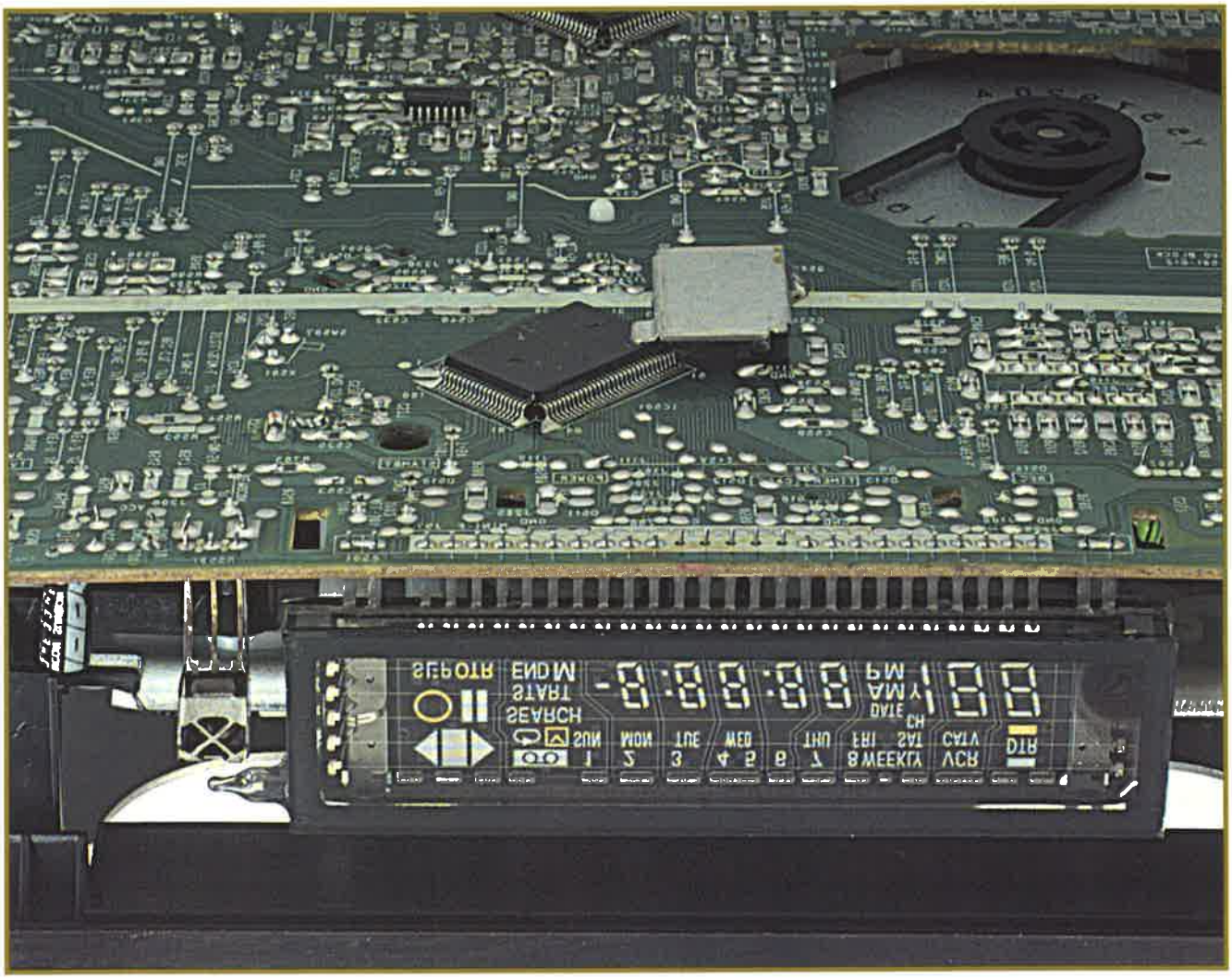
PROGRAMMAZIONE

Software
Strumenti di lavoro
Scheda delle Istruzioni
Applicazioni pratiche

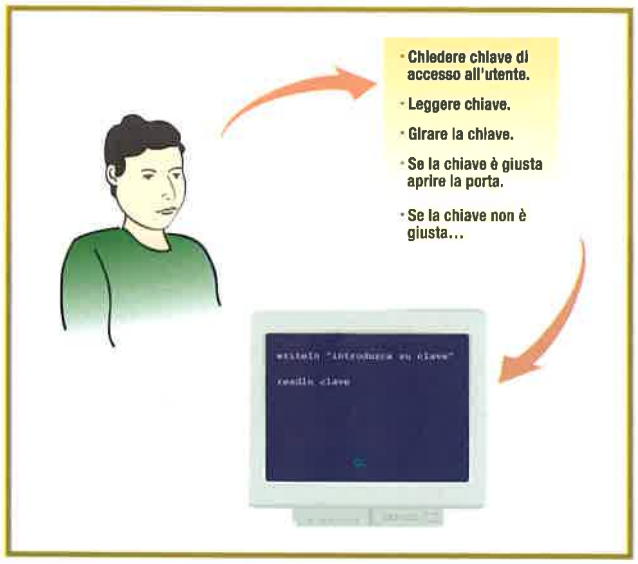
Programmazione: principi generali



Per poter utilizzare molti apparati "intelligenti" è necessario che siano programmati.

Per utilizzare al meglio moltissime apparecchiature "intelligenti" è necessario programmarle. Molti piccoli apparati elettronici di utilizzo comune, ad esempio un videoregistratore per registrare la nostra serie TV preferita, oppure un gioco all'interno del nostro computer, o la moltitudine degli apparati intelligenti che abbiamo intorno a noi, devono essere dotati di un programma per poter eseguire le loro funzioni. Programmare non è altro che indicare a un processore quello che deve fare in un linguaggio che questo comprende. Così quando premiamo il tasto play del video, le istruzioni che sono state fornite relative a quel tasto specifico, mettono in funzione il motore che trascina la pellicola, leggono le informazioni che questa contiene, ecc.

A partire da adesso, diventeremo da semplici utilizzatori della tecnologia, a programmatori che realizzano applicazioni per gli altri o per noi stessi. Per questo dobbiamo strutturare il nostro pensiero e tradurlo in uno dei molti linguaggi di programmazione esistenti. Un lin-



Le diverse fasi per la realizzazione di un programma.

guaggio di programmazione è un insieme di regole, simboli e parole speciali per costruire un programma. Come passo intermedio bisogna trasformare l'idea in quello che viene chiamato pseudo-codice, che è la rappresentazione del linguaggio che noi utilizziamo – il Linguaggio Naturale – in forma strutturata. Il risultato finale è il programma.

