realizzato il foro del pezzo. Inizieremo le operazioni mediante un pulsante e avviseremo acusticamente al termine del processo. Tutto questo si può pianificare nel seguente modo:

Azionando il pulsante "I" (RA0) la testa di foratura esegue una discesa rapida di avvicinamento, azionando il motore di discesa rapida "DR"(RB3).

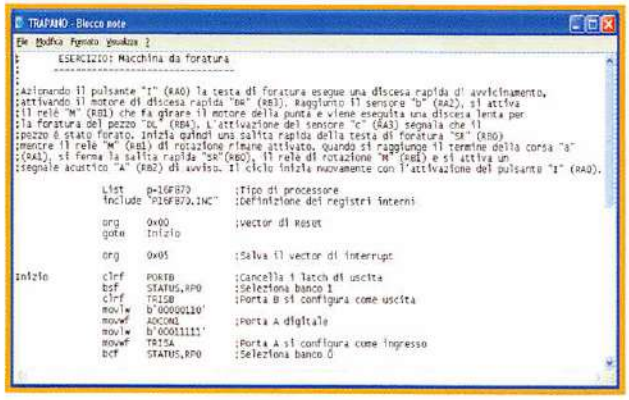
Raggiunto il sensore "b" (RA2), si attiva il relè "M" (RB1) che fa girare il motore della punta e si esegue una discesa lenta per la foratura del pezzo "DL" (RB4). Quando si attiva il sensore "c" (RA3) si considera che il pezzo



Controllo automatico di una macchina per foratura.

Ingressi	RA0	*Pulsante che indica l'inizio del processo, pezzo nuovo.*
	RA1	*Finecorsa che indica che il trapano si trova nella posizione di origine.*
	RA2	*Sensore per indicare l'inizio della rotazione del trapano.*
	RA3	*Sensore di pezzo forato.*
Uscite	RB0	*Attiva la salita rapida del trapano.*
	RB1	*Attiva il relè che attiva il motore di rotazione del trapano.*
	RB2	*Segnale acustico di fine del processo.*
	RB3	*Attiva il motore di discesa rapida del trapano.*
	RB4	*Attiva la discesa lenta per forare il pezzo.*

Segnali di ingresso e uscita.



Configurazione delle porte di I/O.

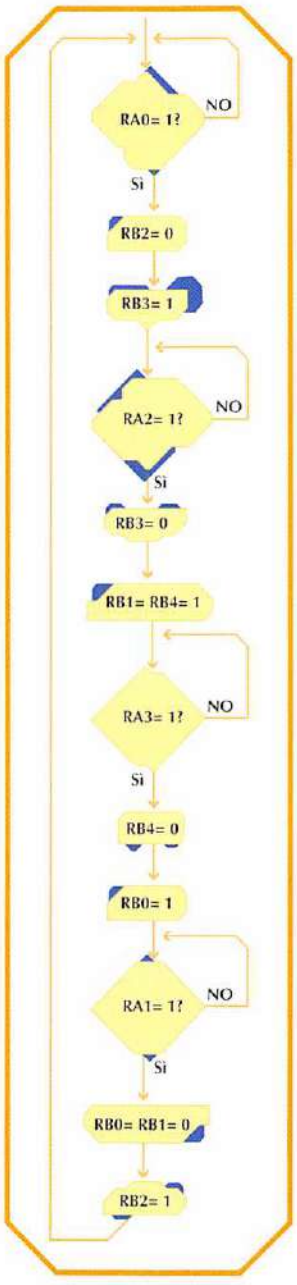
sia forato. Si inizia una salita rapida della punta "SR" (RB0) e durante questo tempo il relè "M" (RB1) di rotazione continua a essere attivato. Quando viene raggiunto il finecorsa "a" (RA1), si ferma la salita rapida "SR" (RB0), il relè di rotazione "M" (RB1) e si attiva un segnale acustico "A" (RB2). Il ciclo comincia con una nuova attivazione di "I" (RA0).

Abbiamo definito i dispositivi necessari per automatizzare il processo e, allo stesso tempo, abbiamo stabilito gli ingressi e le uscite con cui lavorerà il microprocessore. Non sono necessari altri dispositivi, dato che per risolvere questa applicazione è possibile utilizzare un microcontroller più semplice, cosa che ridurrebbe maggiormente i costi.

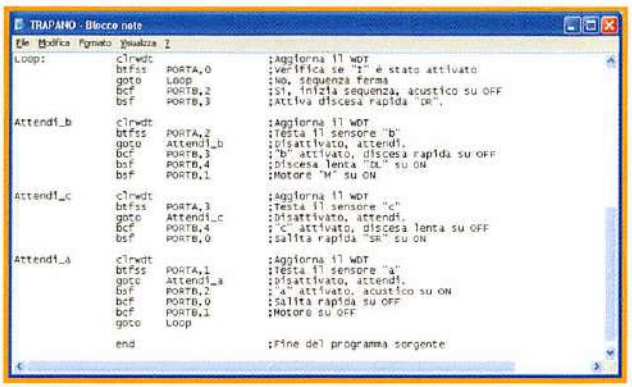
Codice

Predisporremo ora una soluzione utilizzando il nostro PIC16F870. Dopo aver definito tutti i passaggi e averli inseriti in un organigramma, sarà sufficiente far riferimento a esso e sviluppare il codice mantenendo l'ordine indicato.

Intesteremo il programma e definiremo i dispositivi con cui lavorare, in questo caso le porte di ingresso e di uscita. Utilizzeremo la porta B per le uscite e la porta A per gli ingressi, specificando che questi ultimi saranno digitali. Il programma principale resterà fermo nell'attesa che arrivi il segnale di ingresso RA0, segnale che indica l'inizio del processo. Se si attiva, spegneremo il segnalatore acustico (RB2 a 0) e abbasseremo il trapano (RB3 a 1). Manterremo questi segnali fino a quando il sensore ci indicherà che abbiamo raggiunto il punto in cui il trapano deve iniziare a funzionare, nel momento in cui il sensore si attiva (RA2 a 1), abbasseremo il trapano più lentamente (RB3 a 0 e RB4 a 1) e questo inizierà a funzionare (RB1 a 1). Il sensore "b" ci indicherà che il pezzo è stato forato correttamente (RA3 a 1), e a questo punto ritireremo il trapano facendolo risalire rapidamente (RB4 a 0 e RB0 a 1). Quando raggiunge il finecorsa (RA1 a 1) scollegheremo i segnali di movimento del trapano e attiveremo il segnale acustico generando l'avviso di pezzo terminato.



Organigramma dell'applicazione.



Codice che risolve l'applicazione.

Compilazione

Apriremo MPLAB per procedere alla compilazione e simulazione del programma, creando un nuovo progetto a cui aggiungeremo il codice realizzato; selezioneremo poi l'opzione Build All.

Il codice si deve compilare senza errori, come possiamo verificare nella figura.

Simulazione

Quando realizziamo un progetto per automatizzare un processo industriale, prima di implementarlo fisicamente, dobbiamo verificare che risponda correttamente. Non è consigliabile fermare la produzione di un'azienda per montare un sistema che poi non funziona. Per evitare difetti di funzionamento, simuleremo prima il programma.

In MPLAB apriremo le finestre dei registri delle funzioni speciali, e una finestra per poter osservare unicamente i registri di ingressi e uscite (PORTA e PORTB). Vedremo questi registri in binario, perché in questo modo potremo verificare la risposta di ogni pin in modo indipendente. Ricordate che per cambiare la visualizzazione bisogna accedere alle proprietà e selezionare Binary, dopo aver selezionato il registro che si vuole vedere nella finestra.

Per simulare il pin di ingresso apriremo il simulatore di stimoli asincroni e assegneremo i cinque ingressi ai pulsanti corrispondenti.

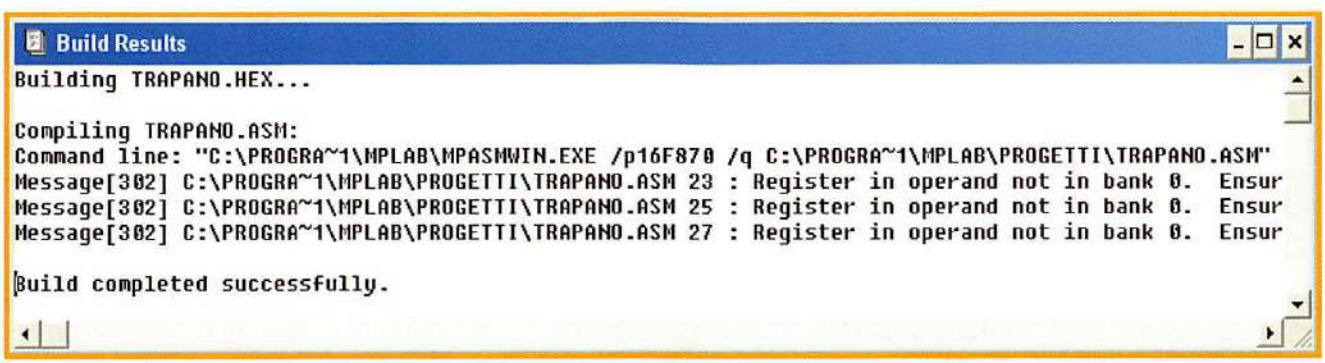
Inizieremo la simulazione passo a passo e vedremo come il programma ripete un ciclo in attesa che l'ingresso si attivi. Imposteremo tutti gli stimoli dei segnali di ingresso a livello alto (High) e simuleremo il processo. Inizieremo cliccando con il mouse sul pulsante assegnato al pin RA0, per simulare l'attivazione dell'ingresso; se proseguiamo nell'esecuzione vedremo che il programma esce dal ciclo ed esegue i primi due ordini di uscita. Ripeteremo questa operazione nel resto del programma, ovvero attiveremo gli ingressi che simulano i sensori e i finecorsa per uscire dai cicli di attesa e osserveremo nella finestra dei registri di uscita in binario che le uscite si attivano correttamente.

Se si desidera fare un nuovo pezzo dovremo impostare i pin del simulatore di stimoli a livello basso (Low) e ripetere nuovamente tutti i passaggi.

Conclusioni

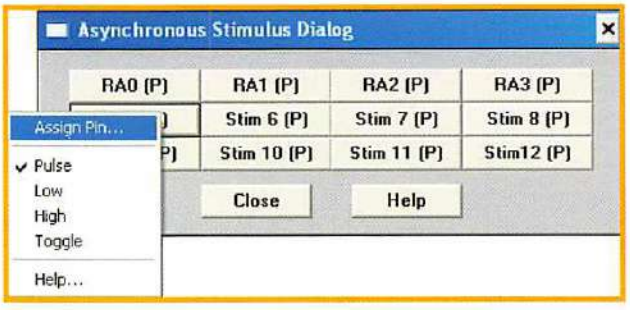
Il programma ha risposto correttamente alla simulazione, quindi lo potremo scrivere sul microcontroller e iniziare a realizzare l'implementazione fisica del sistema.

È indispensabile eseguire la simulazione dei



Risultato della compilazione.

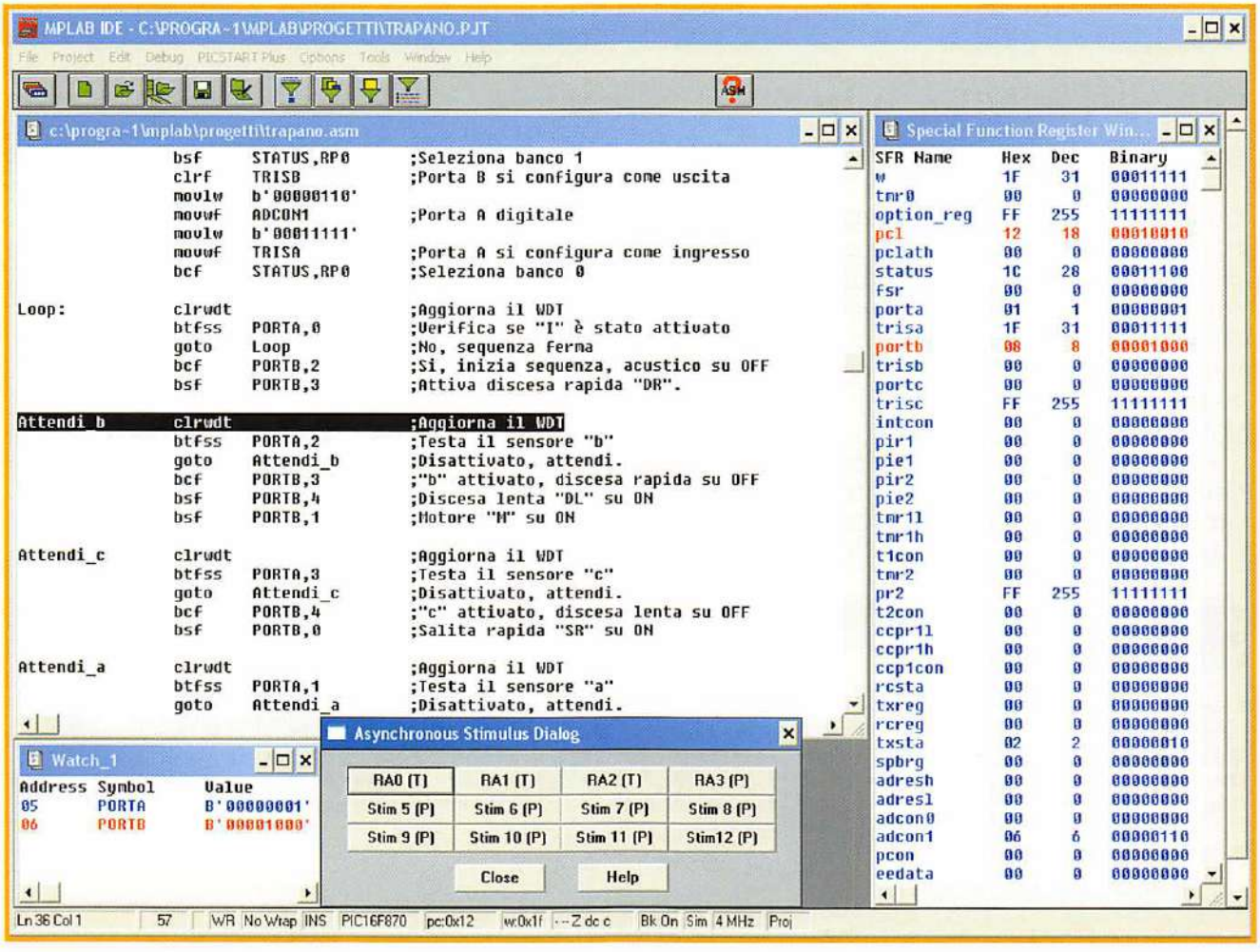
Finestra per vedere in binario le porte di I/O.



Forziamo gli ingressi con il simulatore di stimoli.

programmi che sono destinati a processi industriali, dato che qualsiasi errore può far fermare la produzione, il ché comporterebbe una perdita economica considerevole.

Molte delle applicazioni per le quali si utilizza un microcontroller semplificano molto i processi industriali e non richiedono una programmazione complessa. Grazie a questo esempio reale avete potuto verificare come un microcontroller si possa convertire in un elemento fondamentale per un processo produttivo.



Videata di MPLAB durante la simulazione.

Controllo dei processi industriali (II): il tornio

Vedremo ora un altro esempio reale di automatizzazione di un processo industriale. In questo caso controlleremo la tornitura di pezzi e grazie all'utilizzo di un microcontroller, ottimizzeremo il processo di alimentazione e controllo della macchina di tornitura (tornio).

Il tornio

Il tornio è una macchina molto diffusa nelle officine meccaniche. I sistemi di alimentazione dei pezzi o lo stesso processo di tornitura possono variare in funzione della complessità della macchina. Possiamo anche trovare sul mercato torni programmabili, ma noi lavoreremo su uno dei più semplici e diffusi, offrendo una facile ed economica soluzione per la sua automatizzazione.

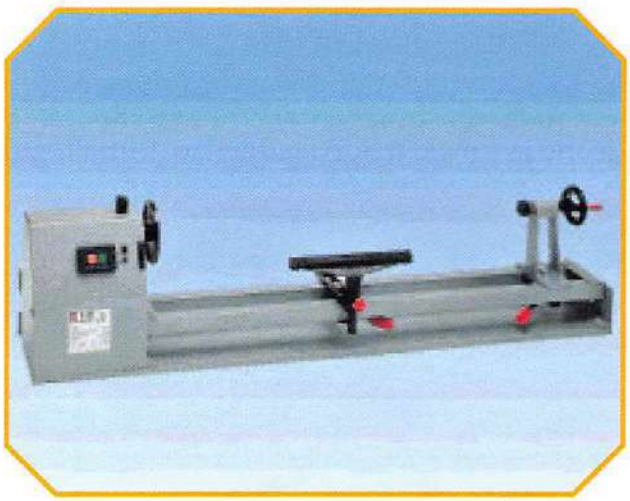
Analisi del progetto

Quando si affronta un progetto di questo tipo la prima cosa da fare è analizzare il funzionamento per adottare una soluzione ottimale. In questa analisi definiremo i dispositivi esterni da utilizzare come sensori, motori, ecc., dopodiché potremo dedurre gli ingressi e le uscite con cui lavoreremo e, infine, definiremo il resto dei dispositivi nel caso fossero necessari.

La soluzione per controllare questo processo è la seguente:

La sequenza inizia premendo "I" (RA0) che attiva il cilindro "V" (RB0). Questi, nel suo avanzamento, incontra il pezzo da tornire e aziona il sensore "b" (RA2), attivando il relè del motore "M" (RB1). Inizia la tornitura. Durante l'avanzamento si raggiunge il sensore "c" (RA3), che disattiva il cilindro "V" (RB0) iniziando la retrocessione dello stesso. Quando si passa nuovamente per "b" (RA2) si disconnette "M" (RB1). Quando si raggiunge il finecorsa "a" (RA1) si attiva un segnale acustico "A" (RB2) per avvisare l'operaio di togliere il pezzo, collocarne uno nuovo e iniziare un nuovo ciclo premendo "I" (RA0).

Diversi tipi di torni presenti sul mercato.



Tornio tipico sul quale possiamo eseguire un controllo automatico.

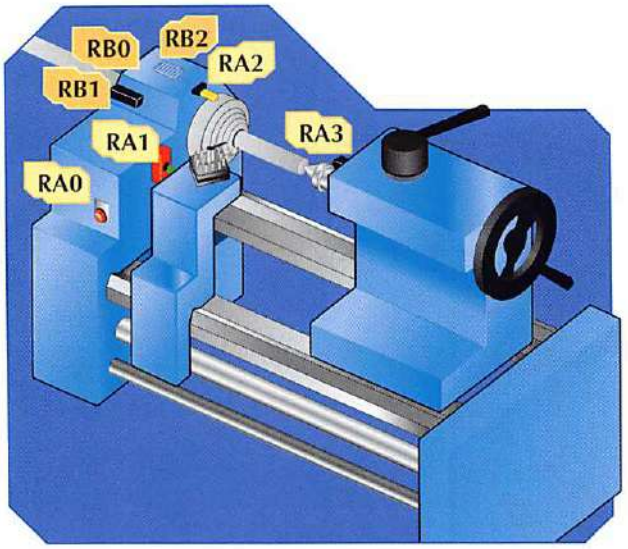


Diagramma del processo.

Organigramma

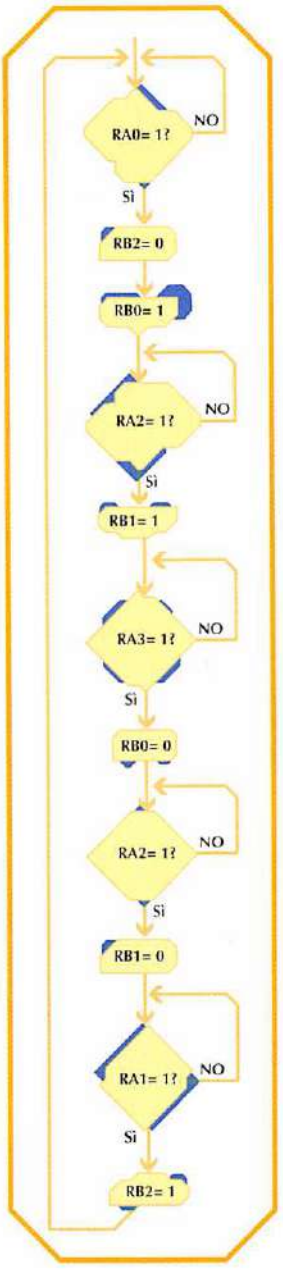
Per questo tipo di programmi, che nascono per controllare un processo industriale, sono fondamentali gli organigrammi. Il processo di automatizzazione avrà un ordine ben preciso che verrà inserito nell'organigramma allegato al funzionamento generale del programma. Se generiamo un organigramma che si adatti correttamente all'applicazione, la fase successiva di creazione del codice risulterà molto semplice. Nella figura possiamo vedere l'organigramma corrispondente a questa applicazione.

Codice

Iniziamo, come sempre, intestando il programma con i commenti pertinenti che chiariscono la funzionalità dello stesso. Definiremo

Ingressi	RA0	Pulsante che indica l'inizio del processo, pezzo nuovo.
	RA1	Finecorsa per indicare che il cilindro si trova nella posizione di origine.
	RA2	Sensore che indica l'inizio della rotazione del tornio.
	RA3	Sensore che indica la retrocessione del cilindro.
Uscite	RB0	Attiva il cilindro di avanzamento del pezzo.
	RB1	Attiva il relè di attivazione del motore di rotazione del tornio.
	RB2	Segnale acustico di fine della tornitura.

Tabella degli ingressi e delle uscite.

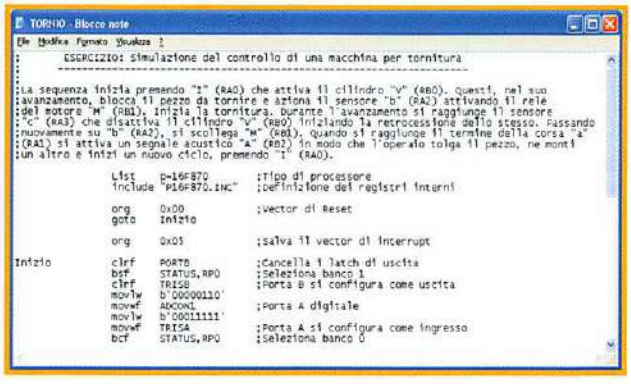


Organigramma dell'applicazione.

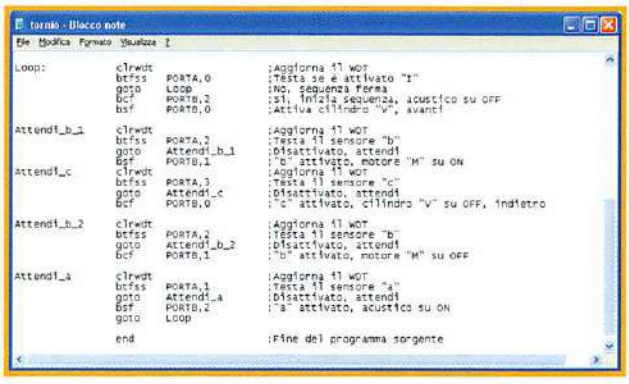
il PIC da utilizzare e la libreria dove sono definiti i registri e, mediante le direttive ORG, organizzeremo la memoria di programma.

Il passo successivo consiste nel definire i dispositivi con cui vogliamo lavorare. Come per l'esempio del trapano, abbiamo bisogno unicamente di lavorare con porte di ingresso e di uscita, quindi le dobbiamo configurare. La porta A sarà una porta di ingressi digitali e la B di uscite.

Con l'aiuto dell'organigramma inizieremo a definire il codice per controllare il processo. Dobbiamo attendere che si attivi il pulsante di inizio del processo per un nuovo pezzo. Nel momento in cui si attiva (RA0=1) spegneremo il segnale acustico di pezzo terminato (RB2=0) e daremo il segnale di avanzamento del cilindro (RB0=1). Quando il pezzo avrà raggiunto la posizione del sensore "b" (RA2=1) attiveremo il motore del tornio (RB1=1) e il pezzo verrà tornito fino a raggiungere la posizione del sensore "c" (RA3=1). A questo punto forniremo uno 0 sul cilindro che sostiene il pezzo (RB0=0), il quale inizierà la sua retrocessione. Quando il pezzo raggiungerà nuovamente il sensore "b" spe-



Prima parte del codice.



Codice che risolve l'applicazione.

gneremo il motore del tornio (RB1=0). Quando il pezzo tornerà sulla posizione di partenza, rilevata mediante un nuovo sensore "a" (RA1=1), il cilindro avrà raggiunto il suo arresto meccanico quindi non dovremo agire su di esso (manterremo l'ordine di retrocessione) ma dovremo avvisare l'operaio che il pezzo è pronto e che il processo sta attendendo un nuovo pezzo, cosa che faremo attivando il segnale di uscita RB2 (RB2=1).

Compilazione e simulazione

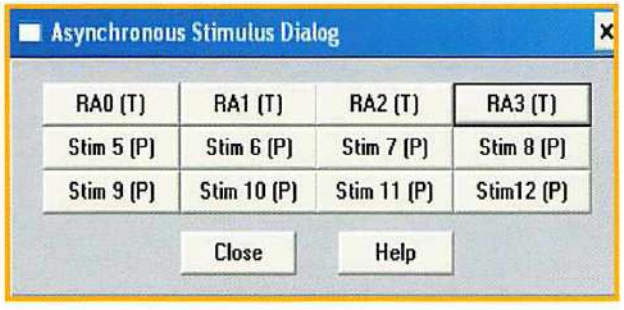
Dopo aver realizzato il codice dobbiamo verificare che non contenga errori e che risponda alle richieste iniziali. Apriamo MPLAB, creiamo un progetto cui allegheremo il nostro codice e selezioniamo l'opzione Build All. Il codice compila senza errori, come possiamo verificare nell'immagine in basso dove sono riportati i risultati di questa operazione.

Per simulare il programma apriamo le finestre abituali, quella dei Registri delle Funzioni Speciali e quella che contiene unicamente i registri più interessanti da visualizzare con la loro presentazione in binario. Sappiamo per esperienze precedenti che se cominciamo a simulare, il programma si fermerà nell'attesa che si attivi il segnale di ingresso e non potremo vedere la sua esecuzione completa. Mediante il simulatore di stimoli asincroni forzeremo gli ingressi per poter uscire dai cicli di attesa e verificare il programma in tutta la sua estensione. Assegneremo i quattro ingressi ai pulsanti corrispondenti e questa volta, anziché programmarli su High per fornire un livello alto all'ingresso attivato, li programmeremo in modo Toggle, in modo che a ogni pressione del pulsante l'ingresso corrispondente cambierà stato. Potremo quindi cambiare da 0 a 1 e viceversa senza dover riprogrammare il pulsante. Fatto questo, inizieremo la simulazione passo a passo e verificheremo l'evoluzione del programma. Il programma risponde in modo soddisfacente alle aspettative.

Montaggio e conclusioni

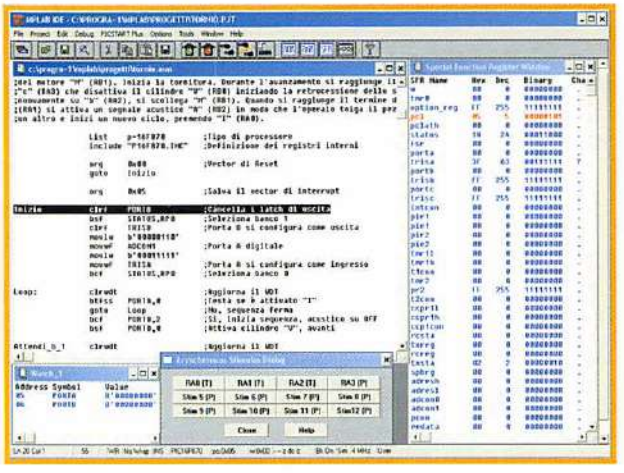
Come avete potuto vedere questo programma è molto simile a quello presentato in pre-

Risultato della compilazione.



Programmiamo l'opzione Toggle sui pulsanti del simulatore di stimoli.

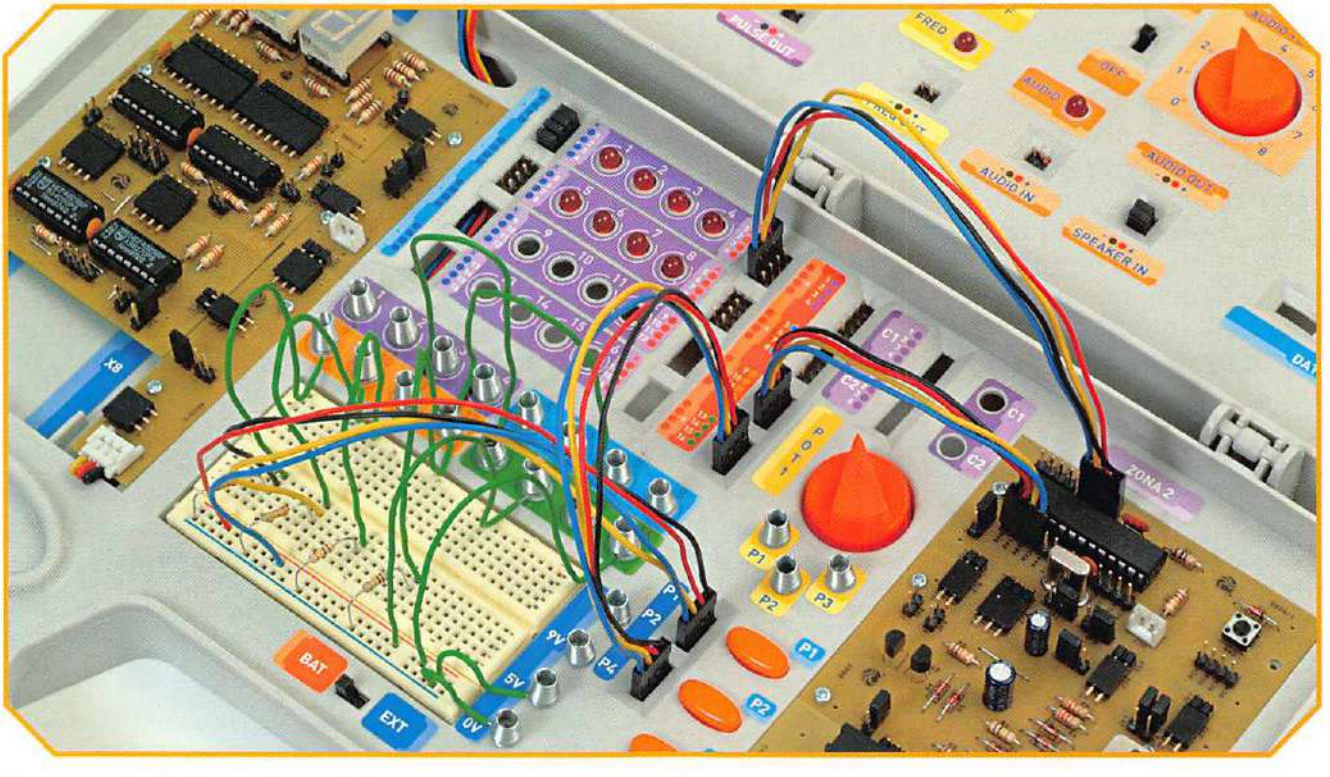
cedenza per automatizzare un processo di foratura. Il montaggio del circuito elettrico sul laboratorio è lo stesso di quello visto per la foratura, con l'unica eccezione costituita dal fatto che il processo di automatizzazione del tornio richiede due uscite in meno. È consigliabile caricare il programma sul PIC e provare l'ultima fase del progetto. La coincidenza col processo di foratura non è casuale, dato che per la maggior parte dei processi industriali l'idea è la stessa. Automatizzeremo qualsiasi processo utilizzando sensori, dispositivi attuatori (motori, relè, ecc.) e un microcontroller il cui



Aspetto generale di MPLAB durante la simulazione.

programma può arrivare a essere molto semplice per il suo programmatore.

Avete potuto verificare anche che il nostro PIC è sovradimensionato per questo tipo di progetti, dato che non utilizza altre risorse che le porte di I/O. Normalmente si lavora con temporizzatori e interrupt, però potremo sempre installare un PIC molto più semplice riducendo ulteriormente i costi del progetto.



Il montaggio della macchina di foratura è valido anche per il progetto del tornio.

Esercizio: generatore di onda, il programma

Le applicazioni che si possono sviluppare basandosi sull'utilizzo di un microcontroller, sono molte e diverse tra loro.
In questo caso progettiamo lo sviluppo di un generatore di onda quadra la cui frequenza possa essere determinata dall'utente.

Enunciato

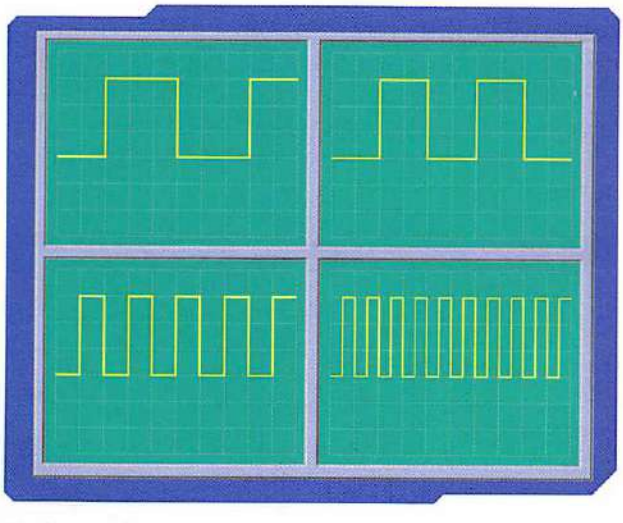
L'obiettivo è quello di progettare un generatore di onda quadra la cui frequenza possa essere determinata dall'utente. Immaginate quali applicazioni possa avere questo generatore: tergicristallo a frequenza variabile di un'automobile, segnale acustico per passaggio pedonale, ecc.

Per determinare la frequenza alla quale verrà generata l'onda utilizzeremo tre ingressi del micro RA2:RA0. In questo modo potremo avere le otto combinazioni della tabella nella figura.

L'onda quadra si otterrà sul terminale di uscita RB0.

Organigramma

La prima cosa da fare prima di progettare un programma che risponda ai requisiti dell'enunciato è plasmarne il funzionamento previsto su un organigramma. Questo ci servirà per sviluppare il codice in modo ordinato e semplice. Nella figura della pagina successiva possiamo vedere l'organigramma generale dell'applicazione, composto da un organigramma per il programma principale e da un altro per la subroutine di servizio all'interrupt. Utilizzando il temporizzatore TMR0 possiamo eseguire una serie di ordini dopo che è trascorso un determinato periodo di tempo o, in altre parole, a una determinata frequenza. Quando il temporizzatore va in overflow si genera un interrupt, ed è nella subroutine dedicata a esso che eseguiremo le azioni pertinenti.



Onde quadre di diverse frequenze.

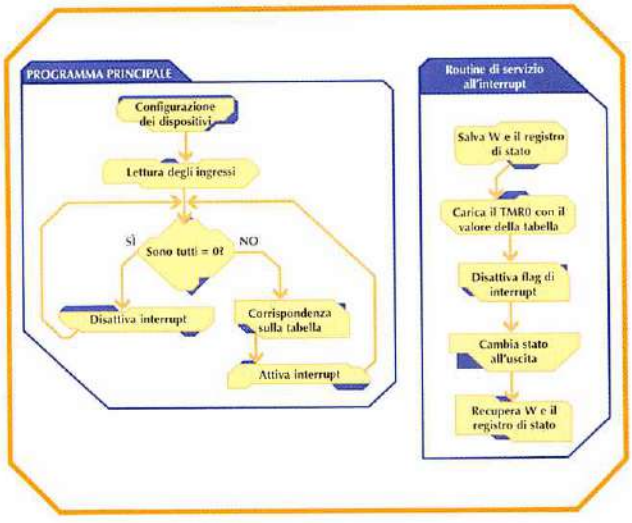
Codice

Intesteremo il codice inserendo sotto forma di commenti ciò che vogliamo che il programma esegua. Continueremo, come sempre, definendo il dispositivo, la libreria in cui si definiscono i registri interni e organizzando la memoria di programma mediante le direttive ORG. Ricordate che in questo programma lavoreremo con gli interrupt, quindi sarà necessario definire il vector di interrupt.

L'altra cosa da fare sarà quella di creare una tabella che contenga i diversi valori da caricare per ottenere le frequenze desiderate. Ogni volta che entreremo in questa subroutine ci sposteremo sulla tabella in funzione dei valo-

RA2	RA1	RA0	Frequenza	Periodo	Semiperiodo
0	0	0	0 kHz	-- --	-- --
0	0	1	1 kHz	1.000 µs	500 µs
0	1	0	2 kHz	500 µs	250 µs
0	1	1	3 kHz	333 µs	166 µs
1	0	0	4 kHz	250 µs	125 µs
1	0	1	5 kHz	200 µs	100 µs
1	1	0	6 kHz	166 µs	83 µs
1	1	1	7 kHz	143 µs	71 µs

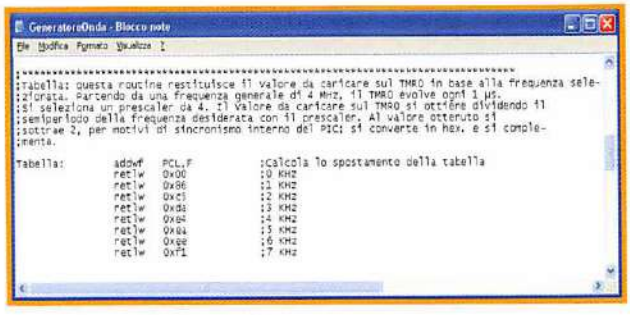
Frequenze in funzione della combinazione degli ingressi.



Organigramma dell'applicazione.

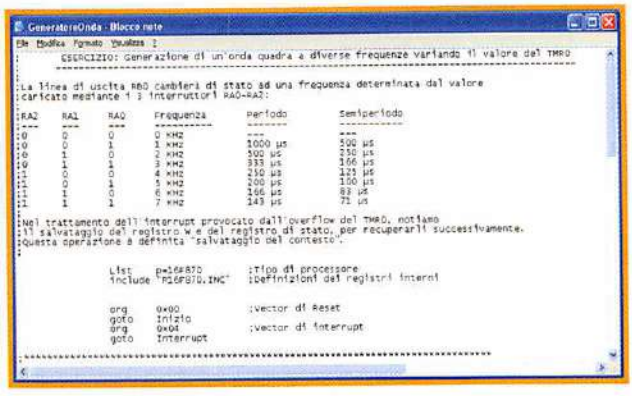
ri degli ingressi e otterremo il valore corrispondente da caricare sul TMR0. Partendo da una frequenza generale di 4 MHz, il TMR0 evolve ogni 1 µs. Selezioneremo un prevedisore di 4. Il valore da caricare sul TMR0 si ottiene dividendo il semiperiodo della frequenza desiderata con il prevedisore. Al valore ottenuto sottrarremo 2, per motivi di sincronismo interno del PIC, convertiremo in esadecimale ed eseguiremo il complemento. Fatto questo, intesteremo il programma principale con l'etichetta inizio e configureremo i dispositivi con cui vogliamo lavorare. Configureremo la porta B come uscita. Programmeremo il registro ADCON1 per fare in modo che la porta A sia a segnali digitali e il registro TRISA per fare in modo che questi segnali siano ingressi. Nel registro OPTION_REG configureremo il prevedisore del temporizzatore e in INT-CON abiliteremo l'interrupt per overflow del TMR0.

Per realizzare il generatore dobbiamo leggere il valore della porta di ingresso. Se i tre



```
GeneratoreOnda - Blocco note
File  Modifica  Formato  Visualizza  ?

;*****************************************************************************
;Tabella: questa routine restituisce il valore da caricare sul TMR0 in base alla frequenza sele-
;zionata. Partendo da una frequenza generale di 4 Mhz, il TMR0 evolve ogni 1 µs.
;Si seleziona un prescaler da 4. Il Valore da caricare sul TMR0 si ottiene dividendo il
;semiperiodo della frequenza desiderata con il prescaler. Al valore ottenuto si
;sottrae 2, per motivi di sincronismo interno del PIC; si converte in hex. e si comple-
;menta.

Tabella:        addwf   PCL,F           ;Calcola lo spostamento della tabella
                retlw   0x00            ;0 KHZ
                retlw   0x06            ;1 KHZ
                retlw   0xc3            ;2 KHZ
                retlw   0xda            ;3 KHZ
                retlw   0xe4            ;4 KHZ
                retlw   0xeb            ;5 KHZ
                retlw   0xee            ;6 KHZ
                retlw   0xf1            ;7 KHZ
```

Tabella per le diverse frequenze.



```
GeneratoreOnda - Blocco note
File  Modifica  Formato  Visualizza  ?

;*****************************************************************************
;       ESERCIZIO: Generazione di un'onda quadra a diverse frequenze variando il valore del TMR0
;*****************************************************************************
;La linea di uscita RB0 cambierà di stato ad una frequenza determinata dal valore
;caricato mediante i 3 interruttori RA0-RA2:
;
;RA2    RA1     RA0     Frequenza       Periodo         Semiperiodo
;---    ---     ---     ---------       -------         -----------
;0      0       0       0 KHZ           ---             ---
;0      0       1       1 KHZ           1000 µs         500 µs
;0      1       0       2 KHZ           500 µs          250 µs
;0      1       1       3 KHZ           333 µs          166 µs
;1      0       0       4 KHZ           250 µs          125 µs
;1      0       1       5 KHZ           200 µs          100 µs
;1      1       0       6 KHZ           166 µs          83 µs
;1      1       1       7 KHZ           143 µs          71 µs
;
;Nel trattamento dell'interrupt provocato dall'overflow del TMR0, notiamo
;il salvataggio del registro w e del registro di stato, per recuperarli successivamente.
;Questa operazione è definita "salvataggio del contesto".
;
                list    p=16f870        ;Tipo di processore
                include "P16F870.INC"   ;Definizioni dei registri interni

                org     0x00            ;vector di Reset
                goto    inizio
                org     0x04            ;vector di Interrupt
                goto    Interrupt
;*****************************************************************************
```

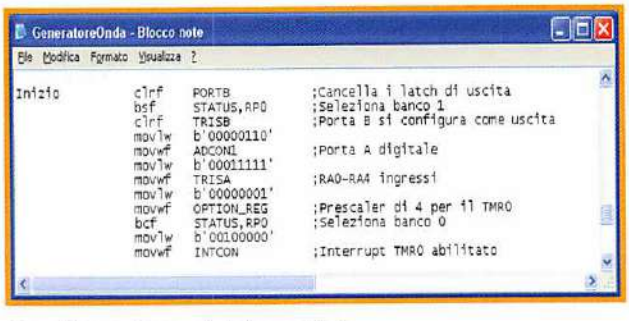
Intestazione del codice.

ingressi sono a zero non si genera nessuna onda, si disabilitano gli interrupt e si salta nuovamente alla lettura degli ingressi. Se al contrario, uno o più ingressi sono a 1, il programma chiamerà la tabella per ottenere il valore corrispondente da caricare sul TMR0. Questo valore lo caricheremo su una variabile che sarà stata definita in precedenza. Abiliteremo gli interrupt e salteremo nuovamente all'inizio del ciclo per leggere di nuovo gli ingressi. Mentre si esegue questo ciclo il temporizzatore continua la sua corsa e quando va in overflow provoca un interrupt.

Nella subroutine di servizio all'interrupt la prima cosa da fare è salvare l'ambiente di lavoro.

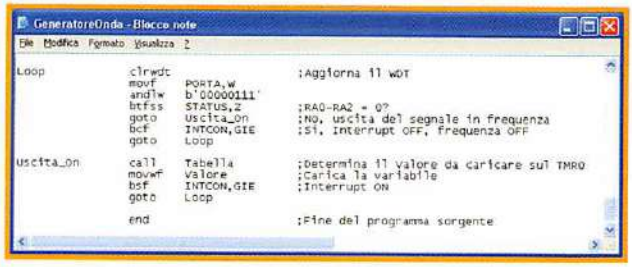
Salvare l'ambiente di lavoro o contesto consiste nel salvare il contenuto dei registri di lavoro e di stato per poi poterlo recuperare quando usciremo dalla subroutine. Sarà necessario, quindi, creare tre nuove variabili per contenere temporaneamente questi registri.

Successivamente scaricheremo il valore del temporizzatore sul suo registro per reinizializzarlo e attiveremo nuovamente l'interrupt resettando il bit del flag.



```
GeneratoreOnda - Blocco note
File  Modifica  Formato  Visualizza  ?

Inizio          clrf    PORTB           ;Cancella i latch di uscita
                bsf     STATUS,RP0      ;Seleziona banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00000110'
                movwf   ADCON1          ;Porta A digitale
                movlw   b'00011111'
                movwf   TRISA           ;RA0-RA4 ingressi
                movlw   b'00000001'
                movwf   OPTION_REG      ;Prescaler di 4 per il TMR0
                bcf     STATUS,RP0      ;Seleziona banco 0
                movlw   b'00100000'
                movwf   INTCON          ;Interrupt TMR0 abilitato
```

Configurazione dei dispositivi.



Programma principale.

Agiremo sull'uscita cambiandone il valore, ovvero imposteremo il valore opposto a quello che aveva in precedenza. Questo lo possiamo fare mediante l'istruzione xorwf. Infine non ci rimane che recuperare il contesto, ovvero i registri di lavoro e di stato, e concludere la subroutine.

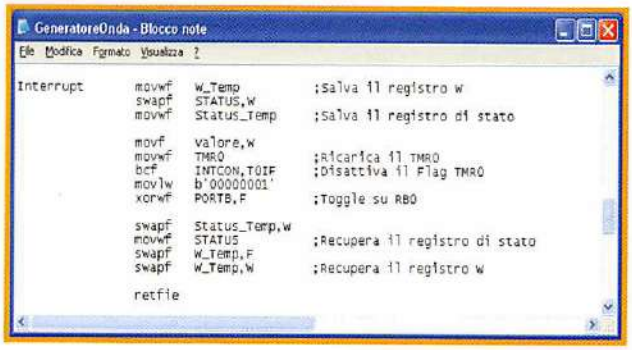
Compilazione

Il file che vi abbiamo spiegato lo potete trovare sul secondo CD allegato all'opera all'interno della cartella "Varie" con il nome "GeneratoreOnda.asm". Questo file, o quello che voi avete creato seguendo le spiegazioni, deve essere compilato e simulato.

Apriamo MPLAB e creiamo un nuovo progetto, ad esempio con il nome "onda.pjt". Aggiungiamo il nostro codice editando il progetto e selezionando Add Node.

Dato che il nome del file è superiore agli 8 caratteri, quando lo selezioneremo apparirà come "GENERA~1.asm". Visualizzeremo a monitor il progetto aprendo il file e procederemo alla compilazione.

Il codice si compila e si assembla con successo generando solamente tre messaggi inerenti ai banchi di memoria.



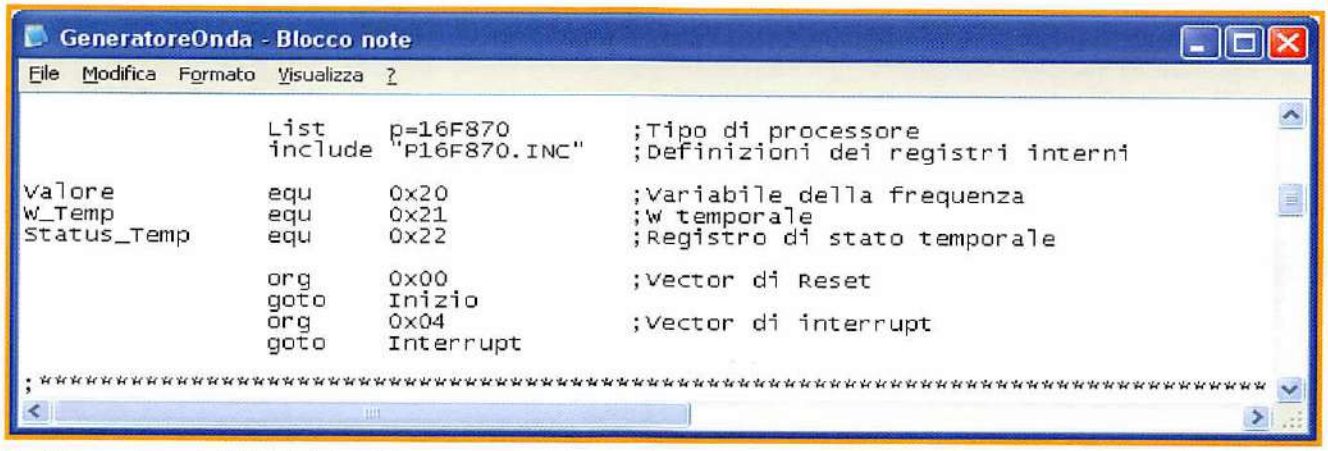
Subroutine di servizio all'interrupt.