TERMINOLOGIA DI BASE

Un programma scritto in pseudo-codice deve servire da base per la realizzazione del programma finale, quale sia il linguaggio di programmazione che utilizzeremo. Per questo, in questa prima fase del programma, dobbiamo avere un'idea chiara di quello che il programma stesso deve fare, per non generare errori di tipo logico o semantico.

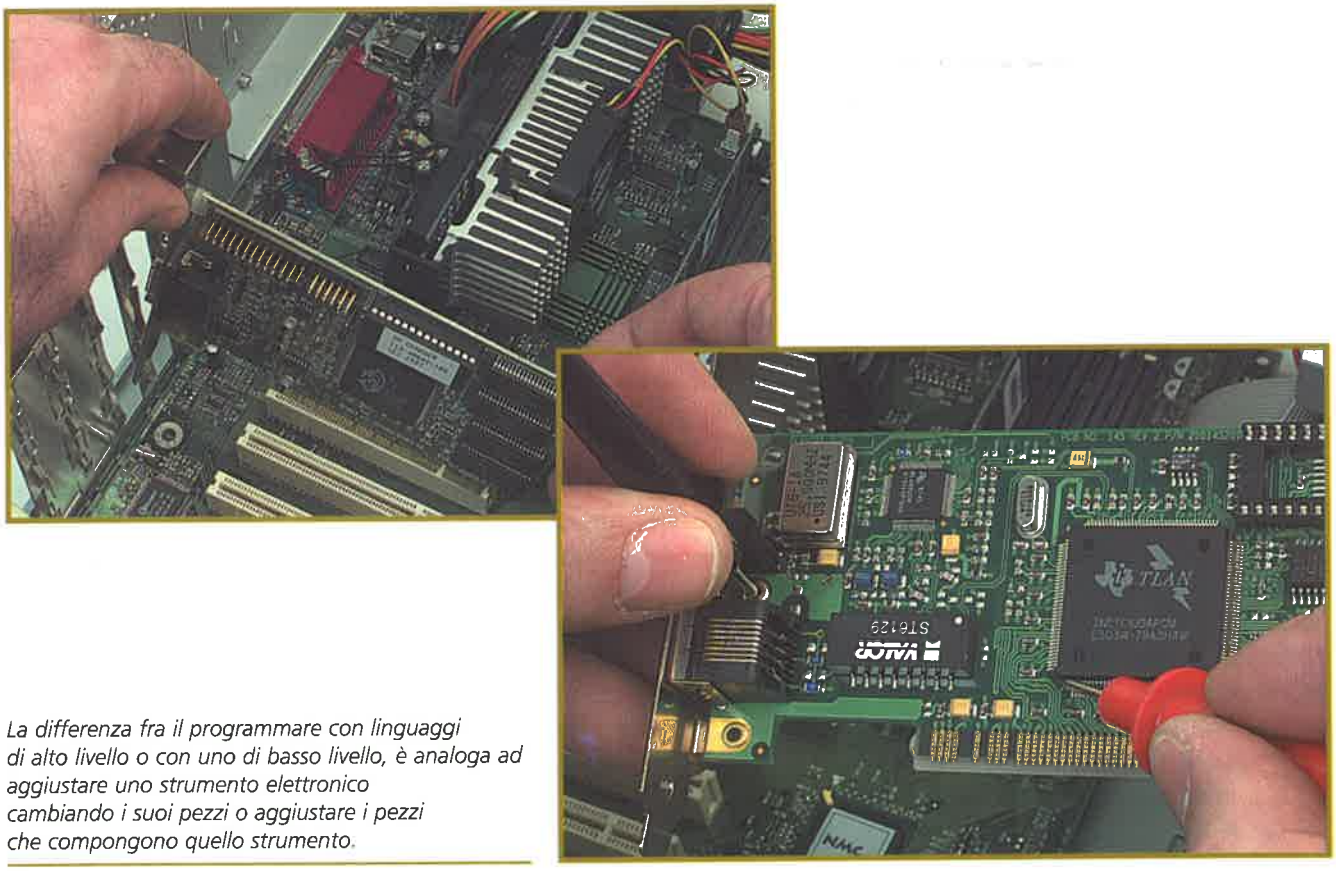
Una volta scelto il linguaggio di programmazione, bisognerà conoscerlo per non commettere errori di tipo sintattico. La conoscenza del programma è facile, basta seguire le norme che lo regolano. Dovremo conoscere la struttura del programma nel suo linguaggio, i tipi di dati che si possono utilizzare e i loro possibili valori, l'insieme di istruzioni, la struttura di controllo, e tutte le cose che vi spiegheremo successivamente per la corretta programmazione di Monty. Il nostro linguaggio sarà l'Assembler, e sfateremo il tipico mito che sia uno dei linguaggi di programmazione più difficili.

STRUMENTI DI LAVORO

Il primo caso è quello di un programma che funziona male, però non presenta errori sintattici, ed è difficile da risolvere; si vede che funziona male e che non dà i valori sperati, però non indica dove c'è l'errore. Il nostro caso è aggravato dal fatto che si tratta di un programma che deve funzionare con un hardware, e non in un computer, per cui l'errore potrebbe non dipendere dal cattivo funzionamento del programma, ma da problemi dell'hardware controllato. Anche se l'esperienza fornita dalla pratica è la migliore alleata in questi casi, noi disponiamo di strumenti che ci facilitano questo lavoro. Nella figura sono rappresentati i passi da seguire quando un progetto, come la programmazione del nostro microrobot, ha una parte software e una parte hardware. La descrizione di ognuno di questi passi verrà fatta nei fascicoli seguenti. Osserviamo attentamente le differenze dei testi all'interno dei rettangoli, e all'interno dei rombi. Le etichette dei dati, la forma di rappresentazione delle azioni, sono tutti dati essenziali al momento di programmare e che meritano una pagina a parte.

Fasi per la realizzazione di un progetto.

Strumenti di lavoro: i linguaggi di programmazione



La differenza fra il programmare con linguaggi di alto livello o con uno di basso livello, è analoga ad aggiustare uno strumento elettronico cambiando i suoi pezzi o aggiustare i pezzi che compongono quello strumento.