Simulazione

Apriamo le finestre abituali: Registri delle Funzioni Speciali e quella di visualizzazione dei registri più importanti. In quest'ultima visualizzeremo PORTB, PORTA e w in binario, la variabile Valore in esadecimale.

Simuleremo una prima esecuzione senza aver attivato i terminali degli ingressi. In questo modo, quando il programma legge lo stato degli ingressi e vede che sono tutti a 0, disattiverà gli interrupt e tornerà a ripetere il ciclo, esattamente come volevamo.

Per ingannare il programma e simulare degli ingressi attivi utilizzeremo la finestra Modify. Quando l'esecuzione arriva alla linea "btfss STATUS,Z", sceglieremo sul campo Address il bit Z e sul campo Data/Optcode gli assegneremo il valore 1 in decimale. Selezioneremo Write per modificare il bit e verifichere-



Definiamo le variabili all'inizio del programma.

```
Build Results
Building GENERA~1.HEX...

Compiling GENERA~1.ASM:
Command line: "C:\PROGRA~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\PROGRA~1\MPLAB\PROGETTI\GENERA~1.ASM"
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\GENERA~1.ASM 73 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\GENERA~1.ASM 75 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\GENERA~1.ASM 77 : Register in operand not in bank 0.  Ensure
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\GENERA~1.ASM 79 : Register in operand not in bank 0.  Ensure

Build completed successfully.
```

Risultato della compilazione.

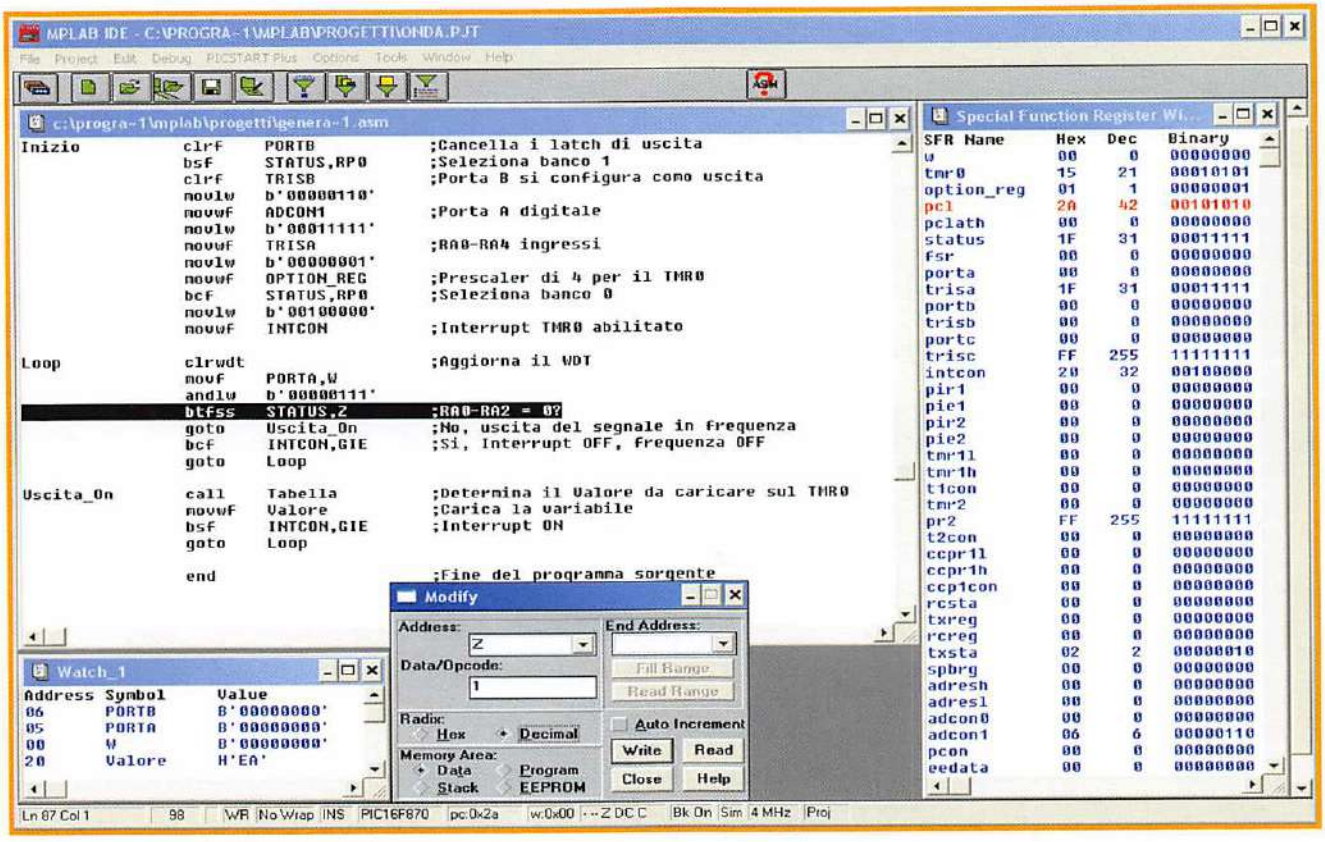
mo come, eseguendo la linea successiva, salteremo direttamente alla routine di interrupt.

Una volta all'interno della routine, faremo la stessa operazione, ma questa volta con la variabile Valore a cui assegneremo un valore qualsiasi tra quelli della tabella (in esadecimale), ad esempio "ea" (5 kHz). Vedremo come vengono assegnati correttamente i valori ai registri e come la porta B di uscita cambia stato.

All'uscita della routine di interrupt il simulatore emette un messaggio di allarme che si può risolvere utilizzando punti di arresto o Break points durante la simulazione.

Provate diverse alternative di simulazione tenendo presente che tutte le simulazioni che comprendono temporizzatori e interrupt sono molto complicate.

È importante che nella simulazione si verifichi il corretto andamento del programma per controllare che non ci siano errori di funzionamento, ma quando si utilizzano i temporizzatori non è possibile sapere se il programma funzionerà in modo adeguato fino a quando non si realizza il montaggio elettrico.



Aspetto di MPLAB durante la simulazione.

Esercizio: il vostro turno, il programma

Se vi appassiona il mondo dell'elettronica e dei microcontroller applicati alla realtà, avrete sicuramente pensato in più di una occasione: potrei sviluppare questa applicazione con un microcontroller? Nella vita quotidiana possiamo trovare molti processi che possono essere automatizzati controllandoli con un PIC.

Enunciato

Vogliamo progettare una macchina di "Il Vostro Turno". In molte strutture commerciali di vendita al pubblico è necessario prendere un numero per essere serviti. Esiste un pannello visualizzatore dove il cliente può vedere il numero che è servito in quel momento e che il dipendente aggiorna mediante un pulsante.

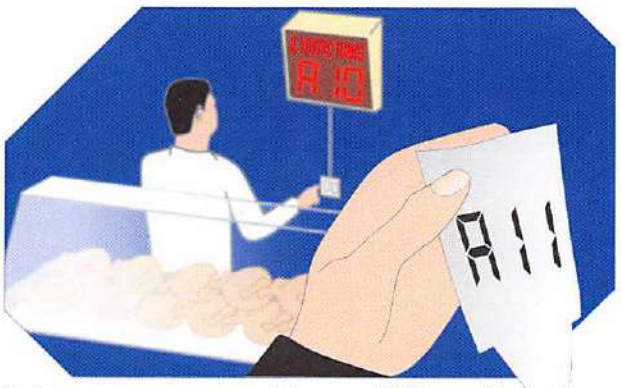
Per realizzare questa applicazione abbiamo bisogno di un pulsante di ingresso e di lavorare con i display a 7 segmenti del laboratorio. Inoltre vogliamo che, a fronte di un buco di tensione, il conteggio riprenda dall'ultimo numero visualizzato prima dell'evento. Questo significa che dovremo lavorare con la memoria EEPROM, dato che non è volatile.

Organigramma

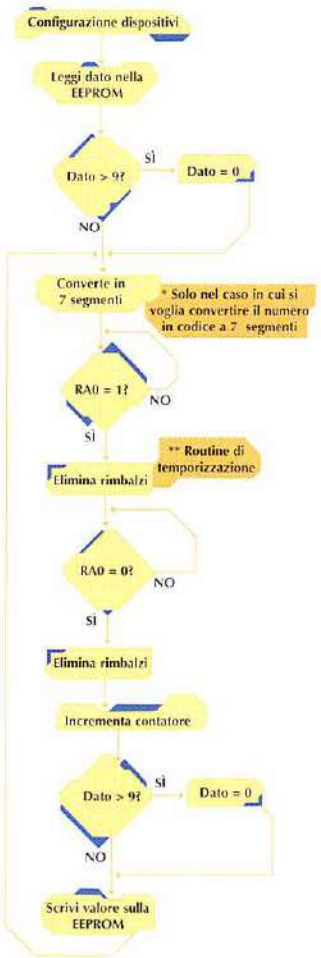
Predisponiamo una soluzione generale all'applicazione. Siamo a conoscenza che la prima cosa da effettuare è la configurazione dei dispositivi con cui vogliamo lavorare (porte e temporizzatori), che per scrivere e leggere la EEPROM dovremo ricorrere alle subroutine già sviluppate (le copieremo da qualche codice già realizzato), e per collegare direttamente il display a 7 segmenti l'uscita deve essere fornita in questo codice per evitare rimbalzi che si possono generare attivando un pulsante, quindi è consigliabile utilizzare una routine di ritardo.

Analizziamo ora ciò che deve fare il programma. Inizialmente si deve leggere il valore scritto nella memoria EEPROM, dato che arriviamo da una mancanza di alimentazione. In base al valore inizieremo a contare da zero o da quest'ultimo valore. Fino a quando non si attiva il pulsante forniremo sull'uscita l'ultimo valore convertito in 7 segmenti, ma nel momento in cui RA0 vale 1, attenderemo il tempo di ritardo, dovuto al cambio di stato del segnale ed entreremo in un ciclo fino a quando il segnale non torna a 0. In questo momento, dopo il suo ritardo corrispondente, dedurremo che è stato generato un impulso e incrementeremo il contatore, visualizzando un nuovo numero e scrivendolo nella memoria EEPROM.

Dopo aver fatto questo torneremo ad attendere un altro cambio di stato su RA0, e ripeteremo la sequenza precedente.



Svilupperemo una macchina per "Il Vostro Turno".



Organigramma dell'applicazione.

vsturno - Blocco note

File Modifica Formato Visualizza ?

```
;       ESERCIZIO: La memoria EEPROM dei dati. La macchina "IL VOSTRO TURNO"
;
;Si tratta di emulare il funzionamento delle macchine tipo "IL VOSTRO TURNO" comuni in molti
;negozi. Sul display LCD si visualizzerà il numero del turno attuale. Questo numero si incrementa
;ad ogni impulso applicato su RA0. Nella memoria EEPROM del PIC16F870 si scrive l'ultimo numero visualizzato,
;in questo modo, in caso di mancanza di tensione (ad esempio), si ricomincia il conteggio dall'ultimo numero.
;
;Si parte dal sistema che inizia per la prima volta, si visualiza lo 0

                List    p=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizioni dei registri interni

Contatore       equ     0x20            ;Variabile per il contatore

                org     0x00            ;Vector di Reset
                goto    Inizio
                org     0x05            ;Salva vector di interrupt

;*****************************************************************************************
```

Intestazione del programma.

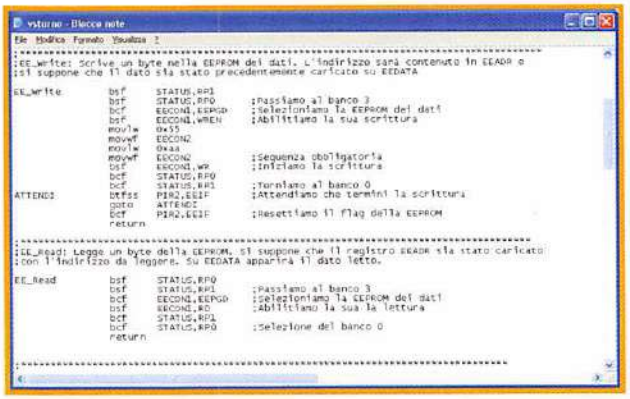
Codice

Siamo pronti per iniziare a progettare il nostro codice e inizieremo con l'intestazione del programma e i relativi commenti, le definizioni e le direttive di organizzazione nella memoria. Possiamo definire anche una variabile che funzioni come contatore.

Subroutine di scrittura e lettura nella EEPROM

In un progetto con i microcontroller difficilmente vi è qualcosa di più prezioso rispetto al tempo di progettazione.

Il tempo è denaro e per questo dobbiamo risparmiarne il più possibile quando sviluppiamo un progetto. Nel primo esercizio in cui abbiamo lavorato con la memoria EEPROM abbiamo realizzato una subroutine di scrittura e una di lettura. Ogni volta che lavoreremo con la memoria EEPROM potremo utilizzare queste subroutine, anche se in qualche caso dovremo fare qualche leggero ritocco. In questo modo guadagnamo tempo ed evitiamo possi-



vsturno - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************************
;EE_write: Scrive un byte nella EEPROM dei dati. L'indirizzo sarà contenuto in EEADR e
;si suppone che il dato sia stato precedentemente caricato su EEDATA

EE_write        bsf     STATUS,RP1
                bsf     STATUS,RP0      ;Passiamo al banco 3
                bcf     EECON1,EEPGD    ;Selezioniamo la EEPROM dei dati
                bsf     EECON1,WREN     ;Abilitiamo la sua scrittura
                movlw   0x55
                movwf   EECON2
                movlw   0xaa
                movwf   EECON2          ;Sequenza obbligatoria
                bsf     EECON1,WR       ;Iniziamo la scrittura
                bcf     STATUS,RP0
                bcf     STATUS,RP1      ;Torniamo al banco 0
ATTENDI         btfss   PIR2,EEIF       ;Attendiamo che termini la scrittura
                goto    ATTENDI
                bcf     PIR2,EEIF       ;Resettiamo il flag della EEPROM
                return

;*****************************************************************************************
;EE_Read: Legge un byte della EEPROM. Si suppone che il registro EEADR sia stato caricato
;con l'indirizzo da leggere. Su EEDATA apparirà il dato letto.

EE_Read         bsf     STATUS,RP0
                bsf     STATUS,RP1      ;Passiamo al banco 3
                bcf     EECON1,EEPGD    ;Selezioniamo la EEPROM dei dati
                bsf     EECON1,RD       ;Abilitiamo la sua la lettura
                bcf     STATUS,RP1
                bcf     STATUS,RP0      ;Selezione del banco 0
                return

;*****************************************************************************************
```

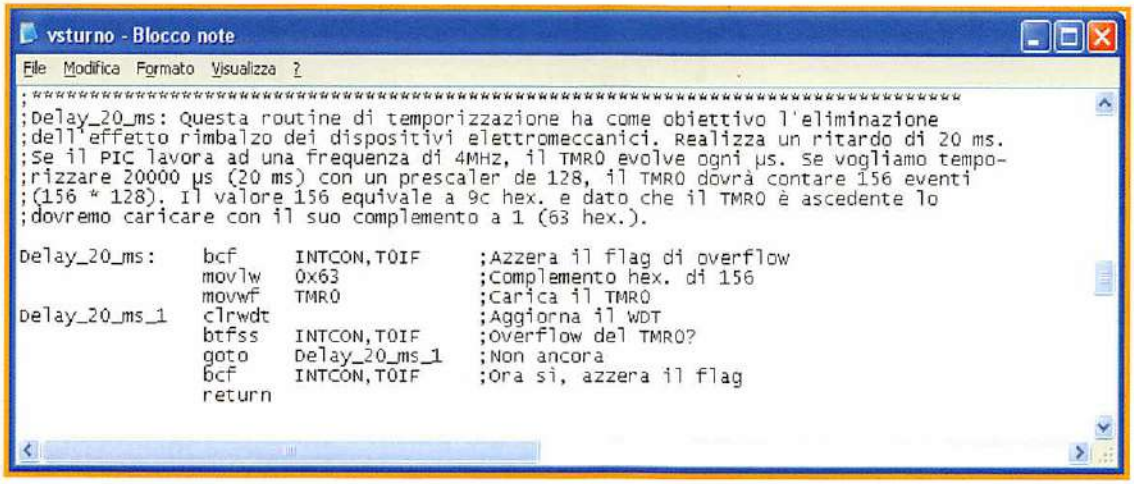
Subroutine di scrittura e lettura della EEPROM.

vsturno - Blocco note

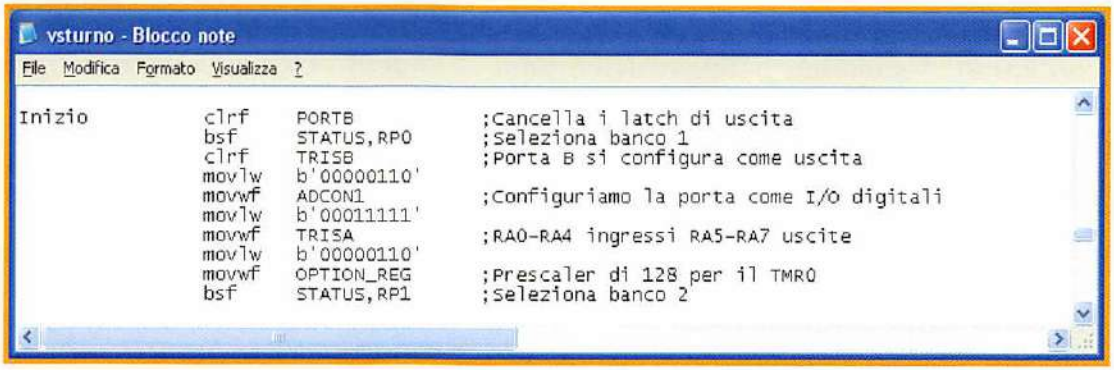
File Modifica Formato Visualizza ?

```
;*****************************************************************************************
;Tabella: Questa routine converte il codice BCD presente sui bit meno significativi
;del reg. W nel loro equivalente a 7 segmenti. Il codice a 7 segmenti rimane anche
;sul reg. W

Tabella:        addwf   PCL,F                   ;Spostamento sulla tabella
                retlw   b'00111111'             ;Digit 0
                retlw   b'00000110'             ;Digit 1
                retlw   b'01011011'             ;Digit 2
                retlw   b'01001111'             ;Digit 3
                retlw   b'01100110'             ;Digit 4
                retlw   b'01101101'             ;Digit 5
                retlw   b'01111101'             ;Digit 6
                retlw   b'00000111'             ;Digit 7
                retlw   b'01111111'             ;Digit 8
                retlw   b'01100111'             ;Digit 9

;*****************************************************************************************
```

Subroutine di conversione in codice a 7 segmenti.



Subroutine di ritardo anti-rimbalzo.



Configurazione dei dispositivi.

bili complicazioni. Nella figura possiamo vedere le due subroutine che verranno incluse nel programma.

Subroutine di conversione in codice a 7 segmenti

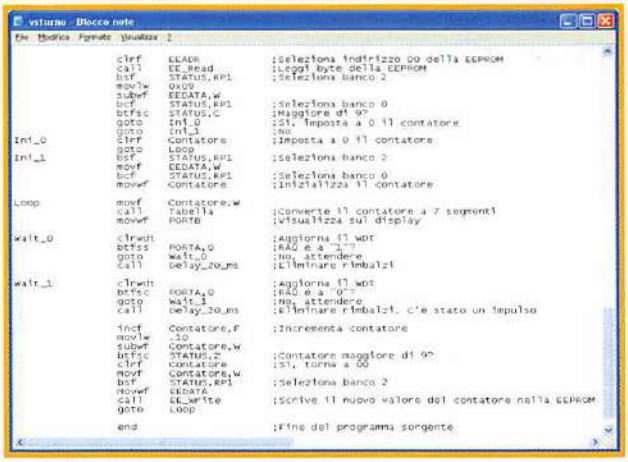
Come abbiamo visto per la memoria EEPROM, per convertire un numero in codice a 7 segmenti possiamo utilizzare nel nostro codice una subroutine che abbiamo realizzato nell'esercizio precedente. Questa parte è opzionale, dato che nell'enunciato non si specifica il formato dell'uscita, quindi potremo fornire l'uscita in codice a 7 segmenti oppure in binario, e far convertire il dato ai driver del display a 7 segmenti. Nel caso in cui si volesse realizzare quest'ultima opzione, sarebbe sufficiente inserire ";" su ogni linea della subroutine e a ogni chiamata a questa con l'istruzione "call Tabella". Impostando il segno "punto e virgola" il compilatore considererà queste linee come commenti.

Subroutine di ritardo anti-rimbalzo

Quando attiviamo un elemento elettromeccanico possiamo produrre l'effetto rimbalzo. Questo fa sì che in un tempo molto breve il segnale possa cambiare di stato fino a quando si stabilizza. Il PIC può rilevare queste fluttuazioni e interpretarle in modo sbagliato. Dobbiamo inserire una subroutine che generi un ritardo ogni volta che c'è un cambio di stato sul segnale di ingresso, come per le subroutine precedenti, anche questa è già stata utilizzata in altri casi. Copieremo la subroutine e la inseriremo nel nostro programma.

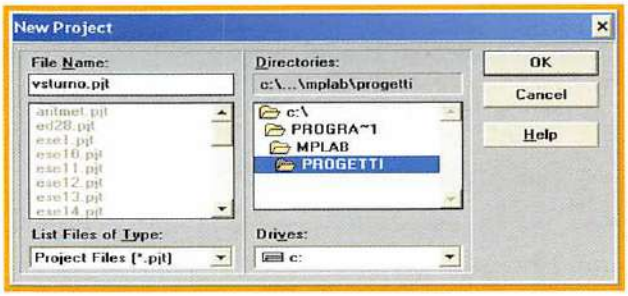
Programma principale

Il programma principale deve iniziare configurando i dispositivi con cui vogliamo lavorare. Sappiamo che la porta B sarà la porta di uscita e la A di ingresso, per i segnali digitali, e che dobbiamo configurare il registro OPTION_REG per lavorare con il TMR0 che utiliz-



Codice che risolve l'applicazione.

za la subroutine di ritardo. Con i dispositivi configurati seguiremo il flusso dell'organigramma codificandolo in istruzioni. Leggeremo la EEPROM e osserveremo il valore che contiene. Visualizziamo e attendiamo che si attivi l'ingresso. In quel momento dovremo eseguire il ciclo necessario per considerare valido l'impulso (ritardo, attesa di ritorno a 0, ritardo). Successivamente incrementeremo il contatore verificando che il suo valore sia minore di 9, dato che il valore del conteggio successivo è 0. Infine passeremo il valore alla EEPROM e torneremo a ripetere il ciclo dal punto in cui è stata chiamata la tabella portando il valore sull'uscita.



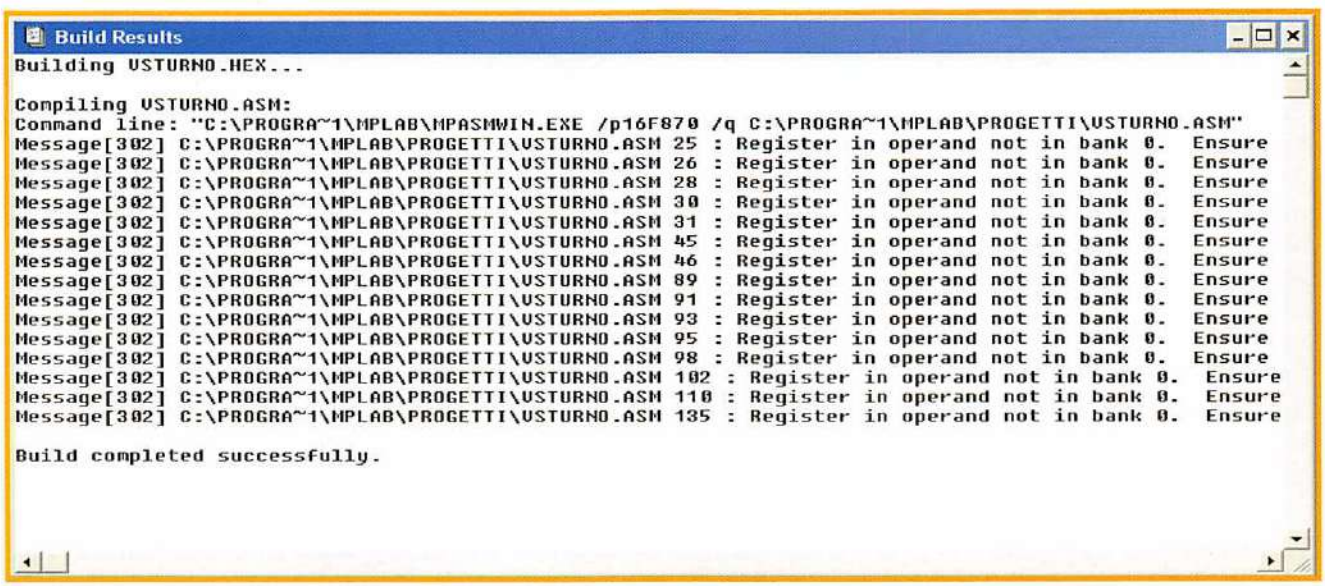
Creiamo un progetto su MPLAB.

Compilazione

Come avete potuto verificare abbiamo risolto un'applicazione complessa sviluppando un codice semplice e inserendo delle subroutine che avevamo già sviluppato per esercizi precedenti. Ora dobbiamo verificare di non aver commesso errori sia nell'includere delle subroutine che nell'indirizzarci a esse e anche nelle strutture sintattiche delle istruzioni.

Apriamo MPLAB, creiamo un nuovo progetto e alleghiamo a esso il nostro codice.

Fatto questo, apriamo il file con il codice per visualizzarlo sul monitor e correggerlo nel caso fosse necessario, selezioneremo poi l'opzione Build All. Il codice si compila senza errori anche se il compilatore ci informa circa 15 possibili errori di indirizzamento dei registri nella memoria. Nell'immagine possiamo osservare il risultato ottenuto dalla compilazione del programma.



Risultato della compilazione.

Display LCD

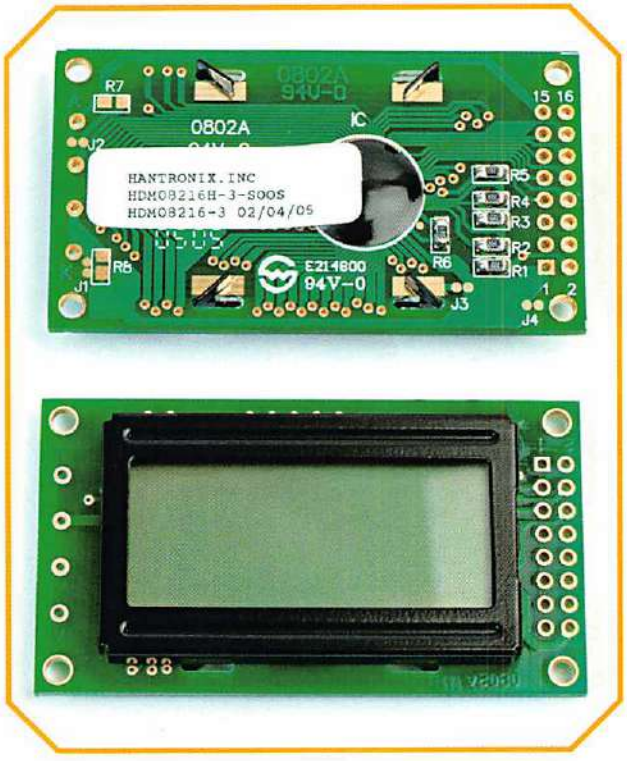
Affrontiamo ora un altro degli elementi che faranno parte del laboratorio, il display LCD. In molti progetti basati sul microcontroller, si deve interagire con l'utente e l'impiego di un LCD è la soluzione più utilizzata. Studieremo la gestione di questi display e faremo pratica con alcuni esempi.

Caratteristiche generali

I display o moduli visualizzatori LCD sono composti da un video con cristalli liquidi LCD che ha una matrice di 16, 32 o 40 caratteri da 5 x 8 pixel e un microcontroller che li gestisce. Possiamo quindi dire che questa periferica è "intelligente", dato che possiede un proprio microcontroller di governo e gestione. Oltre ai dati da visualizzare potremo fornire all'LCD ordini che saranno interpretati e che genereranno azioni conseguenti. Nella figura possiamo vedere i due lati di un display LCD. Sul lato posteriore distinguiamo la circuiteria elettronica in tecnologia SMD propria del display, notiamo inoltre il microprocessore protetto da una resina nera.

Le caratteristiche generali di un modulo LCD sono:

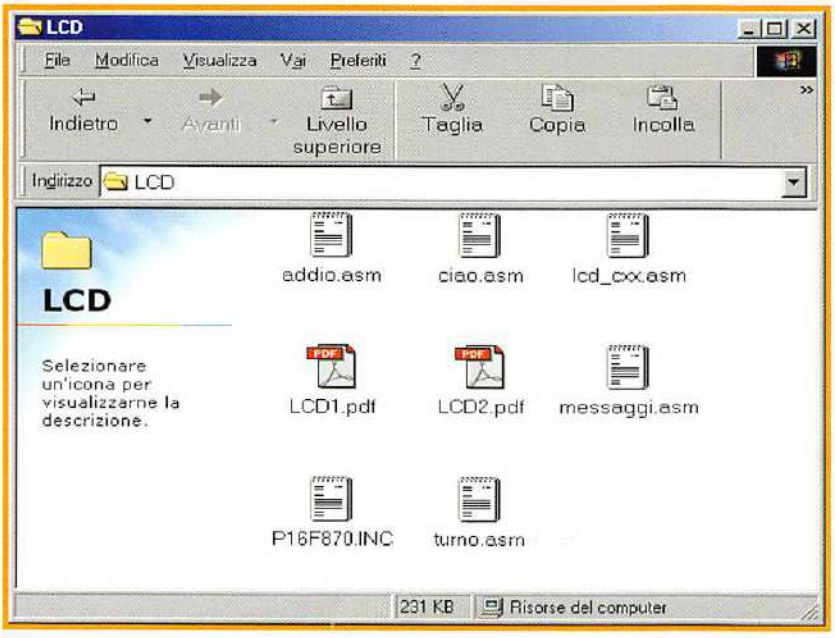
- Display con caratteri ASCII, oltre ai caratteri (ideogrammi) Kanji e a quelli greci.
- Spostamento dei caratteri verso sinistra o verso destra.
- Fornisce l'indirizzo della posizione assoluta o relativa del carattere.
- Memoria di un numero determinato di caratteri per linea di display.
- Movimento del cursore e modifiche dell'aspetto.



Aspetto di un display LCD da entrambi i lati.

LOWER 4 BITS / UPPER 4 BITS	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p		—	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	ß	θ
0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	ｊ	𝕏
1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	㎏	万
1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
1101	(6)	−	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	°	ö	■

Caratteri che si possono visualizzare sull'LDC.



Documentazione sul display LCD.

– Collegamento a un processore utilizzando un'interfaccia a 8 bit.

Il display LCD che utilizzeremo nel laboratorio è della HANTRONIX e ha 8x2 caratteri, cioè due linee da otto caratteri ognuna. Dalla pagina Web del costruttore potremo scaricare tutte le informazioni che fanno riferimento al nostro display. Il display è il modello HDM08216H-3 della ditta HANTRONIX.

Nel secondo CD potrete trovare dei file in formato ".pdf" che contengono le caratteristiche del nostro display LCD di HANTRONIX.

Si tratta di una documentazione che ci sarà utile soprattutto dopo aver imparato a lavorare con il display LCD.

Adattamento del display LCD

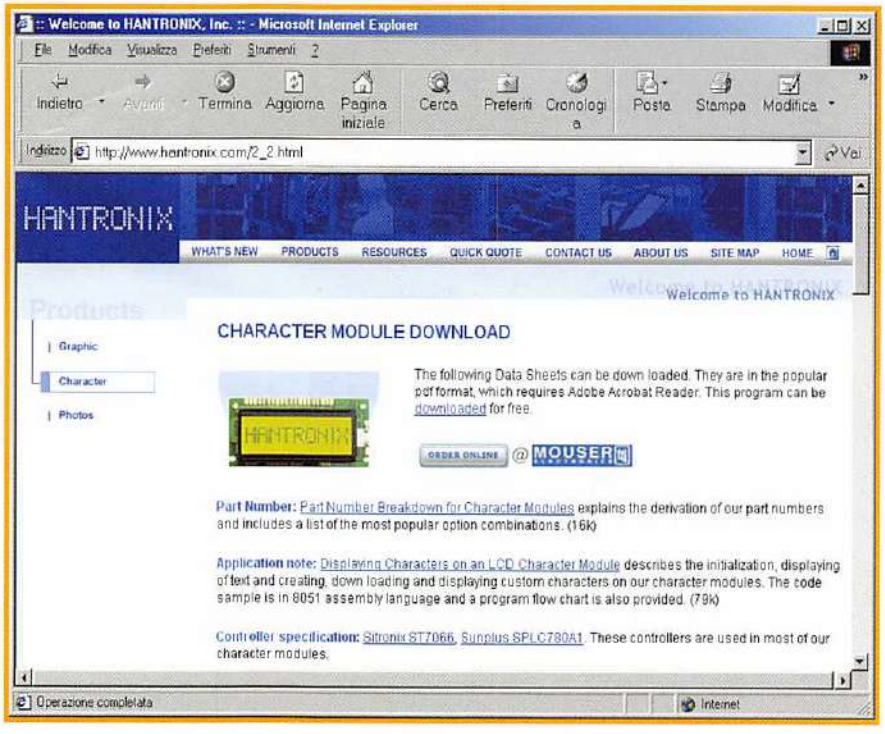
Il display a matrice di cristalli liquidi di HANTRONIX dispone di uno spazio per 16 caratteri, ognuno di essi è composto da 5x8 pixel e il cursore, distribuiti su due file da otto caratteri ognuna. Si può collegare direttamente a un microprocessore o a un microcontroller mediante un bus da 8 bit.

Il consumo del display è molto contenuto dato che il consumo massimo è inferiore a 7,5 mW. Questo, unito alla semplicità della gestione, lo rende idoneo per essere utilizzato in applicazioni che richiedono una capacità di visualizzazione medio-piccola.

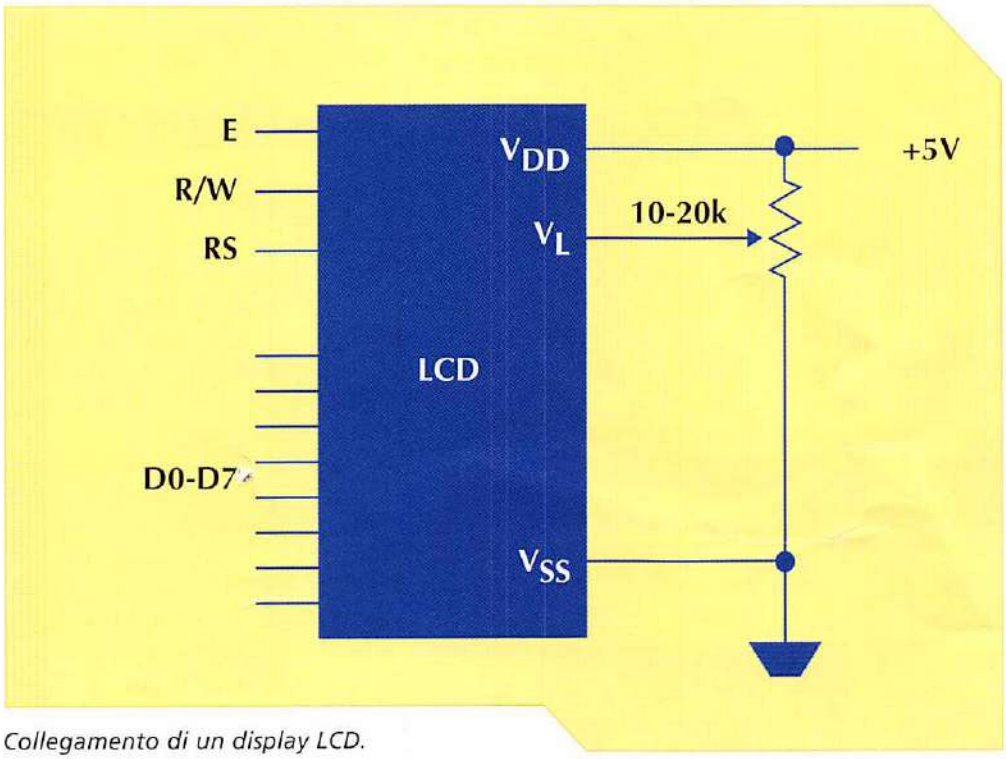
Il modulo LCD ha 14 terminali, la cui descrizione è riportata nella tabella della pagina successiva.

L'alimentazione è di +5 V e la regolazione del contrasto si ottiene mediante un partitore sulla tensione di alimentazione + 5 V utilizzando un potenziometro da 10 KΩ.

Se separiamo le linee che si utilizzano per l'alimentazione e il contrasto, ci rimangono 11 linee di ingressi/uscite. Di queste, 3 linee so-



Pagina Web del costruttore.



Collegamento di un display LCD.

no di controllo (RS, R/W ed E), mentre le rimanenti sono linee di dati.

Le linee di dati passano in stato di alta impedenza quando il display LCD non è attivato, cioè vanno in tristate.

N.	Simbolo	Descrizione
1	VSS	Massa
2	VDD	+ 5 V
3	VO	Regolazione del contrasto
4	RS	Selezione modo
5	R/W	Lettura/Scrittura
6	E	Segnale di abilitazione
7	DB0	Linea dei dati 1
8	DB1	Linea dei dati 2
9	DB2	Linea dei dati 3
10	DB3	Linea dei dati 4
11	DB4	Linea dei dati 5
12	DB5	Linea dei dati 6
13	DB6	Linea dei dati 7

Terminali di collegamento del modulo LCD.

Le linee di controllo dell'LCD

Come abbiamo detto esistono tre linee di controllo sul modulo LCD. La linea E (Enable), terminale 6, è la linea di abilitazione. Quando questa linea è a livello alto, abilitata, il display testa lo stato delle altre due linee e funzionerà di conseguenza. Se questa è disabilitata, non validata, ignorerà il resto delle linee.

La linea R/W terminale 5, indica se si devono leggere o scrivere dei dati sul display LCD. Nelle applicazioni in cui si utilizza il display esclusivamente come elemento di visualizzazione, questa linea si potrà collegare direttamente a massa (0 V) lasciando il display configurato permanentemente in modo scrittura.

Il terminale 4 o linea RS di selezione del modo, determina quando i dati che si inviano all'LCD devono essere gestiti come comandi o come caratteri.

Una sequenza di scrittura sul display si deve completare con i passaggi indicati qui di seguito:

1-. Linea RS a 0 o 1 in base al modo desiderato (comandi o caratteri).
2-. Linea R/W a 0 (scrittura).
3-. Linea E a 1 (abilitato).

Linea	0 - Disattivato	1 – Attivato
E	Disabilitato	Abilitato
R/W	Scrittura	Lettura
RS	Comando	Caratteri

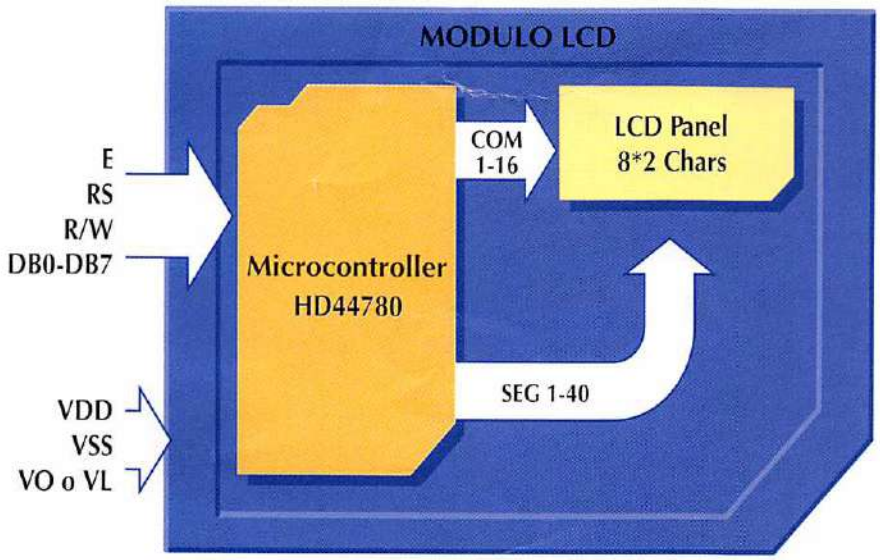
Descrizione dettagliata delle linee di controllo.

DÍGIT	1	2	3	4	5	6	7	8	Posizione sul display
1ª Linea	00	01	02	03	04	05	06	07	
2ª Linea	40	41	42	43	44	45	46	47	Indirizzo nella memoria DDRAM

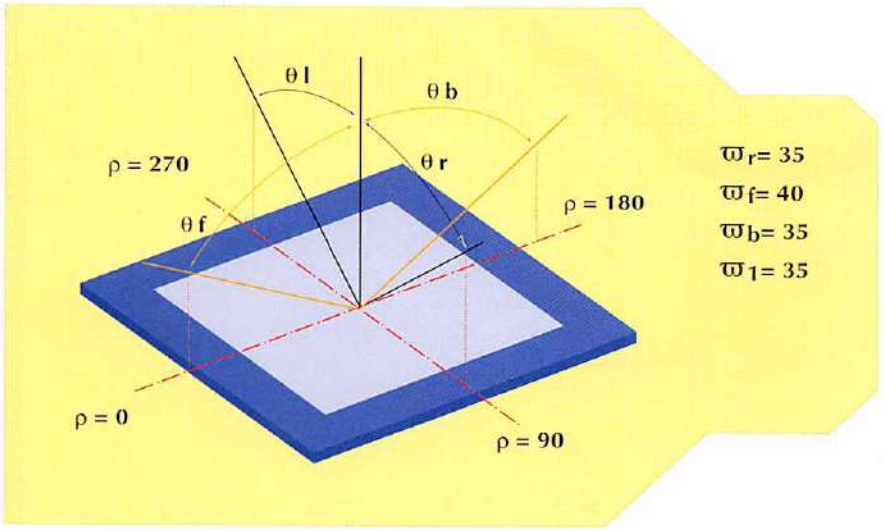
Indirizzi della memoria DDRAM dei digit del display.

4-. Scrivere sul bus dei dati del modulo LCD.
5-. Linea E a 0 (disabilitato).

La lettura segue la stessa frequenza, tranne il passo 2, dove la linea R/W deve essere impostata a 1.



Schema a blocchi del display LCD AMPIRE.



Range dell'angolo visivo consentito dal display LCD.

Funzionamento

Quando si accende il modulo LCD si genera un reset e si resta in attesa di istruzioni. Normalmente queste istruzioni accendono il display e il cursore, e configurano il modo di scrittura da sinistra a destra.

I caratteri sono scritti nella memoria RAM dell'LCD, che si chiama DDRAM. Indipendentemente dal numero dei caratteri che ci sono sul display, la DDRAM è composta da 80 bit. I caratteri non visibili si possono visualizzare se ci spostiamo sul display. L'LCD dispone anche di 64 byte del generatore di caratteri CGRAM.

Sul display LCD non si può scrivere come se si trattasse di indirizzi di memoria, dato che il controller impiega da 40 a 120 µs per completare una lettura o una scrittura. Altre operazioni sono più lente e possono raggiungere 5 ms. Per evitare di far attendere il PIC, LCD ha un'istruzione di 1 µs che legge l'indirizzo del contatore e del flag di occupato. Quando il flag BF di occupato (busy) è a 1, LCD non può né leggere né scrivere.

Continueremo vedendo le caratteristiche del nostro modulo LCD con i processi di inizializzazione, scrittura e lettura, vedendo i comandi che si possono inviare, come si inviano e il loro indirizzamento. Lavoreremo anche con la libreria di subroutine "lcd_cxx.inc", in modo che quando faremo pratica con gli esercizi, questi risultino i più semplici possibili.

Display LCD (II)

Prima di fare i nostri primi programmi in cui il PIC interagirà con il display LCD, dobbiamo capire come funziona il display. Vedremo ora in modo approfondito alcune caratteristiche del modulo LCD e la libreria "lcd_cxx.inc".

Abilitazione dell'LCD

Abbiamo visto che uno dei pin del modulo LCD, nello specifico il pin 6, serve per abilitarlo. Il segnale di "Enable" o abilitazione, sarà attivo con un livello alto, in caso contrario le linee di I/O passano in stato di alta impedenza, cosa che scollega il modulo LCD dal PIC. Il segnale si attiverà quando si desidera scrivere o leggere dati o comandi sul display. La durata dell'impulso necessario deve essere di 230 ns, anche se per standardizzare il metodo di lavoro di solito si considera un minimo di 500 ns. Nella libreria "lcd_cxx.inc", che si può trovare nella cartella LCD del secondo CD fornito con l'opera, esiste una routine che invia impulsi all'ingresso di abilitazione.

Selezione del modo (linea RS)

La linea 4 dell'LCD o linea RS, seleziona fra il modo comando (RS = 0) o modo di lettura/scrittura di carattere (RS = 1). Nella libreria "lcd_cxx.inc" è inserita una routine per scrivere i comandi sui registri del LCD. La chiamata alla routine "LCD_REG" assicura che RS sia a livello basso prima di inviare l'impulso alla linea di abilitazione.

Esiste anche una routine per scrivere dati nella memoria dell'LCD. Richiamando "LCD_DATO" possiamo scrivere un carattere nella DDRAM o CGRAM dell'LCD, verificando che la linea RS sia posta a livello alto prima di inviare l'impulso alla linea di abilitazione.

Lettura/Scrittura nell'LCD

La linea di controllo R/W, linea 5, determina se si legge il contenuto della memoria del LCD (R/W = 1) o se si scrive in essa (R/W = 0). Prima di utilizzare questa linea è necessario inviare

lcd_cxx.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;LCD_E: Impulso di Enable.
;
LCD_E           ENABLE                  ;Attiva E
                nop                     ;Pausa extra di 250 ns
                DISABLE                 ;Disattiva E
                return
;
;*****************************************************************************
```

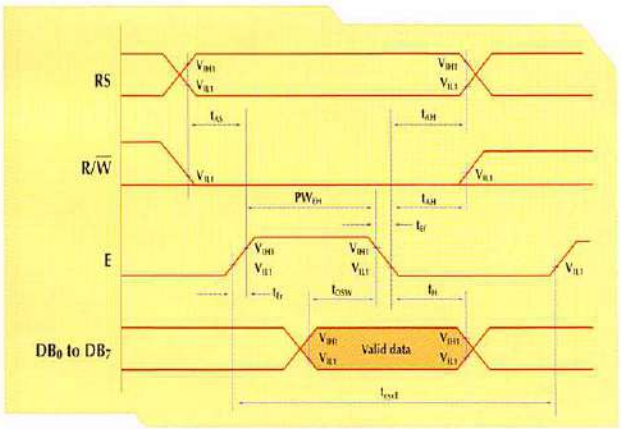
Routine per abilitare l'LCD.

lcd_cxx.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;LCD_DATO: Scrittura dei dati in DDRAM o CGRAM. Invia il dato presente su W
;
LCD_DATO        OFF_COMANDO             ;Disattiva RS (modo comando)
                movwf   PORTB           ;Valore ASCII da portare sulla porta B
                call    LCD_BUSY        ;Attende che si liberi LCD
                ON_COMANDO              ;Attiva RS (modo dato)
                goto    LCD_E           ;Genera impulso di E

;*****************************************************************************
;LCD_REG: Scrittura di comandi nel LCD. Invia il comando presente in W

LCD_REG         OFF_COMANDO             ;Disattiva RS (modo comando)
                movwf   PORTB           ;Codice di comando
                call    LCD_BUSY        ;LCD libero?
                goto    LCD_E           ;Si.Genera impulso di E

;*****************************************************************************
```

Routine di selezione del modo.



Operazione di scrittura.

un impulso di abilitazione. Nella figura si può vedere un esempio del processo di scrittura nel modulo LCD. Osservate l'evoluzione delle linee di controllo E, RS e R/W. Prima di dare l'impulso di abilitazione la linea R/W si trova a livello basso, e la linea RS è a livello corrispondente al modo desiderato (comando o carattere).

Osservate anche in che momento si iniziano a scrivere i dati.

Nella tabella riportata in alto a destra, sono evidenziati in dettaglio i tempi presenti nel grafico.

Il processo di lettura è simile a quello appena visto, con la differenza che in lettura la linea R/W passa a livello alto.

Lettura dell'indirizzo del contatore e del flag di occupato

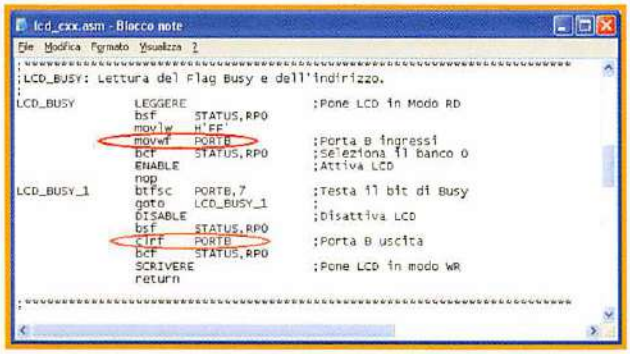
Nel fascicolo precedente abbiamo spiegato che per fare in modo che il PIC non debba attendere, esiste una routine all'interno della libreria "lcd_cxx.inc" che legge l'indirizzo del contatore e del flag di occupato (BF). Quando questo flag è a 1 si considera che il modulo LCD sia occupato, quindi non si può né leggere né scrivere.

Descrizione	Simbolo	Tempo (ns) (VDD = 5 V)	
		Min	Max
Tempo del ciclo di abilitazione	tcycE	500	---
Ampiezza dell'impulso di abilitazione	PWEH	230	---
Tempo di salita/ discesa dell'impulso	tEr,tEf	---	20
Tempo di indirizzamento (di RS e R/W a E)	tAS	40	---
Tempo di ritenzione dell'indirizzamento (di E a RS e R/W)	tAH	10	---
Tempo di configurazione dell'ingresso dei dati	tDSW	80	---
Tempo di ritenzione dell'ingresso dei dati	tH	10	---

Dettaglio dei tempi nelle operazioni di scrittura.

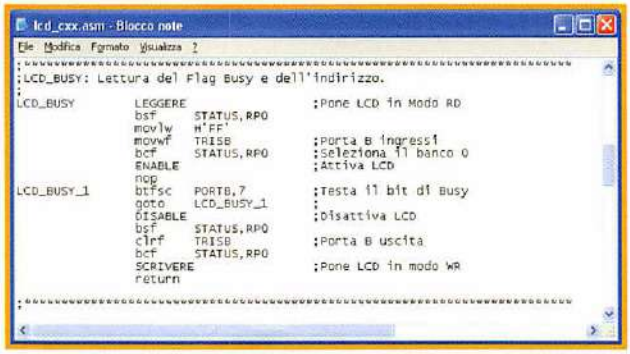
Come possiamo vedere nelle routine presentate, per inviare un carattere o un comando all'LCD prima di abilitare il display si esegue una chiamata alla subroutine "LCD_BUSY". Questa routine configura la porta B per la lettura, ottiene l'indirizzo del contatore e lo stato del flag di occupato e se questo è a 0, configura il modulo LCD per la scrittura.

Avrete notato a questo punto l'errore presente all'interno di questa subroutine. Pote-



```
lcd_cxx.asm - Blocco note
File Modifica Formato Visualizza ?
;LCD_BUSY: Lettura del Flag Busy e dell'indirizzo.
;
LCD_BUSY        LEGGERE                 ;Pone LCD in Modo RD
                bsf     STATUS,RP0
                movlw   H'FF'
                movwf   PORTB           ;Porta B ingressi
                bcf     STATUS,RP0      ;Seleziona il banco 0
                ENABLE                  ;Attiva LCD
                nop
LCD_BUSY_1      btfsc   PORTB,7         ;Testa il bit di Busy
                goto    LCD_BUSY_1      ;
                DISABLE                 ;Disattiva LCD
                bsf     STATUS,RP0
                clrf    PORTB           ;Porta B uscita
                bcf     STATUS,RP0
                SCRIVERE                ;Pone LCD in modo WR
                return
```

Routine per leggere l'indirizzo del contatore e del flag di occupato.