Come già abbiamo spiegato nelle pagine precedenti, un linguaggio di programmazione è un insieme di regole, simboli e parole speciali per la costruzione di un programma. Esistono molti linguaggi di programmazione, di svariati tipi, con caratteristiche a volte simili e a volte molto diverse fra di loro.

CLASSIFICAZIONE DEI LINGUAGGI

Una delle molte classificazioni che si possono fare con i linguaggi di programmazione, differenzia la loro similitudine dal codice che è direttamente accettato dal computer, cioè il "codice macchina". Quindi, si è soliti dividere il settore in linguaggi di alto livello e in linguaggi di basso livello. Anche nel lavoro con il PIC seguiremo questa distinzione. Un linguaggio di alto livello è più vicino al modo di pensare delle persone, mentre un linguaggio di basso livello è più vicino al tipo di dati che accetta il microprocessore. Entrambe le categorie hanno vantaggi e svantaggi.

IL C E IL BASIC

I linguaggi di alto livello più utilizzati per programmare i microprocessori, sono il C e il Basic. Entrambi sono linguaggi di impiego generale, cioè si utilizzano per applicazioni molto diverse fra loro. Questi linguaggi sono prodotti da diverse Case e, pur mantenendo una struttura fondamentalmente simile, possono presentare alcune differenze, anche se, come qualsiasi linguaggio di programmazione, la loro efficacia dipende dal modo con cui il programmatore li utilizza.

```
main()
{
     set_bit( STATUS, RP0 );
     OPTION_REG = 7;              //   Assegna un divisore di 1:256
     set_tris_b( 0 );
     clear_bit( STATUS, RP0 );
     output_port_b( 0 );

     enable_interrupt( GIE );
     enable_interrupt( T0IE ); //   Abilita interrupt del TMR0


     while( 1 );                  //   Ciclo infinito
}
```

Frammento di un programma scritto in linguaggio C.

Nella figura precedente è illustrato un esempio di frammento di programma scritto in linguaggio C per il PIC. In esso si configura nel banco 1, la Porta B come uscita, e si assegna un divisore per 256 alla frequenza di lavoro per il suo utilizzo con il timer TMR0. Si continua togliendo gli zeri dalla Porta B e si abilita l'interrupt all'attivazione del TMR0. Proseguendo, si entra dentro ad un ciclo infinito che verrà forzato solamente dal TMR0 alla fine del suo conteggio.

A prima vista si possono notare alcune caratteristiche di questo programma, che sono quelle che hanno contribuito a renderlo uno dei linguaggi più utilizzati. Le istruzioni di controllo strutturate, come la "while" danno chiarezza al programma: in questo esempio è utilizzata per restare in un ciclo sino a che la condizione scritta all'interno delle parentesi è realizzata. In pratica si converte in un ciclo infinito sino a che un evento non modifichi la condizione scritta all'interno delle parentesi.

Se si volessero ripetere diverse istruzioni per il tempo in cui resta valida una determinata condizione, basterebbe inserire queste istruzioni fra le parentesi. Trattandosi di un pezzo di programma non si può apprezzare la seconda delle caratteristiche fondamentali: nonostante sia collegato al programma principale, e quindi sia stato abilitato da un evento di interrupt, all'interno di questo spezzone potremmo inserire un altro richiamo ad un ulteriore pezzo di programma - anche quest'ultimo sarà inserito all'interno di parentesi - e sarà indipendente dal programma principale, comunicando con esso solo per lo scambio di parametri e risultati.

Oltre alle istruzioni standard, il linguaggio C specifico per il PIC ne dispone di altre, che si possono considerare delle piccole funzioni di uso frequente: come il settaggio da 1 a 0 di un registro specifico, o di un bit di un registro, o l'abilitazione di un interrupt.

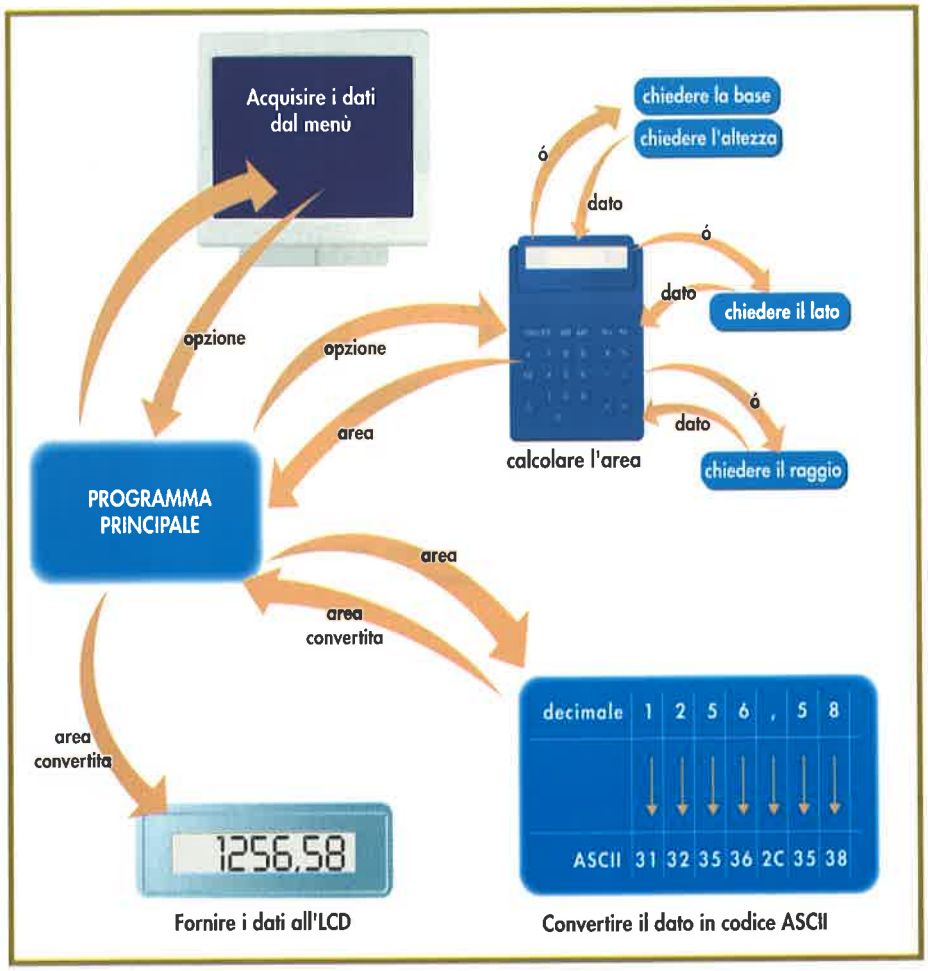
La differenza con linguaggi come l'Assembler non sta nella complessità del codice, inteso come chiarezza, ma nel minimo numero di funzioni utilizzate per dire quello che si deve fare. Altre funzioni

USO DELLE ISTRUZIONI DI CONTROLLO STRUTTURATE

UTILIZZO DI PROCEDIMENTI E FUNZIONI INDIPENDENTI PIÙ PICCOLE, PER SUDDIVIDERE UN PROGRAMMA PIÙ GRANDE. FACILITA LA COMPRENSIONE DEL PROGRAMMA.