```
lcd_cxx.asm - Blocco note
File Modifica Formato Visualizza ?
;LCD_BUSY: Lettura del Flag Busy e dell'indirizzo.
;
LCD_BUSY        LEGGERE                 ;Pone LCD in Modo RD
                bsf     STATUS,RP0
                movlw   H'FF'
                movwf   TRISB           ;Porta B ingressi
                bcf     STATUS,RP0      ;Seleziona il banco 0
                ENABLE                  ;Attiva LCD
                nop
LCD_BUSY_1      btfsc   PORTB,7         ;Testa il bit di Busy
                goto    LCD_BUSY_1      ;
                DISABLE                 ;Disattiva LCD
                bsf     STATUS,RP0
                clrf    TRISB           ;Porta B uscita
                bcf     STATUS,RP0
                SCRIVERE                ;Pone LCD in modo WR
                return
```

Routine dopo aver effettuato la correzione.

Comando	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Cancella display	0	0	0	0	0	0	0	0	0	1
Cursore all'inizio	0	0	0	0	0	0	0	0	1	x
Modo introduzione	0	0	0	0	0	0	0	1	I/D	x
Display ON/OFF	0	0	0	0	0	0	1	D	C	B
Modo spostamento	0	0	0	0	0	1	S/C	R/L	x	x
Funzione	0	0	0	0	1	DL	Linee	Font	x	x
Indirizzo CGRAM	0	0	0	1	Indirizzo CGRAM					
Indirizzo DDRAM	0	0	1	Indirizzo DDRAM						
Lettura occupato e indirizzo contatore	1	1	BF	Indirizzo DDRAM						
Scrive RAM	1	0	Scrive Dato							
Legge RAM	1	1	Legge Dato							

Comandi di controllo del modulo LCD.

te vedere come si cerca di configurare la porta B come ingresso o come uscita modificando il registro PORTB invece di TRISB. Editate la libreria, modificate le due linee di questa subroutine, cambiando PORTB con TRISB e salvate il file con i cambiamenti.

Comandi del display LCD

Il modulo LCD contiene un microcontroller HD44780 che riconosce i comandi riportati nella tabella in alto.

– Cancella display: cancella il display e riporta il cursore nella posizione iniziale.

– Cursore all'inizio: riporta il cursore nella posizione iniziale mantenendo intatto il contenuto del display.

– Modo istruzione, display ON/OFF e modo spostamento: configurano il cursore e lo spostamento della videata in base ai bit, la cui funzione è specificata nella tabella.

– Funzione: DL indica la lunghezza dei dati dell'interfaccia, N il numero delle linee del display e F il tipo di carattere.

– Indirizzo CGRAM: dopo l'esecuzione di questo comando colloca il dato inviato o ricevuto nella CGRAM.

– Indirizzo DDRAM: la stessa funzione spiegata precedentemente ma nella DDRAM.

– Flag di occupato BF: legge BF indicando se è in corso un'operazione interna e legge anche l'indirizzo del contatore.

Bit	0	1
I/D	Decrementa la posizione del cursore	Incrementa la la posizione del cursore
S	Senza spostamento	Con spostamento
D	Display OFF	Display ON
C	Cursore OFF	Cursore ON
B	Lampeggio del cursore OFF	Lampeggio del cursore ON
S/C	Muove il cursore	Sposta la videata
R/L	Sposta a sinistra	Sposta a destra
DL	Interfaccia da 4 bit	Interfaccia da 8 bit
Linee	Una linea	Due linee
Font	5x7 punti	5x10 punti
BF	Può accettare l'istruzione	Operazione interna in corso

Descrizione dei bit dei comandi.

– Scrive/Legge RAM: scrive/legge dati dalla RAM (DDRAM o CGRAM).

Inizializzazione

È necessario inizializzare il display LCD prima di utilizzarlo. Inviamo al display il comando Function con il valore "00111000", cioè una

lcd_cxx.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;LCD_INI: Inizializzazione del LCD inviando il comando "Function Set" 3 volte consecutive
;con un intervallo di 5 ms. LCD si cancella e il cursore va nella prima posizione

LCD_INI         movlw   b'00111000'
                call    LCD_REG         ;Codice di istruzione
                call    LCD_DELAY       ;Temporizza
                movlw   b'00111000'
                call    LCD_REG         ;Codice di istruzione
                call    LCD_DELAY       ;Temporizza
                movlw   b'00111000'
                call    LCD_REG         ;Codice di istruzione
                call    LCD_DELAY       ;Temporizza
                movlw   b'00000001'     ;Cancella LCD e Home
                call    LCD_REG
                return

;*****************************************************************************
```

Routine di inizializzazione del display.

lcd_cxx.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;LCD_DELAY: Routine di temporizzazione di 5 ms. Si utilizzano le variabili LCD_Temp_1
;e LCD_Temp_2 al posto del TMR0. Quest'ultimo rimane libero per le applicazioni dell'utente

LCD_DELAY:      clrwdt
                movlw   .10
                movwf   Lcd_Temp_1
                clrf    Lcd_Temp_2
LCD_DELAY_1:    decfsz  Lcd_Temp_2,F
                goto    LCD_DELAY_1
                decfsz  Lcd_Temp_1,F
                goto    LCD_DELAY_1
                return
```

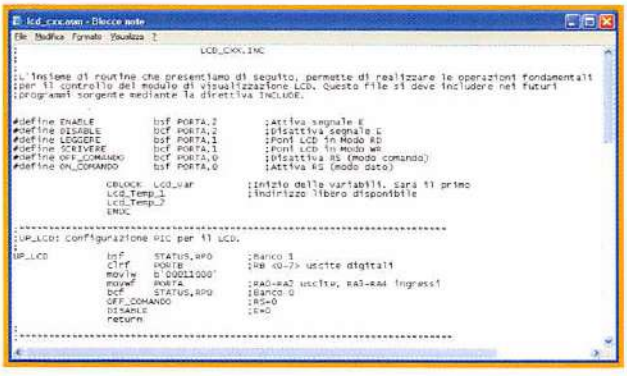
Routine di temporizzazione.

configurazione da 8 bit di dati, due linee di display con una matrice del carattere composta da 5x7 pixel. Il display rimarrà configurato con questi parametri.

L'inizializzazione ha un ordine che dobbiamo mantenere.

Nella routine "LCD_INI" della libreria "lcd_cxx.inc" possiamo vedere questo curioso ordine. Si invia tre volte questo comando con intervalli di tempo determinati dalla routine "LCD_DELAY".

Fatto questo la routine invia il comando cancella display. Nella routine di temporizzazione si attendono 5 ms, ma senza utilizzare i temporizzatori proprio del PIC in modo che questi rimangano liberi e a disposizione dell'utente.



```
                        LCD_CXX.INC

;L'insieme di routine che presentiamo di seguito, permette di realizzare le operazioni fondamentali
;per il controllo del modulo di visualizzazione LCD. Questo file si deve includere nei futuri
;programmi sorgente mediante la direttiva INCLUDE.

#define ENABLE          bsf PORTA,2     ;Attiva segnale E
#define DISABLE         bcf PORTA,2     ;Disattiva segnale E
#define LEGGERE         bsf PORTA,1     ;Pone LCD in Modo RD
#define SCRIVERE        bcf PORTA,1     ;Pone LCD in Modo WR
#define OFF_COMANDO     bcf PORTA,0     ;Disattiva RS (modo comando)
#define ON_COMANDO      bsf PORTA,0     ;Attiva RS (modo dato)

                CBLOCK  Lcd_Var         ;Inizio delle variabili, sarà il primo
                Lcd_Temp_1              ;indirizzo libero disponibile
                Lcd_Temp_2
                ENDC

;-----------------------------------------------------------------------
;UP_LCD: Configurazione PIC per il LCD.
;
UP_LCD          bsf     STATUS,RP0      ;Banco 1
                clrf    PORTB           ;PB <0-7> uscite digitali
                movlw   b'00011000'
                movwf   PORTA           ;RA0-RA2 uscite, RA3-RA4 ingressi
                bcf     STATUS,RP0      ;Banco 0
                OFF_COMANDO             ;RS=0
                DISABLE                 ;E=0
                return
;-----------------------------------------------------------------------
```

Intestazione della libreria e routine di configurazione del PIC.

Intestazione della libreria e routine di configurazione del PIC

Abbiamo visto molte delle routine che potremo trovare nella libreria di gestione dell'LCD, ma manca ancora una nozione molto importante. Quando facciamo una libreria in essa cerchiamo di inserire tutti quei processi che sono relativi o che devono essere ripetuti molte volte in diversi programmi.

Nella libreria che comprende le principali routine di utilizzo di un modulo LCD abbiamo raggruppato tutte le azioni tipiche da eseguire su un display di questo tipo, ma c'è un aspetto della configurazione che non abbiamo ancora contemplato. Quando il PIC lavora con un LCD dovrà sempre avere una precisa configurazione che possiamo fissare nella libreria. Definiremo così la porta B come uscita e della porta A, RA2:RA0 come uscite e RA4 e RA3 come ingressi. La routine si completa con la definizione delle variabili e con l'associazione delle istruzioni più comuni a un nome, in modo da non doverle scrivere ogni volta che si utilizzano, ma semplicemente nominarle.

Primo programma con display LCD

Conosciamo già il funzionamento e l'architettura interna del nostro display LCD, quindi ci rimane solamente da sapere come lavorare con esso. Realizzeremo un semplice progetto con cui interagiremo con il modulo per abituarci poco a poco alla sua gestione.

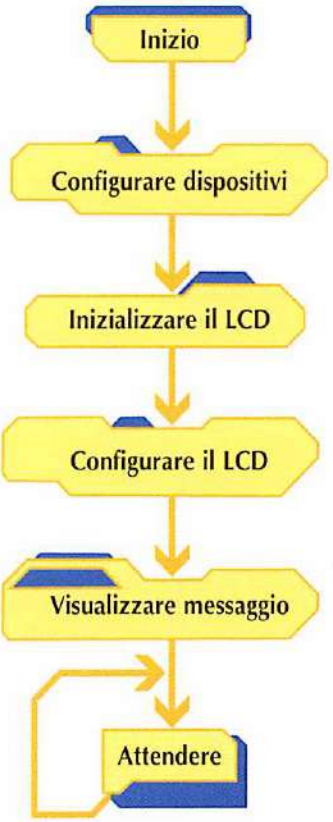
Enunciato

Si vuole visualizzare sul modulo LCD un messaggio, ad esempio: "Ciao". Dobbiamo applicare tutte le conoscenze acquisite per risolvere questo semplice programma e provarlo sul laboratorio.

Nella libreria "lcd_cxx.in", che potete trovare nella cartella LCD nel secondo CD allegato all'opera, esiste una routine che invia impulsi all'ingresso di abilitazione.

Organigramma

Osservando l'organigramma che abbiamo predisposto per risolvere l'applicazione, verificherete che l'unica difficoltà del programma è quella di interagire per la prima volta con il nostro modulo LCD. Provate a pianificare una soluzione al progetto e confrontatela con quella che svilupperemo di seguito.



Organigramma dell'applicazione.

Codice

Apriremo l'editor di testo e mediante dei commenti spiegheremo ciò che dovrà contenere il file e le sue funzionalità. Intesteremo il file definendo il PIC e le librerie con le quali vogliamo lavorare. Solitamente lavoriamo so-

ciao.asm - Blocco note

File Modifica Formato Visualizza ?

```
;       ESERCIZIO: CIAO
;       ---------------
;
;Questo esempio ci vuole introdurre alla gestione del display LCD, per la visualizzazione
;dei diversi messaggi (ad esempio Ciao).

                List    p=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizione dei registri interni

Lcd_var         equ     0x20            ;Variabili (2) delle routine di gestione del LCD

                org     0x00            ;Vector di Reset
                goto    Inizio
                org     0x05            ;Salva vector di interrupt

                include "LCD_Cxx.inc"   ;Include le routine di gestione del LCD
```

Intestazione del programma.

Definizione del blocco delle variabili nella libreria.

lamente con quella che contiene le definizioni dei registri del PIC "P16F870.inc", ma in questo caso abbiamo bisogno anche della libreria che contiene anche tutte le subroutine di gestione del display LCD "lcd-cxx.inc".

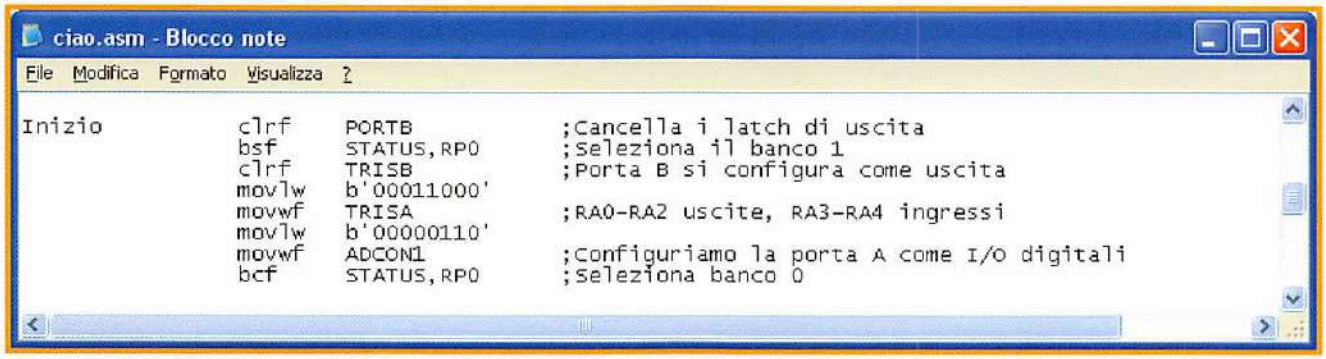
Continueremo l'intestazione con le direttive di intestazione del programma nella memoria ORG e per concludere questa prima parte del programma, definiremo la variabile "Lcd_var", assegnandola a una posizione di memoria. È necessario definire questa variabile ogni volta che lavoriamo con il modulo LCD, in quanto nella libreria di gestione del display è contemplato il suo utilizzo. In questa libreria viene definita una struttura, cioè un blocco di variabili raggruppate sotto lo stesso nome: "Lcd_var". Questo blocco contiene due variabili, "Lcd_Temp_1" e "Lcd_Temp_2", quindi in realtà nel programma che svilupperemo quando definiamo "Lcd_var" stiamo riservando posizioni di memoria per le due variabili prima descritte. È importante mantenere un certo ordine al momento di confezionare un codice. Quindi, tutte le chiamate alle librerie, devono essere raggruppate, così come tutte le definizioni di variabili, commenti, ecc. Questo ordine migliorerà la struttura del programma rendendone più facile il successivo studio e la comprensione.

Configurazione dei dispositivi

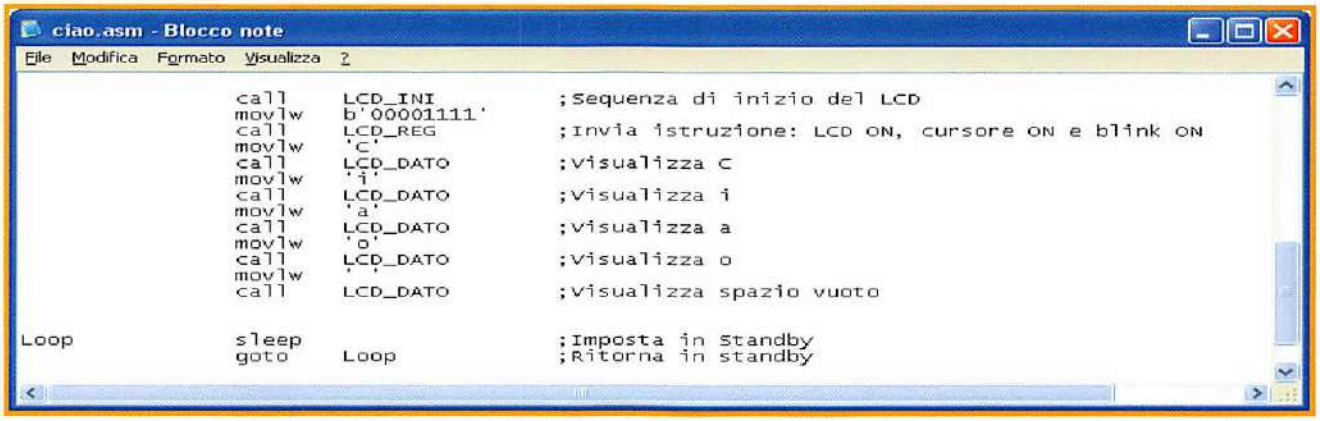
Configureremo i dispositivi del microcontroller che vogliamo utilizzare. In questo caso lavoreremo solamente con le porte di ingresso/uscita. Configureremo la porta B come uscita e la porta A come mista, cioè RA0-RA2 come uscite e RA3-RA4 come ingressi digitali.

Visualizzazione del messaggio

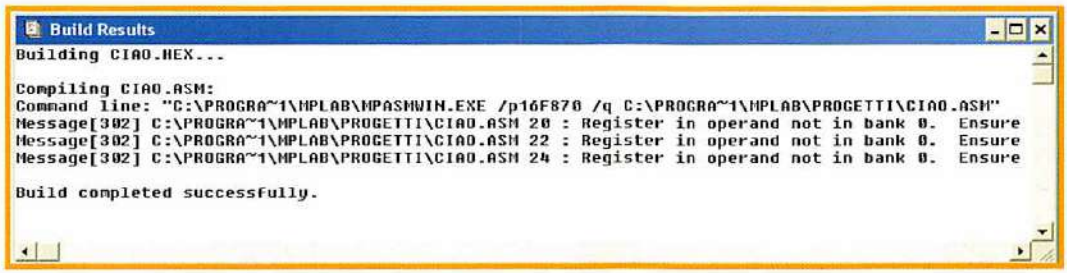
Per risolvere l'applicazione dobbiamo visualizzare il messaggio "Ciao" sul modulo LCD. A questo scopo lo dobbiamo inizializzare dato che la routine che lo inizializza si trova definita nella libreria "lcd_cxx.inc", eseguiremo una



Configurazione dei dispositivi.



Visualizzazione del messaggio.



Risultato della compilazione.

chiamata alla routine mediante l'istruzione call. A questo punto dobbiamo configurare il modulo LCD e per fare questo dobbiamo inviare un comando al display. Come avviene con l'inizializzazione, esiste già una routine che fa questo, la routine "LCD_REG". Prima di chiamare questa routine è necessario caricare il registro di lavoro W con il valore che vogliamo dare al comando, e di seguito tramite la routine, verrà inviato al display. Configureremo il display in modo da attivare LCD ON, il cursore e il lampeggio. A questo punto ci rimane solamente da inviare il messaggio che vogliamo visualizzare. È necessario inviare carattere per carattere, quindi invieremo prima il carattere "C", dopo "i" e così via fino a completare la parola "Ciao". Per inviare un carattere lo dobbiamo caricare sul registro di lavoro e poi chiamare la routine "LCD_DATO".

Fatto questo il programma deve entrare in un ciclo di attesa in modo che il messaggio rimanga visualizzato sul display.

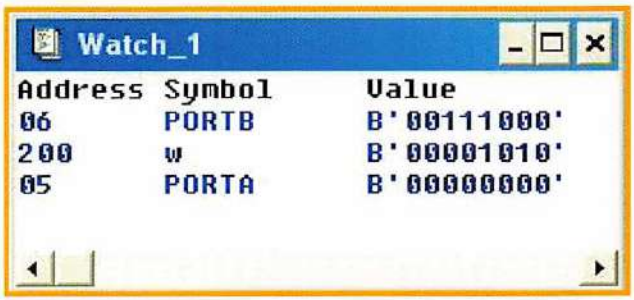
Compilazione

Nel secondo CD allegato all'opera potrete trovare il codice che abbiamo sviluppato con il nome di "Ciao.asm", all'interno della cartella "LCD". Facciamo partire MPLAB per compilare il programma, creiamo un nuovo progetto che chiameremo "Ciao.pjt" e nella finestra di edit allegheremo il nostro codice al progetto.

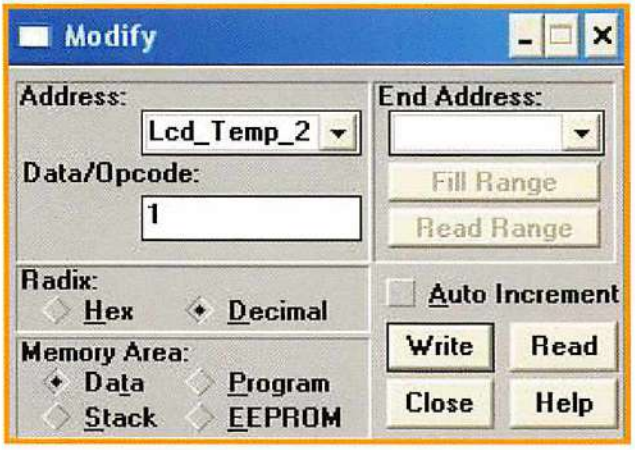
Fatto questo apriamo il nostro programma per visualizzarlo sul display e selezioniamo l'opzione Build All per eseguire la compilazione e l'assemblaggio. Come potete vedere nelle immagini il codice si compila senza errori.

Simulazione

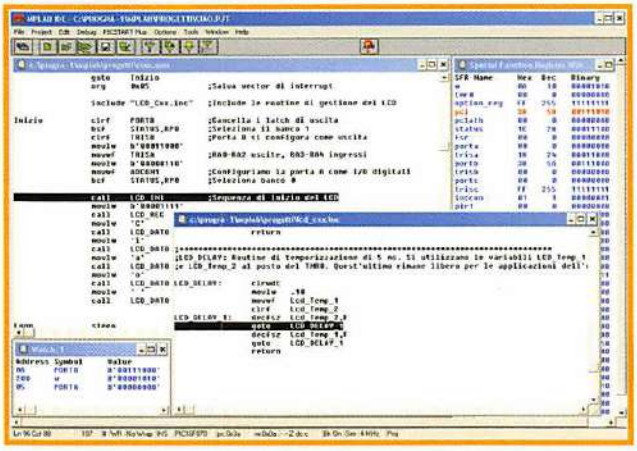
Al momento di simulare ricordate che è necessario copiare nella cartella dei progetti tutti i file associati ai programmi. Per simulare correttamente il nostro programma dobbiamo copiare la libreria del modulo LCD nella cartella dei progetti.



Finestra per visualizzare le porte e W.



Forziamo i valori delle variabili. aprendo la finestra Modify.



Il simulatore entra in un ciclo all'interno della subroutine di ritardo.

Apriamo le finestre tipiche della simulazione.

Nella visualizzazione dei registri più importanti inseriamo le porte B e A in modo binario e il registro di lavoro W. Ricordate che è necessario scegliere "w" in caratteri minuscoli per visualizzarlo correttamente.

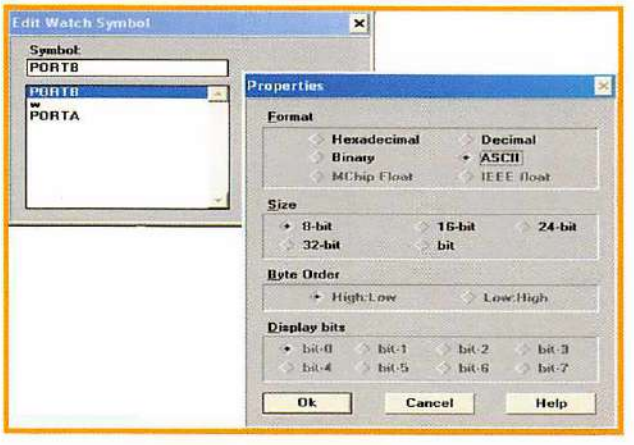
Inizieremo la simulazione e quando arriveremo alla chiamata della subroutine "LCD_INI" il simulatore esegue la routine della libreria. Il simulatore salta alla subroutine avvicinandosi molto a un funzionamento quasi reale del programma, realizzando una simulazione completa dell'applicazione.

Se continuiamo premendo F7, simulando passo a passo verrà eseguita la subroutine, e potremo vedere il simulatore che passerà alle altre subroutine all'interno della libreria, in base allo sviluppo del programma. Saltiamo a "LCD_REG", da questa a "LCD_BUSY", a "LCD_E" e infine a "LCD_DELAY". In quest'ultima subroutine il simulatore si ferma in un ciclo attendendo che la variabile del temporizzatore "Lcd_Temp_2" raggiunga lo zero.

Dobbiamo forzare il valore di questa variabile e per fare questo apriremo la finestra Modify e forzeremo la variabile a 1 in decimale, come possiamo vedere nella figura. Dobbiamo fare lo stesso con la variabile "Lcd_Temp_1" per poter uscire da questa subroutine. Fatto questo il programma continua a eseguire la routine "LCD_INI", le cui istruzioni si ripetono, dobbiamo quindi mantenere aperta la finestra Modify per poter uscire dalla routine di temporizzazione.

Quando termineremo di eseguire la routine "LCD_INI" la simulazione ritorna al programma principale e continua dalla linea successiva a quella della chiamata alla subroutine. Si configura il display LCD e si inizia a inviare i caratteri al display. Per vedere se i caratteri sono inviati correttamente cambiate il modo di visualizzazione della porta B di uscita. Cliccando con il pulsante sinistro del mouse sull'angolo superiore sinistro della finestra ed editandola, sarà possibile cambiare le proprietà di visualizzazione del registro che abbiamo selezionato. Continuate l'esecuzione della simulazione e verificate che il programma risponda in modo soddisfacente.

Lavorare con un display LCD è facile se si hanno ben chiari i concetti. Questo programma è un chiaro esempio di come sia semplice visualizzare un messaggio sul display. Nei prossimi fascicoli complicheremo un po' gli esercizi per poter ottenere il massimo rendimento dal modulo LCD.



Cambiate la visualizzazione dell'uscita in ASCII.

Esercizio: visualizzazione di messaggi, il programma

Continueremo a esercitarci con il display LCD mediante un nuovo esercizio. Aumenteremo progressivamente la difficoltà delle applicazioni, acquisendo in questo modo maggiori conoscenze e dimestichezza, con questo dispositivo e con il PC in generale.

Enunciato

Si tratta di sviluppare un'applicazione che permetta di visualizzare sul modulo LCD diversi messaggi. Sul display verranno presentati i messaggi "Ciao" e "Addio" alternativamente, con un intervallo di tempo di 2 secondi.

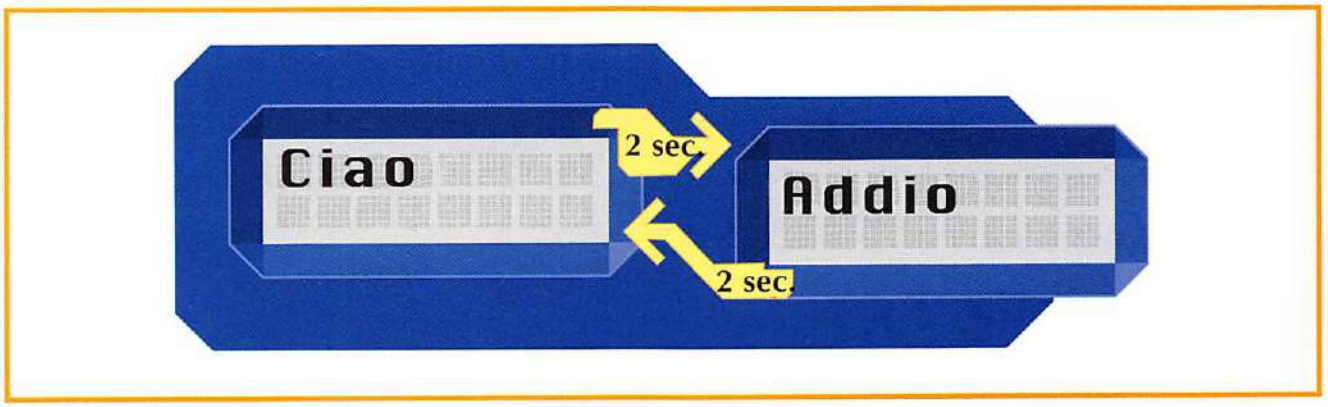
Dall'enunciato possiamo dedurre i dispositivi del microcontroller che utilizzeremo. Avremo bisogno delle porte A e B per lavorare con LCD e del Timer0 per stabilire la temporizzazione.

Organigramma

Prima di iniziare a progettare il codice, lo dovremo plasmare impostandolo attraverso l'organigramma. La prima cosa da prendere in considerazione è la necessità di configurare i dispositivi del PIC.

Fatto questo, inizializzeremo il display LCD e lo configureremo (abilitazione, cursore e lampeggio) preparando il modulo per lavorare con esso. Visualizzeremo il primo messaggio ed eseguiremo un chiamata alla subroutine di temporizzazione caricandola con il valore che provoca il ritardo desiderato. Trascorso questo tempo, visualizzeremo il secondo messaggio e temporizzeremo nuovamente, prima di ripete-

Organigramma dell'applicazione.



Ogni due secondi cambierà il messaggio sul display LED.

addio.asm - Blocco note

File Modifica Formato Visualizza ?

```
;       ESERCIZIO: LCD con diversi messaggi
;       -----------------------------------
;

                List    p=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizione dei registri interni

Lcd_var         equ     0x20            ;Variabili (2) delle routine di gestione del LCD
Delay_Cont      equ     0x21            ;Variabile per la temporizzazione
Temporale_1     equ     0x22            ;Variabile Temporale
Temporale_2     equ     0x23            ;Variabile Temporale

                org     0x00            ;Vector di Reset
                goto    Inizio
                org     0x05            ;Salva vector di interrupt

                include "lcd_cxx.inc"   ;Include la routine di gestione del LCD

;*****************************************************************************
```

Intestazione del programma.

re il ciclo dal punto in cui visualizzeremo il primo messaggio.

Codice
Intestazione del programma

Come d'abitudine, la prima cosa da fare quando apriamo l'editor di testo per creare un nuovo programma, è scrivere all'interno di un commento la spiegazione di ciò che vogliamo fare in un programma. Definiamo il microcontroller che utilizzeremo e la libreria di definizione dei registri. Quando lavoriamo con il display LCD dobbiamo inserire anche la libreria che contiene le subroutine di funzionamento del display.

Terminiamo la prima fase del programma con le direttive ORG per organizzare il codice nella memoria e definendo le variabili che interverranno all'interno del programma.

Sono necessarie le due variabili per lavorare con la libreria dell' LCD, saranno definite come "Lcd_var", dato che nella libreria a questo nome corrisponde un blocco di due variabili. Definiremo la variabile che interverrà nella temporizzazione "Delay_Cont" e in ultimo, verranno definite le due variabili temporali per lo sviluppo del programma "Temporale_1" e "Temporale_2".

Nel caso non si conoscano le variabili necessarie per lo sviluppo del programma, verranno definite man mano che si rendono necessarie.

addio.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;Delay_var: Questa routine di carattere generale realizza una temporizzazione variabile
;fra 50 ms e 12.8". Si utilizza un prescaler da 256 e si carica il TMR0 con 195.
;La velocità di lavoro è di 4 MHz quindi il TMR0 si incrementa ogni µs. In questo
;modo il TMR0 deve contare 195 eventi che, con un prescaler da 256 generano un
;intervallo totale di  50000 µs cioè 50 ms (195 * 256). Il valore 195 deve essere espresso
;in Hex. (c3) e dato che il TMR0 è ascendente dovremo caricare il suo complemento (3C hex.)
;Questo intervallo di 50 ms è ripetuto tante volte quante indicate dalla variabile "Delay_cont",
;è per questo che il ritardo minimo è 50 ms ("Delay_cont=1) e il massimo è 12.8"
;(Delay_cont=255).

Delay_var:      bcf     INTCON,T0IF     ;Resetta il flag di overflow
                movlw   0x3c            ;Complemento hex. di 195
                movwf   TMR0            ;Carica il TMR0
Intervallo      clrwdt                  ;Aggiorna il WDT
                btfss   INTCON,T0IF     ;Overflow del TMR0?
                goto    Intervallo      ;Non ancora
                decfsz  Delay_Cont,F    ;Decrementa il contatore di intervalli
                goto    Delay_var       ;Ripete l'intervallo di 50 ms
                return

;*****************************************************************************
```

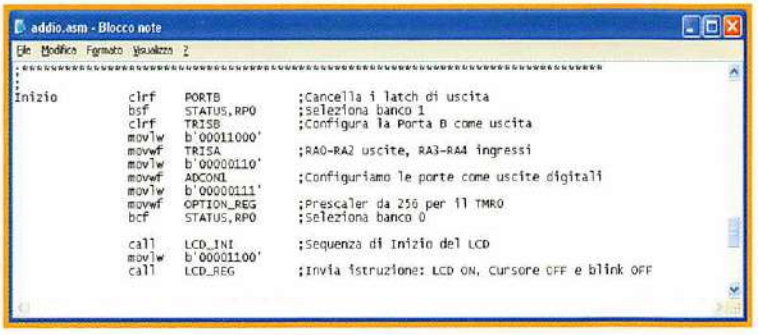
Routine di temporizzazione.

addio.asm - Blocco note

File Modifica Formato Visualizza ?

```
;*******************************************************************************************************
;In base al valore contenuto nel registro w, si ottiene il carattere da visualizzare

Tab_Messaggi    movwf   PCL             ;Calcola lo spostamento sulla tabella

Mess_0          equ     $               ;Mess_0 punta al primo carattere del messaggio 0
                retlw   'C'
                retlw   'i'
                retlw   'a'
                retlw   'o'
                retlw   0x00            ;ultimo carattere del messaggio 0

Mess_1          equ     $               ;Mess_1 punta al primo carattere del messaggio 1
                retlw   'A'
                retlw   'd'
                retlw   'd'
                retlw   'i'
                retlw   'o'
                retlw   0x00            ;ultimo carattere del messaggio 1

;*******************************************************************************************************
```

Tabella dei messaggi.



Configurazione dei dispositivi.

creeremo una tabella dove contempleremo tutti i caratteri. Quando richiameremo la tabella , il PCL o contatore di programma si caricherà con il valore del registro di lavoro e in funzione di quest'ultimo restituirà il carattere da visualizzare. Quando il valore fornito è 0 significa che il messaggio è stato presentato nella sua completezza. Nell'immagine della figura in alto potete vedere il formato di questa tabella.

Routine di temporizzazione

Sappiamo che abbiamo bisogno di una routine di temporizzazione. Dato che negli esercizi precedenti abbiamo già utilizzato routine di ritardo o temporizzazioni, copieremo la routine nel nostro codice.

Nella figura alla pagina precedente possiamo vedere una routine di temporizzazione tipica degli esercizi che abbiamo eseguito fino a questo momento.

Tabella dei messaggi

Dato che presenteremo due messaggi diversi ('Ciao' e 'Addio') e che questi devono essere inviati carattere per carattere al modulo LCD,

Configurazione dei dispositivi

Senza aver iniziato a programmare abbiamo già dei blocchi di codice nel nostro file. Inizieremo il programma principale che risolve l'applicazione interagendo con i blocchi e le librerie incluse, configurando i dispositivi del PIC che utilizzeremo.

Definiamo la porta B come porta di uscita e nella porta A imposteremo RA0-RA2 come uscite e RA3-RA4 come ingressi. Configurando il registro ADCON1 definiremo la porta A di tipo digitale e configurando il registro OPTION_REG definiremo il predivisore per il temporizzatore Timer0. Infine inizializzeremo il display e lo configureremo in modo che sia abilitato, con cursore e lampeggio definiti.

```
ADIOS - Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
;*****************************************************************************
;Mensaje: Esta rutina visualiza en el LCD el mensaje cuyo inicio está indicado en
;el acumulador. El fin de un mensaje se determina mediante el código 0x00

Mensaje         movwf   Temporal_1      ;salva posición de la tabla
Mensaje_1       movf    Temporal_1,W    ;Recupera posición de la tabla
                call    Tabla_Mensajes  ;Busca caracter de salida
                movwf   Temporal_2      ;Guarda el caracter
                movf    Temporal_2,F
                btfss   STATUS,Z        ;Mira si es el último
                goto    No_es_ultimo
                return
No_es_ultimo    call    LCD_DATO        ;visualiza en el LCD
                incf    Temporal_1,F    ;Siguiente caracter
                goto    Mensaje_1
;*****************************************************************************
```

Subroutine per la configurazione del messaggio.

```
addio.asm - Blocco note
File  Modifica  Formato  Visualizza  ?

Loop            movlw   b'00000001'
                call    LCD_REG         ;Cancella LCD e Home (colloca il cursore nella 1ª posizione)
                movlw   Mess_0
                call    Messaggio       ;visualizza il Messaggio 0
                movlw   .20
                movwf   Delay_Cont
                call    Delay_var       ;Temporizza 2 secondi
                movlw   b'00000001'
                call    LCD_REG         ;Cancella LCD y Home (colloca il cursore nella 1ª posizione)
                movlw   Mess_1
                call    Messaggio       ;visualizza il Messaggio 1
                movlw   .20
                movwf   Delay_Cont
                call    Delay_var       ;Temporizza 2 secondi
                goto    Loop

                end                     ;Fine del programma sorgente
```

Programma principale.

Programma principale

Questa è la parte del codice in cui risolveremo realmente l'applicazione. La prima cosa da fare è inviare un comando al modulo LCD per posizionare il cursore nella posizione iniziale del display. Ora dobbiamo decidere come visualizzare i messaggi.

Per vedere i messaggi dobbiamo prima sapere quale dobbiamo presentare sul display e dopo muoverci sulla tabella presentando carattere per carattere fino ad arrivare all'ultimo. Per eseguire questo creeremo una routine che visualizzi sull'LCD il messaggio il cui inizio è indicato nell'accumulatore e che identifichi la fine di un messaggio determinato mediante il codice 0x00.

Questa routine ha bisogno di due variabili, una per salvare l'indirizzo della tabella da cui abbiamo ottenuto il carattere e l'altra per salvare il carattere stesso. Queste variabili sono Temporale_1 e Temporale_2.

La prima cosa che fa la routine è salvare l'indirizzo della tabella nella prima variabile temporale. Dopo inizia un ciclo in cui si scrive nella variabile l'indirizzo del carattere da visualizzare, si cerca nella tabella e si scrive nella seconda variabile. Nel ciclo si verifica anche che questo carattere non sia l'ultimo, e nel caso in cui non lo sia, lo si visualizza chiamando la routine LCD_DATO e si incrementa la posizione della tabella per andare al carattere successivo. Se è l'ultimo si esce dal ciclo e dalla routine.

Abbiamo risolto in una subroutine come presentare i messaggi, dato che il programma principale richiamerà questa subroutine, caricherà la variabile di temporizzazione con il valore appropriato per il ritardo desiderato e richiamerà la subroutine di ritardo. Fatto questo, ripeteremo gli stessi passaggi per il messaggio successivo: posizionamento del cursore, routine dei messaggi e routine di ritardo.

Tutto questo sarà all'interno di un ciclo per ripetere ciclicamente questi passaggi.

Compilazione

Abbiamo terminato quella che si definisce la prima stesura del programma. È normale, durante la compilazione, trovare qualche errore che non ci costerà molta fatica risolvere. L'esempio che abbiamo presentato si trova sul secondo CD allegato all'opera, all'interno della cartella "LCD" con il nome "Addio.asm".

Quando apriamo MPLAB ed eseguiamo tutti i passaggi necessari per compilare il programma vedremo che la compilazione non genererà errori, ma darà i messaggi tipici di posizionamento dei registri nella memoria.

```
Build Results
Building ADIOS.HEX...

Compiling ADIOS.ASM:
Command line: "C:\ARCHIV~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\ARCHIV~1\MPLAB\PROYEC~1\ADIOS.ASM"
Message[302] C:\ARCHIV~1\MPLAB\PROYEC~1\ADIOS.ASM 84 : Register in operand not in bank 0.  Ensure
Message[302] C:\ARCHIV~1\MPLAB\PROYEC~1\ADIOS.ASM 86 : Register in operand not in bank 0.  Ensure
Message[302] C:\ARCHIV~1\MPLAB\PROYEC~1\ADIOS.ASM 88 : Register in operand not in bank 0.  Ensure

Build completed successfully.
```

Risultato dell'applicazione.

Esercizio: visualizzazione di messaggi, il programma (II)

Abbiamo ancora in sospeso la simulazione dell'esercizio di visualizzazione di diversi messaggi "Addio.asm". Anche se non abbiamo ancora il display LCD per provare gli esercizi possiamo simularli e analizzarne la risposta, verificando che il programma risponda all'enunciato iniziale.

Simulazione del programma "Addio.asm"

Abbiamo visto come programmare il PIC per fare in modo che l'LCD visualizzi diversi messaggi sviluppando il codice "Addio.asm" e compilandolo alla ricerca di errori. Ora dobbiamo simularne il funzionamento del programma mediante MPLAB.

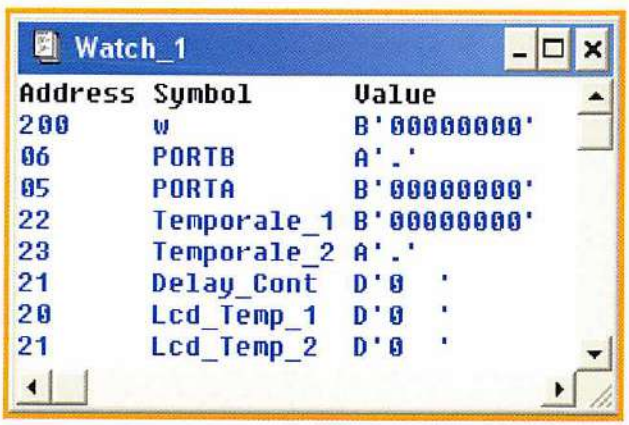
Apriamo il progetto "Addio.pjt" e il codice associato per visualizzarlo sul display. Apriremo anche la finestra dei registri speciali e una finestra per vedere i registri più importanti dell'applicazione. Presenteremo la porta B e la variabile Temporale_2 in ASCII, il registro di lavoro, la porta A e la variabile Temporale_1 in binario e le variabili dei temporizzatori in decimale (Delay_Cont, LCD_Temp_1 e LCD_Temp_2). Nella figura potete vedere la finestra che abbiamo descritto.

Inizieremo la simulazione passo a passo premendo F7. Come era già successo quando abbiamo simulato il primo esercizio di gestione dell'LCD, quando nel programma principale si esegue la chiamata alla subroutine LCD_INI, si salta alla libreria "lcd_cxx.inc" e si eseguono le sue istruzioni. Per uscire dalla subroutine LCD_DELAY terremo premuto F7 fino a quando le due variabili temporali saranno 0 o apriremo la finestra Modify e cambieremo il valore delle due variabili, come abbiamo fatto nell'esercizio precedente. Ricordate che dobbiamo ripetere questa operazione tre volte.

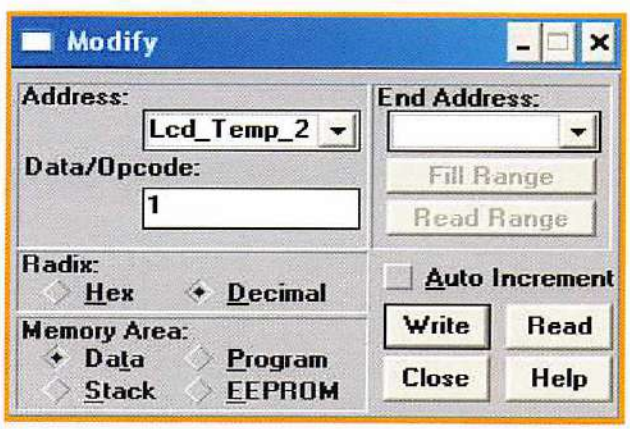
Dopo che il simulatore è ritornato nel programma principale ed è entrato nell'esecuzione del ciclo (Loop) potremo osservare che la porta B sta spedendo correttamente il primo messaggio (ciao) carattere per carattere. Fatto questo, il programma entra nella routine di temporizzazione del TMR0 e rimane in attesa che il flag del temporizzatore si attivi. Forzeremo questo valore a 1 mediante la finestra Modify, come potete vedere dall'immagine in basso e usciremo dalla routine.

Dopo questa attesa simulata il programma continuerà l'esecuzione visualizzando ora il secondo messaggio (Addio). Terminata questa visualizzazione si entrerà nuovamente nella routine di temporizzazione e dopo esserne usciti, il ciclo si ripeterà dall'inizio.

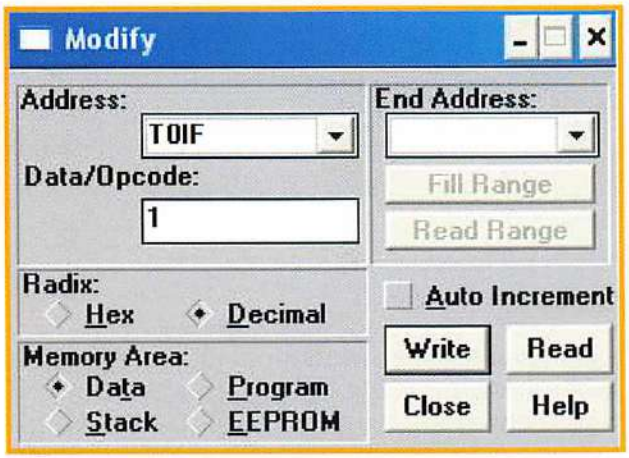
Abbiamo verificato che il programma fun-



Finestra per visualizzare i registri più importanti.