CONTIENE FUNZIONI E PROCEDIMENTI GIÀ IMPLEMENTATE, CHE FACILITANO IL LAVORO DEL PROGRAMMATORE.

Vantaggi dell'uso dei linguaggi di alto livello



Il programma principale comunica normalmente con i vari procedimenti per lo scambio di parametri e risultati.

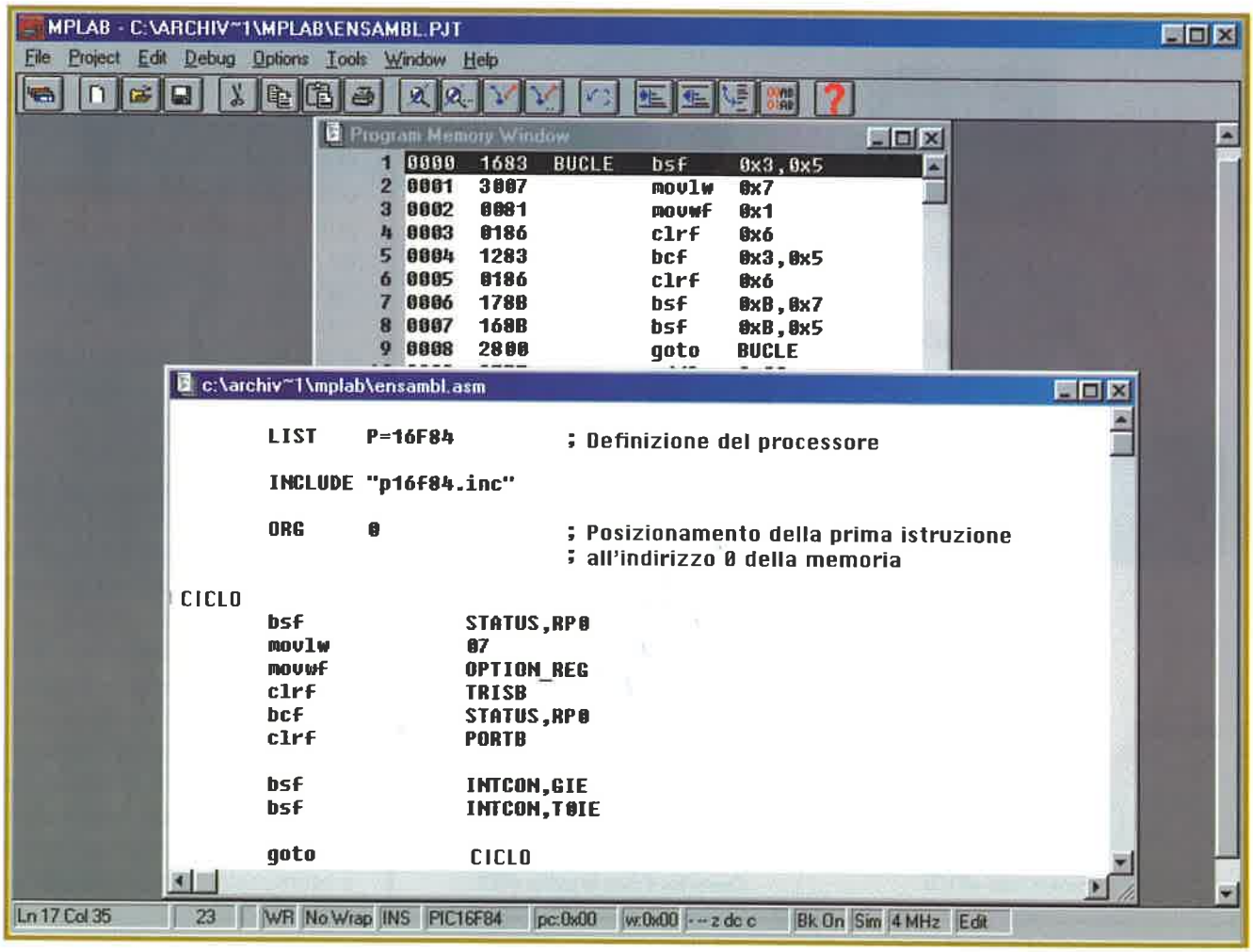
come la capacità di attendere un evento, facilitano l'utilizzo, a vantaggio della chiarezza.

L'ASSEMBLER

È il linguaggio denominato di basso livello ed è il più vicino al computer. Nella figura a lato è illustrato il precedente programma redatto in Assembler. Il programma scelto è molto simile sia in un linguaggio che nell'altro, e non presenta grandi complicazioni. Senza dubbio, per chi non conosce nessuno dei due linguaggi, potrebbe risultare più comprensibile il linguaggio scritto in C, dato che non necessita della conoscenza dell'architettura del PIC per realizzare le istruzioni, ed è più intuitivo. Come contropartita, il programma scritto in Assembler si traduce in tante posizioni di memoria quante sono le istruzioni utilizzate, nel nostro caso nove (CICLO non è un'istruzione). Nel programma scritto in C invece, ogni istruzione scritta – quelle dell'esempio riportato sono sette più la struttura "while" – si tradurrà in una o più istruzioni macchina (posizioni di memoria), a seconda del compilatore che verrà utilizzato. Questo perché in Assembler ogni istruzione si traduce con un solo codice macchina, al contrario dei linguaggi di alto livello. Possiamo osservare questa particolarità nella figura della pagina seguente, dove la seconda colonna della finestra del programma (Program Memory Window) riflette la posizione in cui si trova ogni istruzione. Benché la programmazione in Assembler sia più faticosa per chi la esegue, il codice generato è ottimizzato, con un considerevole risparmio di spazio occupato dal programma, cosa molto importante, considerando le restrizioni del PIC in quanto a capacità di memoria, che nel caso del PIC16F84 è solo di 1024 posizioni. In ogni caso i compilatori C di ultima

```
CICLO
      bsf       STATUS,RP0
      movlw     07
      movwf     OPTION_REG
      clrf      TRISB
      bcf       STATUS,RP0
      clrf      PORTB

      bsf       INTCON,GIE
      bsf       INTCON,T0IE

      goto      CICLO
```

Frammento di programma in linguaggio Assembler.



In Assembler ogni istruzione si traduce come un solo codice macchina.

generazione producono un codice ben ottimizzato.

Come abbiamo detto, una delle caratteristiche che fanno sì che il linguaggio C sia uno dei più utilizzati, è l'uso delle funzioni. Nella figura a lato si mostrano alcune funzioni di un compilatore del C per il PIC. La maggior parte sono funzioni molto specifiche del lavoro col PIC, che non si incontrano in compilatori di C standard.

Per inciso, dato che sono molto specifici non apportano vantaggi rispetto all'Assembler, se non per il fatto che utilizzano espressioni molto più chiare.

Dove realmente si nota la potenza del C con i microprocessori, è nella struttura di controllo, nei procedimenti, e nelle funzioni proprie dei linguaggi di alto livello.

SET_TRIS_A	SET_TRIS_B	SET_TRIS_C
OUTPUT_PORT_A	OUTPUT_PORT_B	OUTPUT_PORT_C
OUTPUT_HIGH_PORT_A	OUTPUT_HIGH_PORT_B	OUTPUT_HIGH_PORT_C
OUTPUT_LOW_PORT_A	OUTPUT_LOW_PORT_B	OUTPUT_LOW_PORT_C
INPUT_PORT_A	INPUT_PORT_B	INPUT_PORT_C
INPUT_PIN_PORT_A	INPUT_PIN_PORT_B	INPUT_PIN_PORT_C
SET_MODE	SET_OPTION	
DELAY_S	DELAY_MS	DELAY_US
ENABLE_INTERRUPT	DISABLE_INTERRUPT	
SET_BIT	CLEAR_BIT	
PUTCHAR	GETCHAR	
BCD_TO_CHAR	CHAR_TO_BCD	
CLEAR_WDT	SLEEP	NOP

Alcune funzioni di uno dei tanti compilatori C per il PIC.

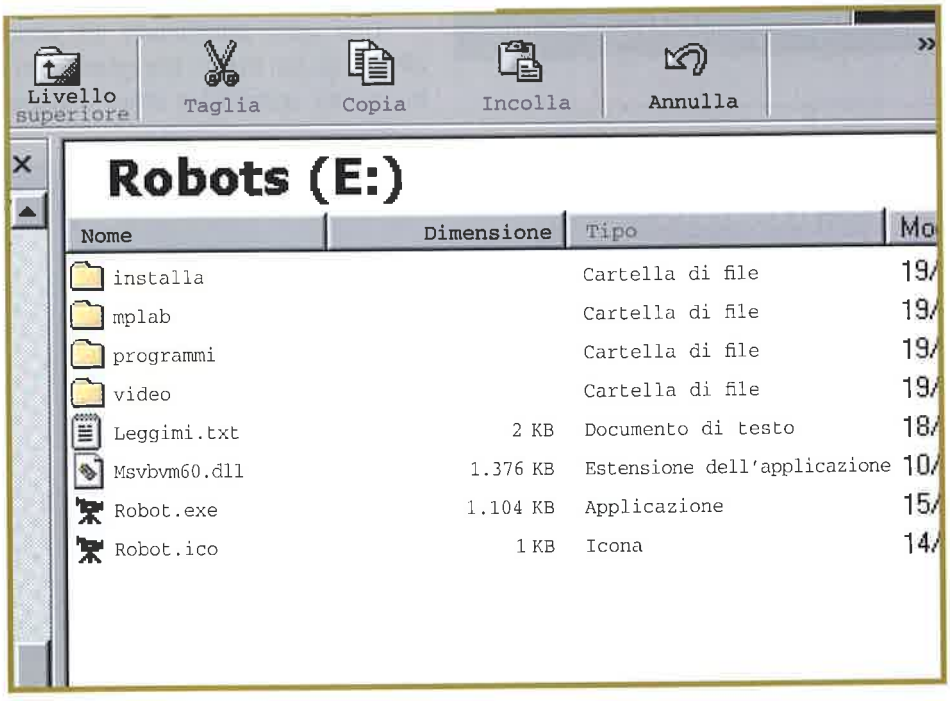
Strumenti di lavoro: installazione e uso del CD

A destra, il CD allegato al primo fascicolo.

Il CD-ROM che commenteremo, è quello allegato al primo fascicolo di quest'opera. È formato da varie cartelle e files con diversi contenuti, che verranno utilizzati per il lavoro con il nostro microrobot. Ora lo analizzeremo in modo dettagliato.

CONTENUTO DEL CD

Introducendo nel lettore il CD e selezionandolo, si accede alle sue quattro cartelle, e ai quattro files, così come indicato nella figura in basso. Il CD non è autoinstallante. Le informazioni che troviamo nel file leggimi.txt, descrivono il modo di installazione e il contenuto di queste cartelle, che si possono aprire con qualunque editor di testo. Gli altri 3 files, per ora, non li prenderemo in considerazione.



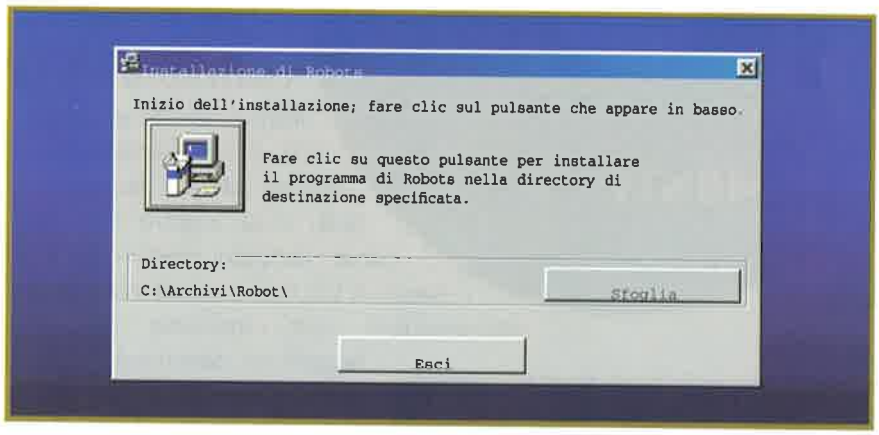
Contenuto del CD.