Forziamo il valore delle variabili di temporizzazione.



Forziamo il valore di TOIF per uscire dalla routine.

ziona come ci aspettavamo, quindi dobbiamo solamente verificarne il funzionamento sul laboratorio, eseguendo il corrispondente montaggio hardware.

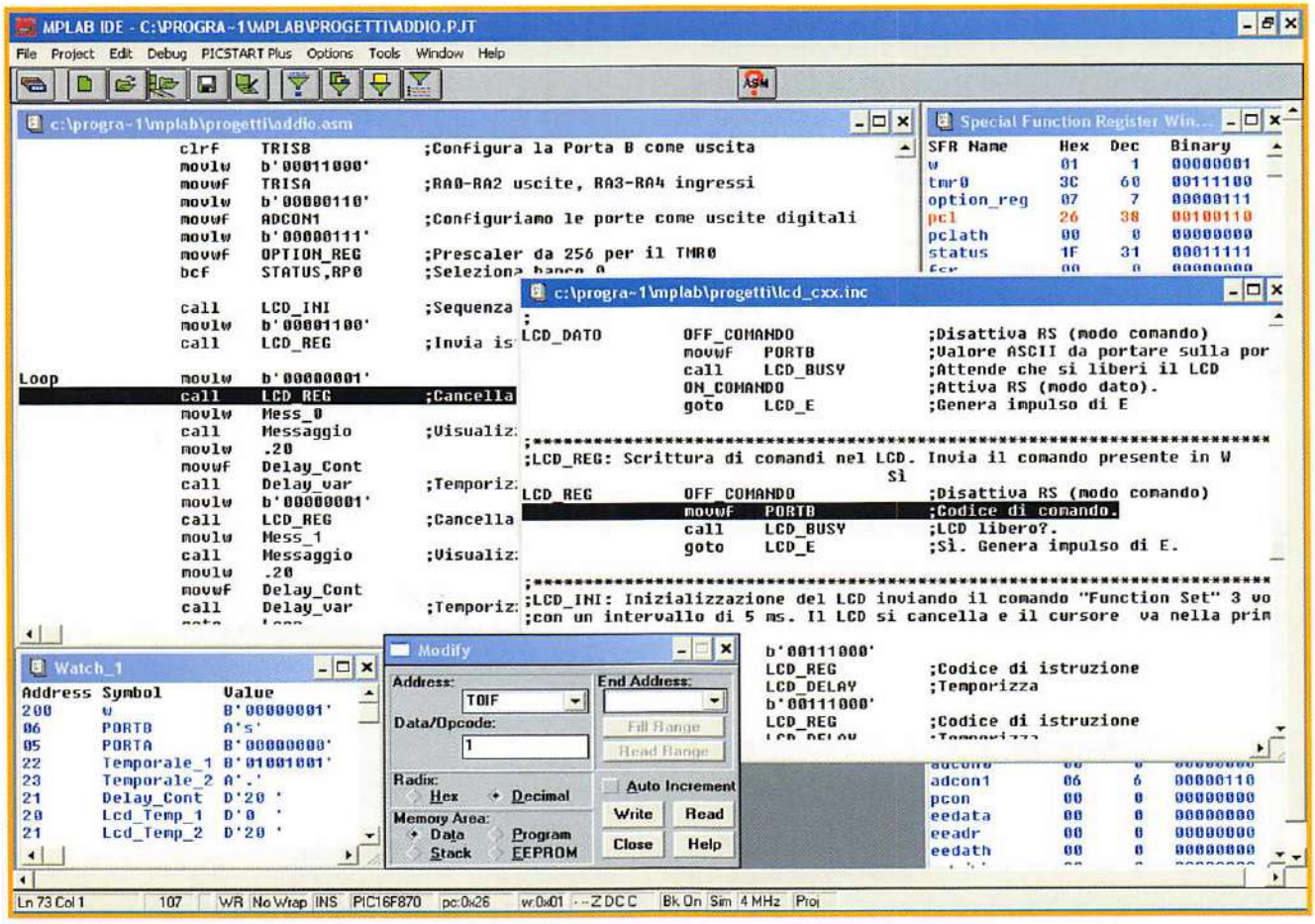
Modifiche al programma

Prima di affrontare nuovi argomenti di programmazione, lavoriamo sul programma che abbiamo realizzato presentando diverse alternative.

Nuovo messaggio

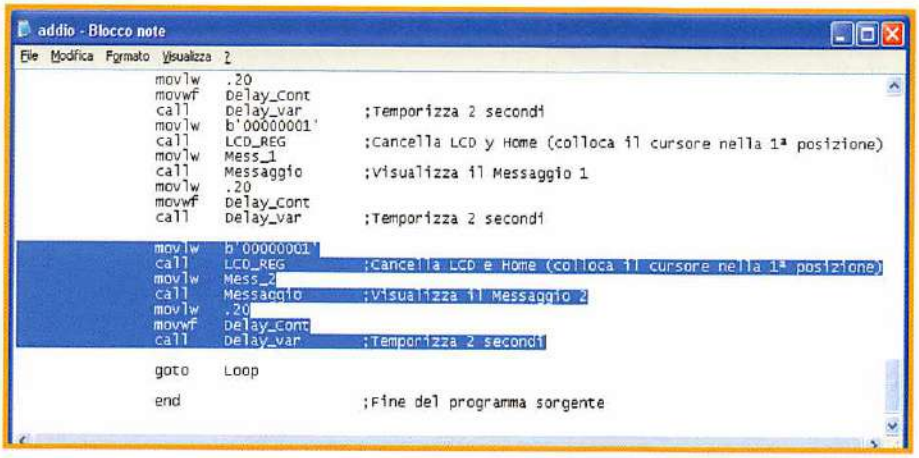
Nel codice "Addio.asm" si contempla solamente la possibilità di visualizzare due messaggi, ma che modifiche dovremmo eseguire per poter visualizzare più messaggi?

Per visualizzare un messaggio nuovo la prima cosa da fare è inserire questo nella tabella dei messaggi. Mantenendo la struttura dei precedenti, inseriamo un nuovomessaggio, ad esempio con il testo "Allarme". L'immagine nella figura della pagina successiva vi permette di verificare come viene inserito il codice corrispondente al nuovo messaggio nella



MPLAB durante la simulazione.

Modifichiamo la tabella inserendo un nuovo messaggio.



Modifichiamo il programma per far visualizzare il nuovo messaggio.

tabella dei messaggi. Possiamo vedere il nuovo codice evidenziato in azzurro.

Il nuovo messaggio è ora all'interno della tabella ma non abbiamo ancora modificato il programma che visualizza i messaggi. Se andiamo al ciclo di visualizzazione, a partire dall'etichetta Loop possiamo verificare che ogni messaggio che viene visualizzato ha delle istruzioni associate. Queste iniziano con l'istruzione tramite la quale carichiamo sul registro di lavoro il valore '00000001' per chiamare la routine "LCD_REG" in modo da cancellare il display e collocare il cursore nella prima posizione. Il blocco di istruzioni associate a ogni messaggio termina quando chiamiamo la routine di temporizzazione "call Delay_var" per temporizzare i 2 secondi.

Se copiamo questo blocco di istruzioni e lo adattiamo per il nuovo messaggio che vogliamo visualizzare, otterremo la visualizzazione sul display LCD dei nostri tre messaggi.

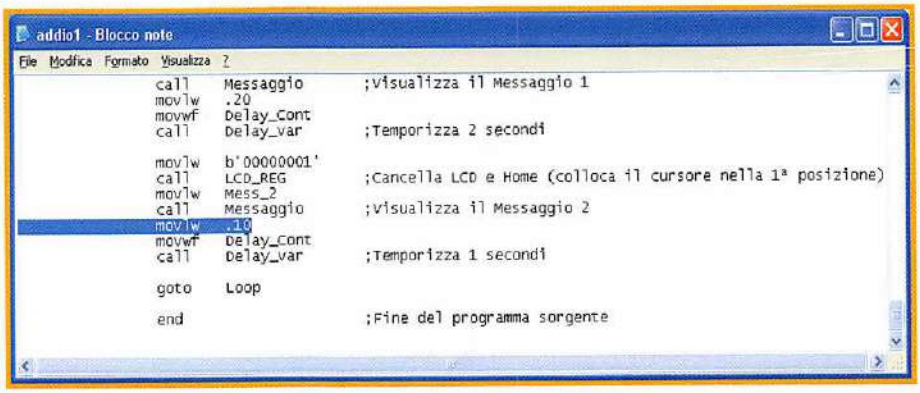
Notate dove viene inserito il blocco di istruzioni (dopo quelle che corrispondono al secondo messaggio) e come è stato adattato per il nostro messaggio (movlw Mess_2).

Fatto questo, dobbiamo salvare le modifiche che abbiamo realizzato nel programma memorizzandolo con un nome diverso, ad esempio "Addio1.asm". Il passo successivo è compilare il codice per verificare che le modifiche non diano errori.

Risultato della compilazione.

Cambi nella temporizzazione

Immaginate ora di volere che la temporizzazione dei tre messaggi non sia uguale e che il nuovo messaggio si veda la metà del tempo degli altri. Utilizzando la stessa routine potremo temporizzare tra 50 millisecondi e 12,8 secondi. Dovremo semplicemente cambiare il valore della variabile Delay_cont e inserire in essa il fattore da far moltiplicare con i 50 ms di ritardo minimo che genera la subroutine. Se inseriamo 10 invece di 20, che è il valore degli altri due messaggi, otterremo che il nuovo messaggio si visualizzi la metà del tempo degli altri due. Nell'immagine della figura possiamo vedere la modifica sull'istruzione che carica il registro di lavoro con il valore che successivamente verrà dato alla variabile.



Cambiamo la temporizzazione della visualizzazione.

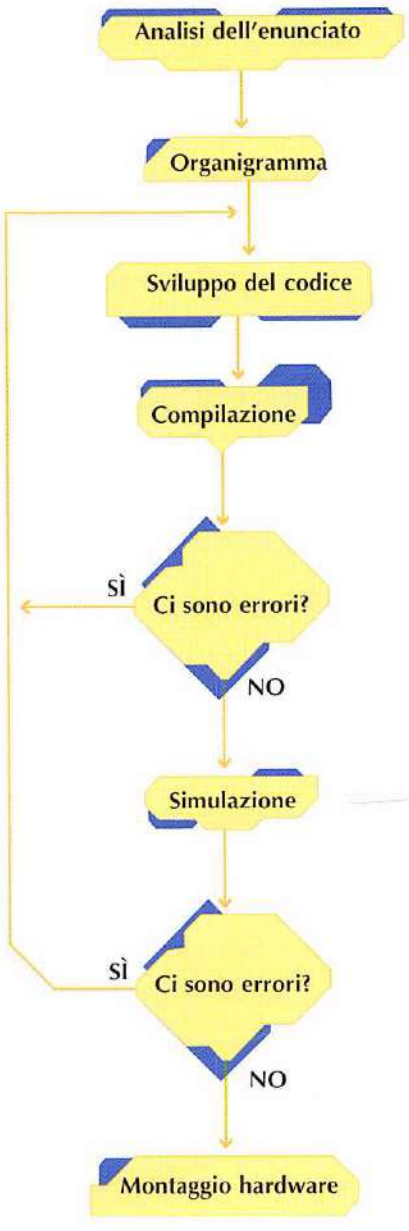
Conclusioni

Qualsiasi variazione nel programma richiede una nuova compilazione e simulazione. Dobbiamo prendere come regola quella di compilare e simulare il programma modificato prima di procedere alla scrittura sul PIC.

Avete potuto verificare con quale semplicità si possono fare variazioni in un programma per adattarlo alle nuove necessità. Questo è un grande vantaggio al momento di sviluppare nuove applicazioni, dato che normalmente potremo utilizzare programmi precedentemente risolti in modo che servano come base per lo sviluppo di quelli nuovi.

Nel programma di visualizzazione con display LCD "Addio.asm" abbiamo predisposto due semplici modifiche tra tutte quelle che si sarebbero potute fare, ma le possibilità dei programmi dipendono dall'immaginazione che ha il programmatore. Ad esempio, sarà possibile predisporre un programma che visualizzi un messaggio in funzione dell'ingresso selezionato, inoltre è possibile incatenare messaggi in funzione degli ingressi, ecc.

Nei programmi successivi complicheremo un po' di più le applicazioni di gestione dell'LCD in modo da affinare le conoscenze e il metodo di lavoro per utilizzare questo dispositivo esterno.



Passaggi raccomandati per sviluppare un'applicazione.

Esercizio: generatore di messaggi, il programma

Come abbiamo già anticipato, lavoreremo nuovamente con il display LCD, ma con un esercizio più complesso. Mediante questo esercizio potremo scrivere nella memoria del PIC (EEPROM) un messaggio qualsiasi e successivamente visualizzarlo sul display.

Enunciato

Mediante questo esercizio si vuole realizzare un generatore di messaggi per il display LCD.

Con RA4 a "1" il sistema è in modo programmazione. In questo modo il messaggio si scrive nella memoria EEPROM dei dati. Con RA4 a "0" entriamo nel modo riproduzione. Il messaggio scritto nella EEPROM si visualizza sul display.

Quando l'interruttore RA3 è a livello "1" in corrispondenza del cursore appariranno in sequenza i diversi caratteri disponibili. Portandolo a "0" si seleziona quello presente e si scrive sulla EEPROM. Tornando nuovamente a "1" si seleziona il carattere successivo.

Possiamo scegliere il messaggio che dobbiamo scrivere nella memoria selezionando uno a uno i caratteri, oltre a vedere il messaggio memorizzato.

Organigramma

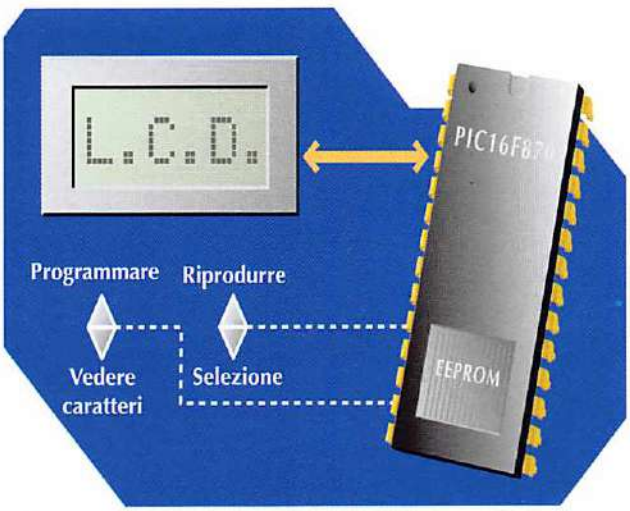
Con le informazioni di cui disponiamo possiamo dedurre quali sono i dispositivi del PIC con cui dovremo lavorare. Utilizzeremo le porte di ingresso/uscita, la memoria EEPROM per scrivere i caratteri del messaggio, e il temporizzatore TMR0 per ottenere i ritardi per selezionare i caratteri del messaggio e per riprodurre quest'ultimo.

Nell'organigramma pianifichiamo una soluzione generale all'applicazione. Questo ci servirà da guida durante lo sviluppo del codice e contiene già l'idea di come vogliamo risolvere l'applicazione.

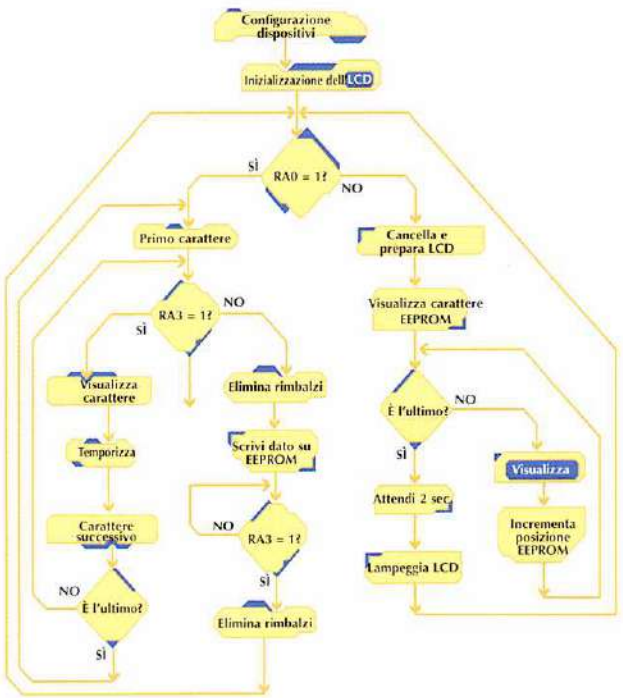
Routine da utilizzare

Quando si affronta un'applicazione complessa che richiede molta programmazione o l'utilizzo di alcuni dispositivi, la possiamo risolvere in due modi: sviluppando tutto il codice o sfruttando routine precedentemente sviluppate per altri programmi che assembleremo nel nostro codice. Ovviamente il secondo modo è più semplice, inoltre utilizzando queste routine verificheremo che non abbiano errori.

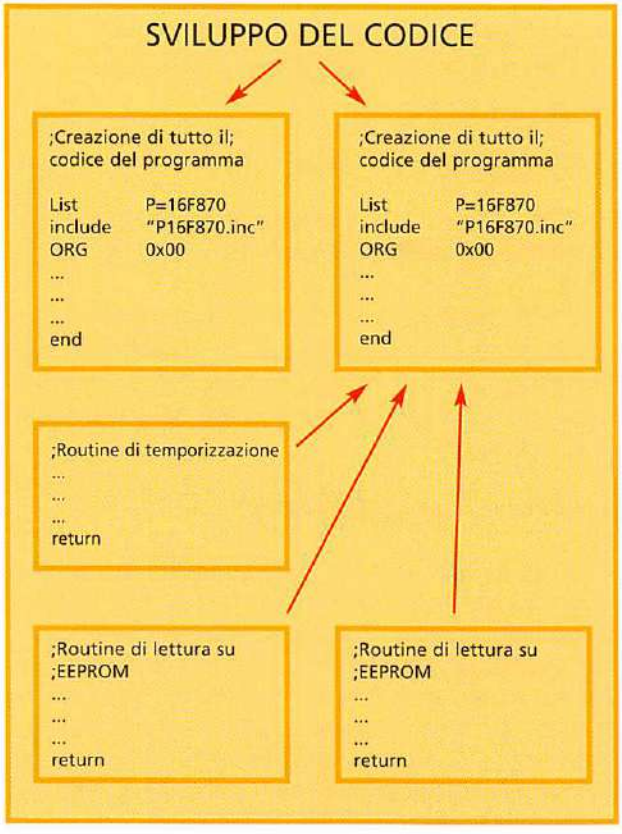
Per l'esercizio pianificato dobbiamo lavorare con la memoria EEPROM e con il temporizzato-



Schema dell'applicazione.



Organigramma dell'applicazione.



Utilizziamo routine sviluppate per altri programmi

re TMR0, per eliminare i rimbalzi e per temporizzare le visualizzazioni. Quando abbiamo lavorato con la memoria EEPROM abbiamo creato una routine per scrivere in essa e un'altra per leggere i dati. Queste routine ci saranno utili per il programma di cui ci stiamo occupando adesso. Inoltre abbiamo a disposizione anche delle routine di temporizzazione, sia per l'eliminazione dei rimbalzi che per temporizzare un tempo variabile.

Routine di lavoro della memoria EEPROM

Negli esercizi in cui abbiamo lavorato con la memoria EEPROM abbiamo già visto che per scrivere nella memoria dobbiamo seguire una procedura, una sequenza di istruzioni che è sempre la stessa. Nell'immagine della figura possiamo vedere la routine di scrittura nella memoria EEPROM che integreremo nel nostro codice.

Per leggere un dato dalla EEPROM avevamo predisposto una routine un po' più semplice della precedente, in cui veniva scelto il banco dove si trovava il registro EECON1, selezionavamo la EEPROM dei dati e ne per-

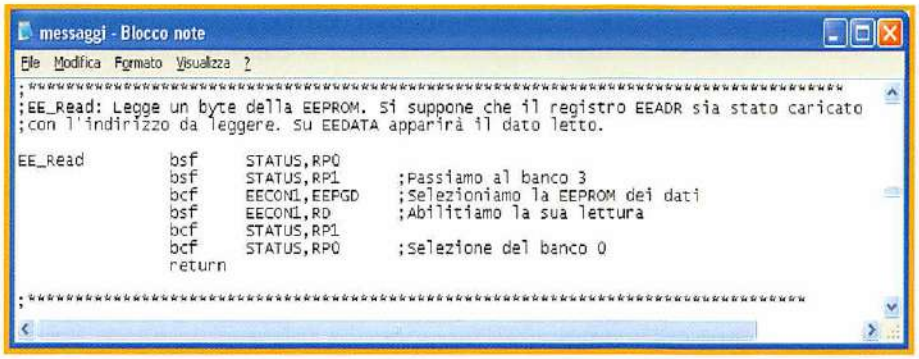
messaggi - Blocco note

File Modifica Formato Visualizza ?

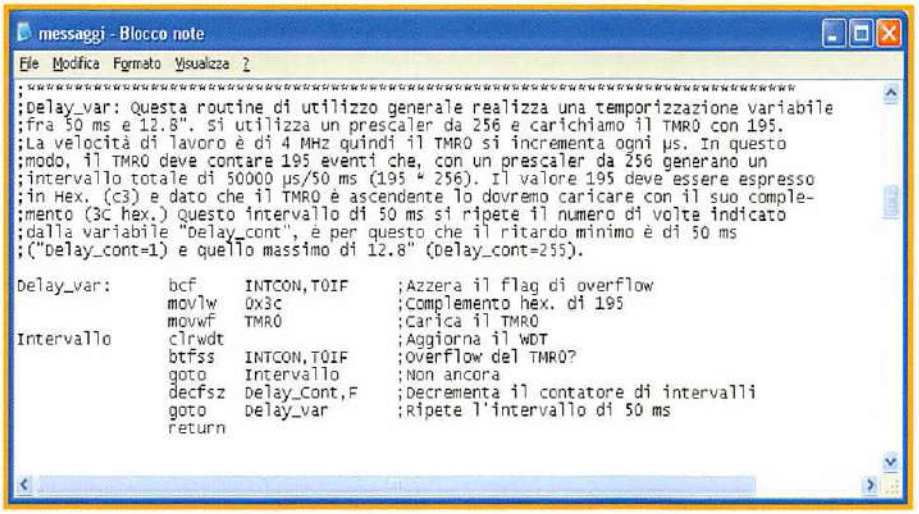
```
;*************************************************************************************
;EE_Write: Scrive un byte nella EEPROM dei dati. L'indirizzo sarà contenuto in EEADR e
;si suppone che il dato sia stato precedentemente caricato su EEDATA

EE_Write        bsf     STATUS,RP1
                bsf     STATUS,RP0      ;Passiamo al banco 3
                bcf     EECON1,EEPGD    ;Selezioniamo la EEPROM dei dati
                bsf     EECON1,WREN     ;Abilitiamo la sua scrittura
                movlw   0x55
                movwf   EECON2
                movlw   0xaa
                movwf   EECON2          ;Sequenza obbligatoria
                bsf     EECON1,WR       ;Iniziamo la scrittura
                bcf     STATUS,RP0
                bcf     STATUS,RP1      ;Torniamo al banco 0
ATTENDI         btfss   PIR2,EEIF       ;Attendiamo che termini la scrittura
                goto    ATTENDI
                bcf     PIR2,EEIF       ;Resettiamo il flag della EEPROM
                return

;*************************************************************************************
```

Routine di scrittura nella memoria EEPROM.

mettevamo la lettura ritornando poi al banco di memoria 0. Questa routine è riportata nella figura a fianco e anch'essa verrà integrata nel nostro codice.



Routine di lettura nella EEPROM.



Routine di temporizzazione a tempo variabile.

Routine di temporizzazione per eliminare i rimbalzi.

Routine di temporizzazione con ritardo variabile

Abbiamo bisogno di una routine di temporizzazione per visualizzare il messaggio scritto e i caratteri da scegliere per un determinato tempo. A questo scopo possiamo utilizzare la routine di temporizzazione variabile tra 50 ms e 12,8 s che abbiamo già utilizzato in diversi esercizi. Nella variabile "Delay_Cont" introduciamo il fattore con il quale vogliamo moltiplicare l'unità minima di tempo della routine (50 ms).

Ogni volta che vogliamo inserire un ritardo o temporizzare un'operazione dovremo solamente richiamare questa routine, caricando la variabile "Delay_Cont" con un valore che sarà in funzione del tempo di ritardo desiderato.