CARTELLA INSTALLA

La prima cartella da esaminare è la cartella "installa". Una volta aperta si dovrà eseguire il file setup.exe, così si installerà il programma robot.exe, che ci permetterà di visualizzare, tramite il PC, i video dimostrativi del robot Monty, e le differenti prove dei concorsi di microrobots.

Questa installazione, installa solo la parte indispensabile del software, dato che il CD dovrà

Videata iniziale del Programma di installazione di Robots.



Scelta della directory di installazione.

Proseguendo nell'installazione, ci verrà presentata la videata successiva, nella quale dovremo decidere l'indirizzo dove installare il software. Si può lasciare l'indirizzo già predisposto, in questo caso C:\Archivi\Robot\ o cambiare indirizzo con un altro che ci sembri più adeguato.

CARTELLA VIDEO

Nella cartella video troveremo i video dimostrativi, con prove di lotta e di ricerca sui differenti tipi di microrobots, tra questi anche Monty. Per vederli non dovremo entrare direttamente nella cartella, ma eseguire il programma Robot.exe dalla cartella principale, che è già stato correttamente installato seguendo i passi spiegati precedentemente. Apparirà la videata di inizio.

La funzione di ogni tasto è semplice, e ci ricorda quelli che sono normalmente utilizzati negli apparecchi musicali o video. All'inizio si potrà solo attivare il pulsante superiore, che ci permette di scegliere il video che si vuole visualizzare.

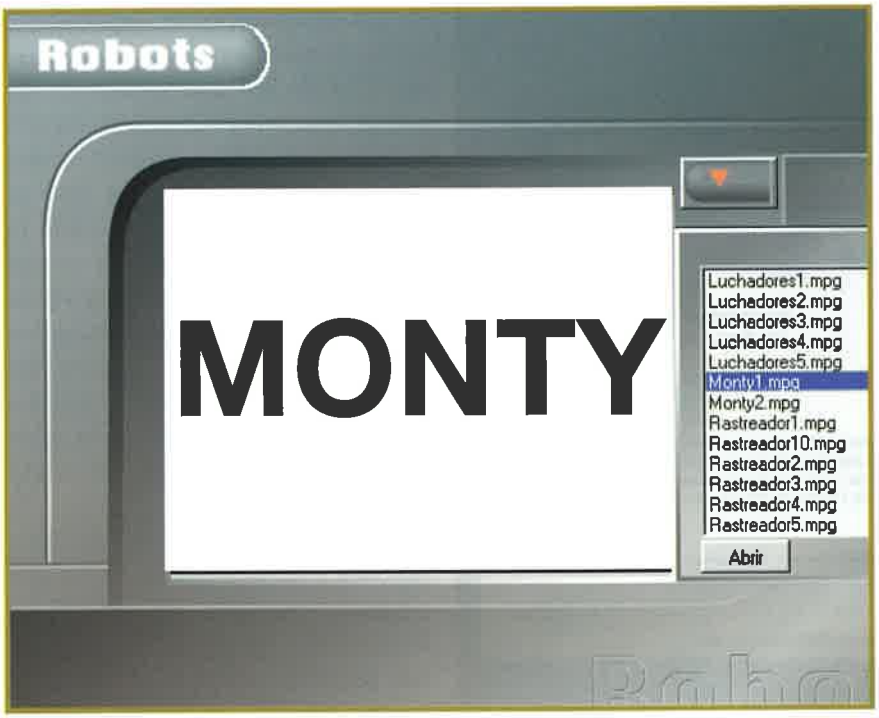
Una volta selezionato un video all'interno del menù, bisognerà aprirlo, questa operazione attiverà anche essere presente nel lettore al momento dell'esecuzione dei video; in questo modo i video non verranno copiati sul computer, permettendoci di risparmiare spazio.

Questo programma di visualizzazione dei video è predisposto per lavorare con Windows 98.

Quando lo eseguiremo apparirà la schermata iniziale del programma.

Se non si stanno eseguendo altri programmi, si cliccherà sul pulsante "accetta", o al contrario si dovrà uscire dall'installazione e rientrare dopo che si saranno chiusi gli altri programmi.

Schermata iniziale del video dimostrativo di Robots.



Scelta del video da visualizzare.

CARTELLA PROGRAMMI

Questa cartella contiene tre programmi esempio, che in seguito verranno caricati sul nostro robot Monty per fargli eseguire diverse operazioni. Queste operazioni, sono quelle tipiche per iniziare la programmazione di Robots: come seguire una traiettoria, evitare degli ostacoli o percorrere un determinato tracciato, e servono da base per la realizzazione di altre applicazioni molto più complesse.

La metodologia di esecuzione per la programmazione del robot, sarà tema di studio in un fascicolo do prossima pubblicazione.

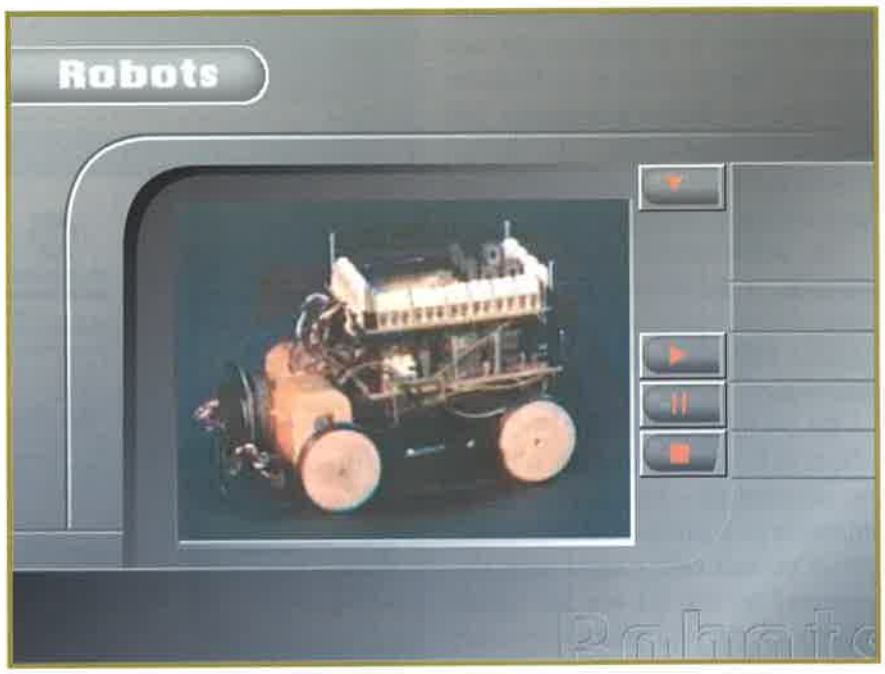
CARTELLA MPLAB

Quest'ultima cartella contiene un solo programma eseguibile per l'installazione di un'applicazione, che porta il medesimo nome della cartella: MPLAB. In realtà è formato da vari programmi, integrati in uno solo, che formano quello che viene comunemente chiamato "ambiente di sviluppo". Questo programma è indispensabile per le applicazioni con il PIC e dovrà essere studiato in modo la

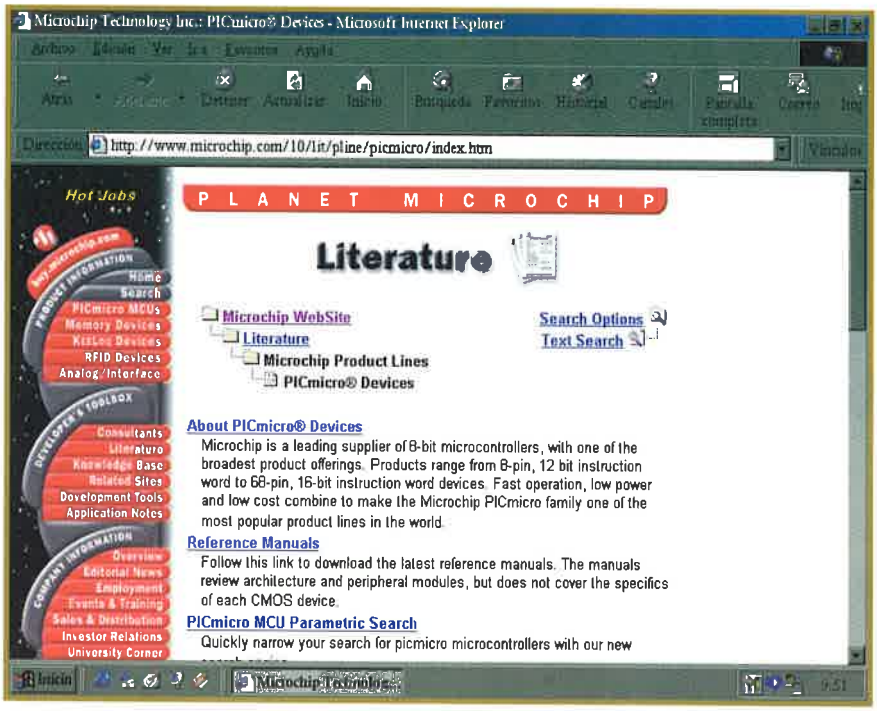
gli altri tre tasti. Partendo dall'alto verso il basso le icone indicano la messa in marcia, la pausa, la nuova attivazione, e lo stop definitivo. La videata centrale sarà quella che ci mostra il video prescelto.

Per uscire dal programma di visualizzazione, bisognerà cliccare sopra la parola Robots nell'angolo in alto a sinistra del monitor.

Se ci fossero dei problemi con la visualizzazione dei video, si dovrà reinstallare Windows Media Player, che è il software che ci permette queste visualizzazioni, lo faremo eseguendo il programma mpfull.exe, che si trova nella cartella "installa". Alla fine dell'installazione, verrà attivata la procedura di connessione alla pagina web di Windows Media Player, che si può cancellare senza nessun problema, perché non provoca nessun errore di installazione. Inoltre, se il software di accesso a Internet non fosse già installato sul PC, si attiverà una finestra in cui ci verrà chiesto se vogliamo installare questo tipo di software; a questo punto sarà quindi necessaria un'altra cancellazione.



Esempio di visualizzazione di un video.



Come ottenere la più completa informazione sui PIC.

sua installazione sia per il suo utilizzo. MPLAB è fornito direttamente da Microchip, che è la Casa produttrice dei microprocessori PIC. Il software lo troviamo solamente sul CD, però se si accede alla pagina web http://www.microchip.com/10/lit/pline/picmicro/index.htm si otterranno tutte le informazioni aggiornate riferite ai PIC e ai prodotti ad essi relativi. I file saranno sia in formato PDF che in forma compressa ZIP, oppure come file di testo. Troveremo inoltre diverse utility per lavorare con i microprocessori.

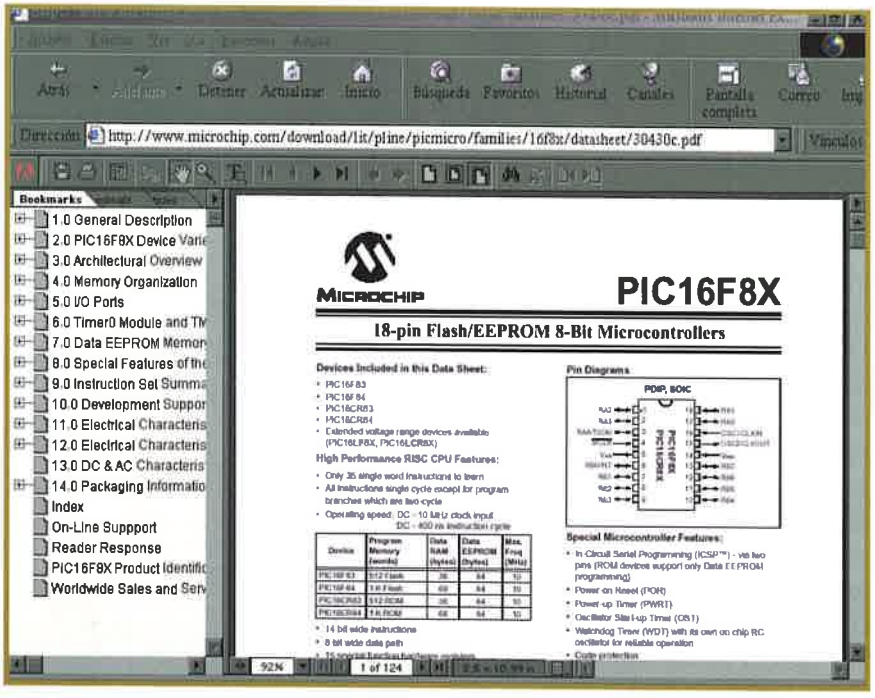
Le informazioni più complete riguardo il PIC16F84, protagonista della nostra opera, si trovano nella sezione "PICmicro MCUs" di questa schermata.