Routine per eliminare i "rimbalzi"

Negli esercizi in cui il valore dell'ingresso determina l'operazione da eseguire è necessario verificare che il valore inserito sull'ingresso sia stabile, prima di procedere all'esecuzione di un'azione. A questo scopo si inserisce un piccolo ritardo dopo la lettura dell'ingresso, assicurando così che, una volta trascorso il tempo di ritardo, l'ingresso avrà un valore stabile e se si sarà verificata una variazione del suo stato, questa verrà accettata. In altre parole, in questo modo non si tiene conto di qualsiasi cambiamento del valore dell'ingresso (rimbal-

zo) che si genera dal momento in cui è stata rilevata la prima variazione di valore fino a quando non saranno trascorsi i 20 ms. Anche questa routine è stata utilizzata negli esercizi precedenti, e la inseriremo nel nostro codice per risolvere la presente applicazione.

Codice

A questo punto possiamo dire di aver fatto dei passi avanti per il codice, dato che molte difficoltà sono state risolte, ma dobbiamo ancora definire lo "scheletro" del codice che contiene le subroutine e che risolve l'applicazione.

Iniziamo a svilupparlo inserendo l'enunciato del problema nell'intestazione del codice per mezzo dei commenti. In questa parte, inoltre, definiremo il PIC, le librerie con cui lavoreremo, e organizzeremo il codice nella memoria mediante le direttive ORG. Dato che conosciamo già le routine con cui lavoreremo, possiamo già definire alcune variabili: Lcd_var e Delay_Cont.

Nell'immagine della figura a fianco possiamo vedere l'aspetto dell'intestazione del programma.

Fatto questo possiamo inserire le subroutine commentate in precedenza.

Configurazione dei dispositivi

Il codice deve iniziare configurando i dispositivi del PIC con cui vogliamo lavorare. Configureremo la porta B come uscita, dato che attraverso essa passeranno le linee dei dati verso l'LCD. La porta A la configureremo con due dei suoi pin come ingressi (RA3 e RA4) e altri tre come uscite (RA2:RA0). Configureremo il registro OPTION_REG con il valore desiderato per il predivisore del temporizzatore TMR0. Passiamo ora a configurare il display. Definiremo una posizione iniziale per il cursore utilizzando una variabile "Cur_Pos" che dovremo definire nell'intestazione del programma. Eseguiamo la sequenza di inizio dell'LCD e cancelliamo il display posizionando il cursore all'inizio, sequenza che viene anche chiamata Home. Infine configureremo il display attivandolo e abilitando il cursore, disattivando il lampeggio.

L'ultimo registro che dobbiamo configurare in questa parte del programma è il registro EEADR. Questo registro indica l'indirizzo iniziale della memoria EEPROM e deve essere impostato a zero.

messaggi - Blocco note

File Modifica Formato Visualizza ?

```
;In questo esempio realizzeremo un generatore di messaggi per il LCD. Con RA4 a "1", il
;sistema è in modo programmazione. In questo modo il messaggio viene scritto nella EEPROM
;dei dati. Con RA4 a "0" entriamo nel modo riproduzione. Il messaggio scritto nella
;EEPROM si visualizza sul display LCD.
;
;Quando l'interruttore RA3 è a livello "1", sul display appaiono in modo sequenziale
;i diversi caratteri disponibili. Impostandolo a "0" si seleziona quello presente
;che viene scritto nella EEPROM. Passando nuovamente a "1" si seleziona il
;carattere successivo.
;
                List    p=16F870        ;Tipo di processore
                include "P16F870.INC"   ;Definizioni dei registri interni

Lcd_var         equ     0x20            ;Variabili (2) della routine di gestione del LCD
Delay_Cont      equ     0x21            ;Variabile per la temporizzazione
Temporale_1     equ     0x22            ;Variabile Temporale
Temporale_2     equ     0x23            ;Variabile Temporale
Pos_Curs        equ     0x24            ;Posizione del cursore

                org     0x00            ;Vector de Reset
                goto    Inizio
                org     0x05            ;Salva vector di interrupt

                include "LCD_Cxx.inc"   ;Include la routine di gestione del LCD

;*****************************************************************************
```

Intestazione del programma.

messaggi - Blocco note

File Modifica Formato Visualizza ?

```
;*****************************************************************************
;
Inizio          clrf    PORTB           ;Cancella i latch di uscita
                bsf     STATUS,RP0      ;Seleziona banco 1
                clrf    TRISB           ;Porta B si configura come uscita
                movlw   b'00011000'
                movwf   TRISA           ;RA0-RA2 uscite, RA3-RA4 ingressi
                movlw   b'00000110'
                movwf   ADCON1          ;Configuriamo la porta A come I/O digitali
                movlw   b'00000111'
                movwf   OPTION_REG      ;Prescaler da 256 per il TMR0
                bcf     STATUS,RP0      ;Seleziona banco 0
                movlw   0x80
                movwf   Pos_Curs        ;Posizione iniziale del cursore
                call    LCD_INI         ;Sequenza di inizio del LCD
                movlw   b'00000001'
                call    LCD_REG         ;Cancella LCD e Home
                movlw   b'00001110'
                call    LCD_REG         ;Invia istruzione: LCD ON, cursore ON e blink OFF
                bsf     STATUS,RP1      ;Passiamo al banco 2
                clrf    EEADR           ;Indirizzo iniziale della EEPROM
```

Configurazione dei dispositivi.

Esercizio: generatore di messaggi (II)

Continuiamo l'applicazione del generatore di messaggi terminando lo sviluppo del codice ed eseguendo la compilazione e simulazione dell'esercizio. Sono già state definite le subroutine da utilizzare, l'intestazione del programma e la configurazione dei dispositivi del PIC.

Programma principale

Per continuare con il programma e affrontare la risoluzione dell'esercizio, dovremo avvalerci dell'aiuto dell'organigramma. La prima condizione prevista dopo la configurazione dei dispositivi, è lo stato dell'ingresso RA4. In funzione del valore RA4 selezioneremo il modo programmazione o riproduzione del messaggio. Il ciclo principale del programma – etichetta Loop – inizierà leggendo il valore dell'ingresso e prevedendo un salto condizionale in funzione del suo valore.

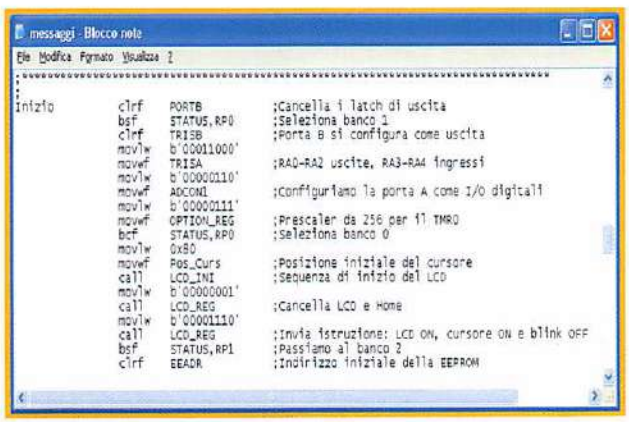
Modo programmazione

Nel caso in cui RA4=1 entreremo nel modo programmazione, quindi dovremo utilizzare una nuova variabile Temporale_1 per scrivere il primo carattere da visualizzare e leggere il valore dell'ingresso RA3 per sapere se il carattere visualizzato lo si deve memorizzare.

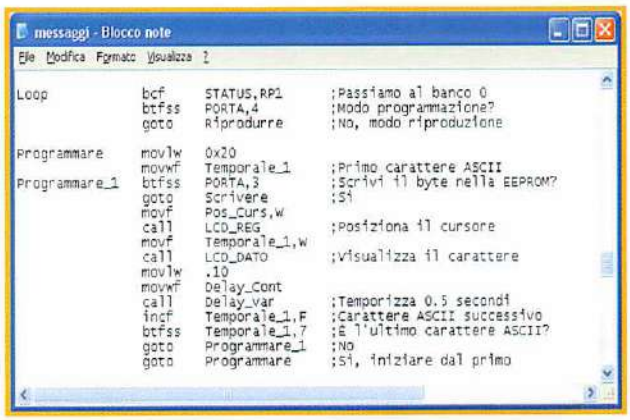
Se RA3=0 salteremo all'etichetta Scrivere, a partire dalla quale svilupperemo il codice necessario per scrivere il carattere nella memoria EEPROM. Se RA3=1 vedremo sul display i caratteri uno alla volta con un intervallo di 500 ms. Per fare questo posizioneremo prima il cursore nella posizione determinata dalla variabile Pos_Curs, visualizzeremo il carattere contenuto nella variabile Temporale_1, dopodiché richiameremo la routine di temporizzazione inizializzandola con il fattore 10 per ottenere 500 ms. Incrementeremo in seguito la variabile Temporale_1 per visualizzare il carattere successivo e, in funzione del fatto che sia o meno l'ultimo, ripeteremo il ciclo dal punto in cui assegniamo alla variabile il primo carattere o da quello in cui leggiamo lo stato di RA3. Nella figura in basso è riportato il codice corrispondente al blocco software che abbiamo appena visto.

Se RA3=0 scriveremo il carattere visualizzato. In questo caso, quindi, salteremo al codice localizzato a partire dall'etichetta Scrivere. La prima cosa che faremo sarà richiamare la routine per eliminare i rimbalzi, dopo ridurremo la variabile Temporale_1 per rimanere con il carattere visualizzato e, successivamente, lo scriveremo nella memoria EEPROM. Fatto questo, dobbiamo incrementare la posizione della memoria RAM per prepararci a scrivere un nuovo carattere, pulendola (ovvero scrivendo uno 0) per evitare possibili errori. Dobbiamo anche incrementare la posizione del cursore dell'LCD per presentare nuovamente i caratteri da scegliere.

Infine attenderemo che RA3 ritorni nuova-



Configurazione dei dispositivi.



Codice per visualizzare i caratteri da scegliere.

messaggi - Blocco note

File Modifica Formato Visualizza ?

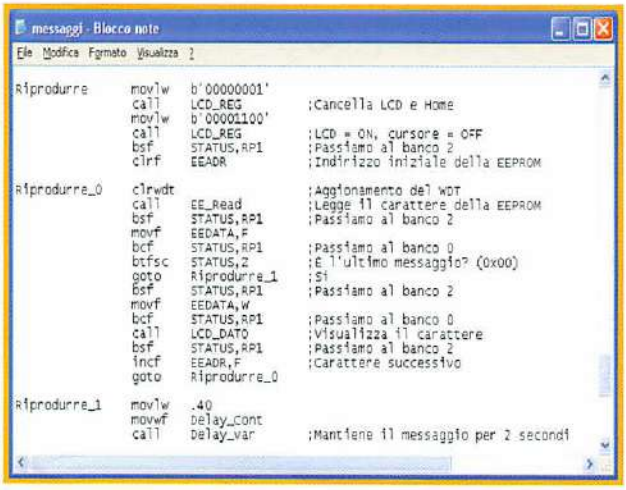
```
Scrivere        call    Delay_20_ms     ;Elimina rimbalzi
                decf    Temporale_1,w
                bsf     STATUS,RP1      ;Passiamo al banco 2
                movwf   EEDATA          ;Dato da scrivere nella EEPROM
                call    EE_Write        ;Scrivi il carattere
                bsf     STATUS,RP1      ;Passiamo al banco 2
                incf    EEADR,F         ;Indirizzo successivo della EEPROM
                clrf    EEDATA
                call    EE_Write        ;Scrivi 0x00
                incf    Pos_Curs,F      ;Posizione successiva del cursore
Scrivere_1      clrwdt                  ;Aggiorna il WDT
                btfss   PORTA,3         ;Attendi che RA3 ritorni a "1"
                goto    Scrivere_1
                call    Delay_20_ms     ;Eliminare rimbalzi
                goto    Loop            ;Carattere successivo del messaggio
```

Codice per salvare il carattere nella memoria.

mente a valore 1, istante in cui richiameremo la routine per eliminare i rimbalzi e torneremo all'inizio del programma all'etichetta Loop.

Modo riproduzione

Quando RA4=0 il PIC visualizza il messaggio scritto nella memoria EEPROM. I passaggi che seguiremo per sviluppare questo codice sono molto semplici. Per prima cosa prepariamo il display, lo cancelliamo e lo configuriamo. Dopo ci posizioneremo al primo indirizzo della memoria RAM e inizieremo un ciclo. All'interno del ciclo leggeremo il primo carattere della memoria e verificheremo che non sia l'ultimo; nel caso fosse l'ultimo salteremo a un'etichetta da dove richiameremo una routine di temporizzazione, inizializzandola con una variabile di valore 40 in modo che temporizzi due secondi. Se non è l'ultimo visualizzeremo il carattere, incrementeremo la posizione della memoria per andare al carattere successivo e ripeteremo il ciclo.



messaggi - Blocco note

File Modifica Formato Visualizza ?

```
Riprodurre      movlw   b'00000001'
                call    LCD_REG         ;Cancella LCD e Home
                movlw   b'00001100'
                call    LCD_REG         ;LCD = ON, cursore = OFF
                bsf     STATUS,RP1      ;Passiamo al banco 2
                clrf    EEADR           ;Indirizzo iniziale della EEPROM

Riprodurre_0    clrwdt                  ;Aggiornamento del WDT
                call    EE_Read         ;Legge il carattere della EEPROM
                bsf     STATUS,RP1      ;Passiamo al banco 2
                movf    EEDATA,F
                bcf     STATUS,RP1      ;Passiamo al banco 0
                btfsc   STATUS,Z        ;E l'ultimo messaggio? (0x00)
                goto    Riprodurre_1    ;Sì
                bsf     STATUS,RP1      ;Passiamo al banco 2
                movf    EEDATA,W
                bcf     STATUS,RP1      ;Passiamo al banco 0
                call    LCD_DATO        ;Visualizza il carattere
                bsf     STATUS,RP1      ;Passiamo al banco 2
                incf    EEADR,F         ;Carattere successivo
                goto    Riprodurre_0

Riprodurre_1    movlw   .40
                movwf   Delay_Cont
                call    Delay_var       ;Mantiene il messaggio per 2 secondi
```

Codice per riprodurre il messaggio.

Lampeggìo

Dopo aver riprodotto o visualizzato il messaggio per il tempo prefissato (2 s) il display inizia una sequenza di lampeggìo. Dovremo definire una nuova variabile Temporale_2 che svolga la funzione di contatore del lampeggìo. Ad esempio, possiamo inizializzare la variabile con un valore pari a 6, in modo che vengano

Ingresso	Stato	Descrizione
RA4	0	Riproduzione
	1	Programmazione
RA3	0	Visualizza carattere
	1	Scrivi carattere

Opzioni del programma in funzione degli ingressi.

```
Build Results
Building MESSAGGI.HEX...

Compiling MESSAGGI.ASM:
Command line: "C:\PROGRA~1\MPLAB\MPASMWIN.EXE /p16F870 /q C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM"
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 76 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 77 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 79 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 81 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 82 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 96 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 97 : Register in operand not in bank 0.  Ensure that b
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 106 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 108 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 110 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 112 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 122 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 147 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 150 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 151 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 165 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 170 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 175 : Register in operand not in bank 0.  Ensure that
Message[302] C:\PROGRA~1\MPLAB\PROGETTI\MESSAGGI.ASM 179 : Register in operand not in bank 0.  Ensure that

Build completed successfully.
```

Risultato della compilazione.

eseguiti 6 lampeggii. Accenderemo il display e attenderemo 250 ms, utilizzando la routine di temporizzazione, e spegneremo il display per un tempo pari al precedente. Fatto questo, ripeteremo il ciclo tante volte quante indicate nella variabile Temporale_2.

Terminato il lampeggio attenderemo un secondo e ripeteremo il programma dall'inizio, ovvero salteremo all'etichetta Loop.

Compilazione

Abbiamo terminato lo sviluppo del codice e dobbiamo compilarlo alla ricerca di eventuali errori.

Il programma completo potete trovarlo sul secondo CD allegato all'opera, all'interno della cartella LCD, con il nome "messaggio.asm". Creiamo un nuovo progetto su MPLAB cui allegheremo il nostro codice, lo visualizzeremo a video ed eseguiremo la compilazione. Il risultato di quest'ultima possiamo vederlo nell'immagine della figura in alto.

Simulazione

Per simulare l'applicazione apriamo le finestre abituali, quella dei registri delle funzioni speciali e una che contenga i registri che risultano essere più interessanti. Nella figura in basso potete vedere le variabili che abbiamo considerato come più importanti e il modo di visualizzarle (ASCII, decimale o binario). Inizieremo la simulazione passo a passo e, come

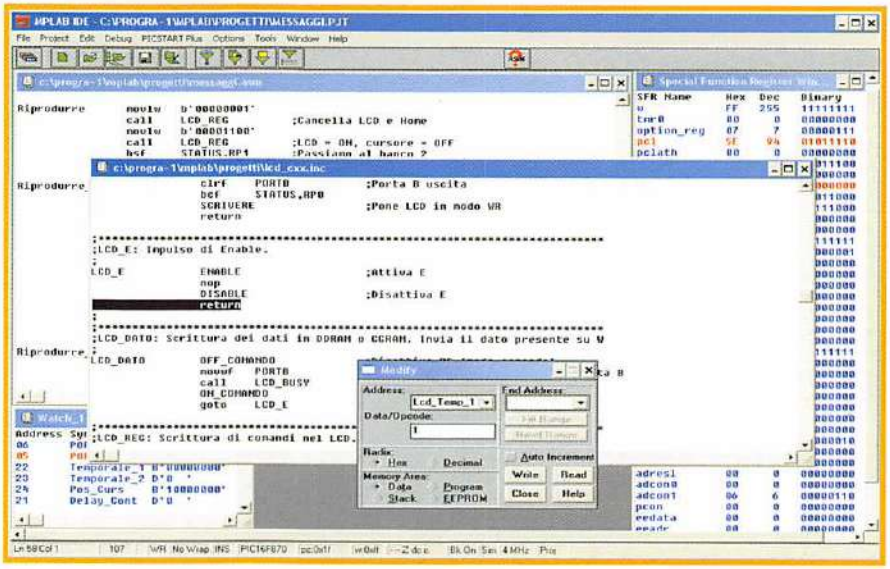
```
messaggi - Blocco note
File  Modifica  Formato  Visualizza  ?

;******************************** SEQUENZA DI LAMPEGGIO ********************************
Blink       movlw   .6              ;Inizializza il contatore di lampeggi
            movwf   Temporale_2
Blink_1     movlw   b'00001100'     ;LCD a ON
            call    LCD_REG
            movlw   .5
            movwf   Delay_Cont
            call    Delay_var       ;Temporizza 0.25 secondi
            movlw   b'00001000'     ;LCD a OFF
            call    LCD_REG
            movlw   .5
            movwf   Delay_Cont
            call    Delay_var       ;Temporizza 0.25 secondi
            decfsz  Temporale_2,F   ;Ripete il lampeggio
            goto    Blink_1
            movlw   .20
            movwf   Delay_Cont
            call    Delay_var       ;Temporizza 1 secondo
            goto    Loop

            end                     ;Fine del programma sorgente
```

Codice della sequenza per il lampeggio.

```
Watch_1
Address  Symbol      Value
06       PORTB       A'.'
05       PORTA       B'00000000'
22       Temporale_1 B'00000000'
23       Temporale_2 D'0   '
24       Pos_Curs    B'00000000'
21       Delay_Cont  D'0   '
```

Registri più importanti visualizzati in una finestra indipendente.

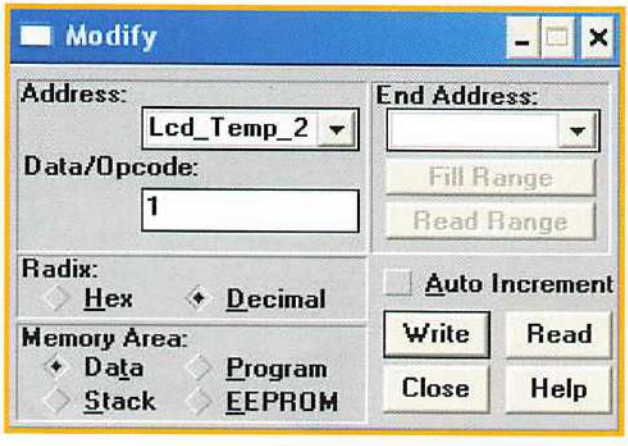
per gli esercizi precedenti, il simulatore entra nella libreria "Lcd_cxx.inc" per eseguire le routine del display LCD. Ricordate che per uscire da queste routine dobbiamo forzare le variabili Lcd_Temp_2 e Lcd_Temp_1 utilizzando la finestra Modify. Fatto questo e terminata l'inizializzazione del display, la simulazione continua all'interno del codice del programma principale configurando i dispositivi e il display LCD. La simulazione arriva al punto in cui si verifica lo stato dell'ingresso RA4. Dato che non abbiamo forzato il suo valore è chiaro che l'ingresso sarà a zero, quindi ci troviamo nel modo riproduzione. Siccome non c'è alcun messaggio registrato nella memoria, il programma presenta il contenuto della memoria stessa ma non trova la fine del messaggio.



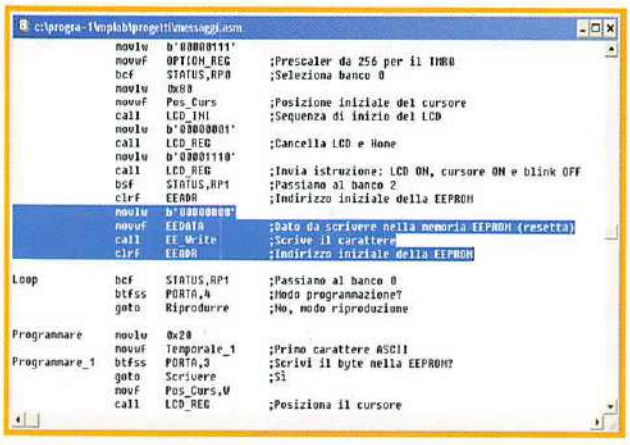
MPLAB durante la simulazione.

Abbiamo quindi trovato un errore, infatti, dato che l'applicazione alla partenza entra nel modo riproduzione, non trovando la fine del messaggio continua a riprodurre costantemente il contenuto della memoria EEPROM. Possiamo risolvere questo problema in due modi, il primo consisterebbe nel provocare un interrupt ogni volta che cambia lo stato dell'ingresso RA4 (in questo modo usciremmo dal ciclo), e il secondo, più semplice, sarebbe quello di cancellare il contenuto del primo indirizzo della RAM al momento di configurare i dispositivi. Nell'immagine possiamo vedere come risolvere questo errore.

Come avete potuto verificare, mediante la simulazione è possibile rilevare degli errori di funzionamento che in altro modo non potremmo evidenziare. Modificate il programma utilizzando l'editor di MPLAB, salvatelo e compilate nuovamente. Ripetete la simulazione e verificate che ora avvenga in modo adeguato. Per eseguire una simulazione completa sarà necessario utilizzare il simulatore di stimoli asincroni e attivare gli ingressi, in questo modo sarà possibile analizzare le diverse combinazioni verificando il funzionamento completo dell'applicazione.



Forziamo i valori delle variabili.



```
c:\progra~1\mplab\progetti\messaggi.asm
                movlw   b'00000111'
                movwf   OPTION_REG      ;Prescaler da 256 per il TMR0
                bcf     STATUS,RP0      ;Seleziona banco 0
                movlw   0x80
                movwf   Pos_Curs        ;Posizione iniziale del cursore
                call    LCD_INI         ;Sequenza di inizio del LCD
                movlw   b'00000001'
                call    LCD_REG         ;Cancella LCD e Home
                movlw   b'00001110'
                call    LCD_REG         ;Invia istruzione: LCD ON, cursore ON e blink OFF
                bsf     STATUS,RP1      ;Passiamo al banco 2
                clrf    EEADR           ;Indirizzo iniziale della EEPROM
                movlw   b'00000000'
                movwf   EEDATA          ;Dato da scrivere nella memoria EEPROM (resetta)
                call    EE_Write        ;Scrive il carattere
                clrf    EEADR           ;Indirizzo iniziale della EEPROM
Loop            bcf     STATUS,RP1      ;Passiamo al banco 0
                btfss   PORTA,4         ;Modo programmazione?
                goto    Riprodurre      ;No, modo riproduzione
Programmare     movlw   0x20
                movwf   Temporale_1     ;Primo carattere ASCII
Programmare_1   btfss   PORTA,3         ;Scrivi il byte nella EEPROM?
                goto    Scrivere        ;Sì
                movf    Pos_Curs,W
                call    LCD_REG         ;Posiziona il cursore
```

Modifica per correggere l'errore.