Da qui in poi troveremo i PIC classificati nelle diverse famiglie, e potremo selezionare le informazioni che ci interessano a partire dal PIC16X8X, che ci porterà nella pagina dove troveremo il file che ci serve. Navigando all'interno del sito potremo trovare tutte le informazioni di cui avremo bisogno.

Le informazioni ci appariranno, come mostra la figura, dopo aver aperto il file PDF corrispondente. Gli archivi in formato PDF si leggono con il programma Adobe Acrobat Reader 3.0 o superiore. Nel caso non lo avessimo si può scaricare dal sito di Adobe. Queste informazioni possono essere lette direttamente a video, oppure stampate, come qualsiasi file di testo. Nella pagina sinistra del video troviamo un piccolo indice che ci permette di spostarci nelle varie sezioni del documento in modo comodo e rapido.

Oltre ai microprocessori, Microchip fabbrica e commercializza anche altri dispositivi: memorie, convertitori, kit di sicurezza, amplificatori operazionali eccetera. Anche le informazioni relative a questi componenti si trovano sul sito web e possono essere liberamente consultate.

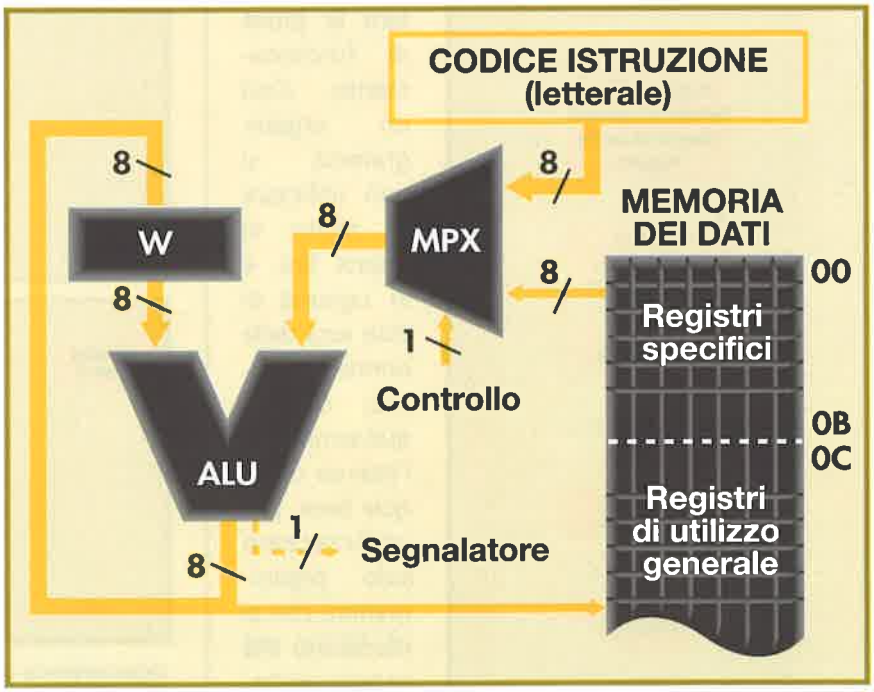
Nelle prossime pagine utilizzeremo questo CD per installare sul nostro computer gli strumenti software che ci fornisce Microchip per il lavoro con i microprocessori PIC.



Dati tecnici del PIC16F84 in formato PDF.

Programmazione: gli organigrammi

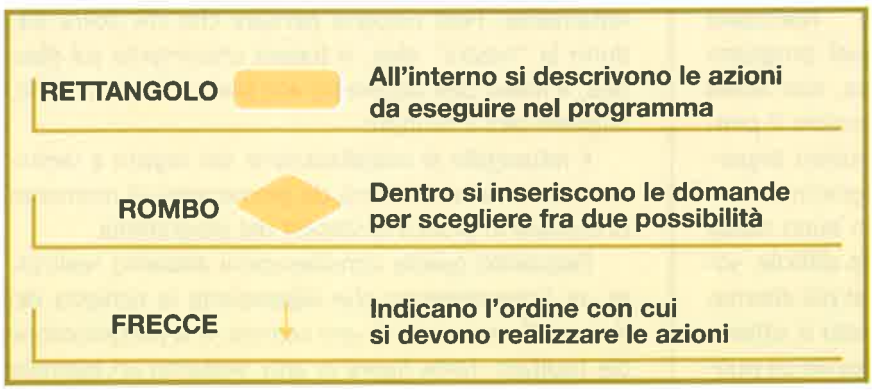
Le persone hanno la necessità di esprimere il loro pensiero sotto forma di comunicazione con gli altri. Alcuni lo fanno attraverso la pittura, altri attraverso la musica, la scultura, la scrittura e il ballo. Gli organigrammi sono la forma di espressione che più ci aiuta a plasmare l'idea astratta di un programma, in una serie di decisioni e azioni che vengono riprodotte in forma logica e ordinata. Così come nell'arte, esistono molti modi di esprimere un'idea mediante un organigramma, è compito dell'analista, cioè colui che compone l'organigramma, seguire una serie di "norme d'uso" per far sì che ciò che si vuole esprimere venga interpretato da tutti nello stesso modo.



Considerazioni al momento di costruire un organigramma.

SIMBOLI DI RAPPRESENTAZIONE DI UN ORGANIGRAMMA

Un organigramma è formato da rettangoli e rombi che si uniscono tra di loro con le frecce. I rettangoli rappresentano azioni che vengono eseguite nel programma. Possono essere semplici, come visualizzare un dato, o complesse come "Calcolare l'area di una figura scelta dall'utilizzatore, e mostrarla nel video insieme alla sua rappresentazione....". In questo caso può essere necessario un secondo organigramma, per realizzare meglio le azioni. I rombi contengono domande che vengono poste in base ai dati variabili all'interno del programma: sensori che forniscono un dato al sistema, opzioni dell'utilizzatore scelte attraverso interruttori, tastiera, eccetera. A seconda della risposta a una specifica domanda, bisognerà prendere una decisione per l'azione successiva. Le frecce indicano l'ordine con cui si devono eseguire le azioni e le domande.

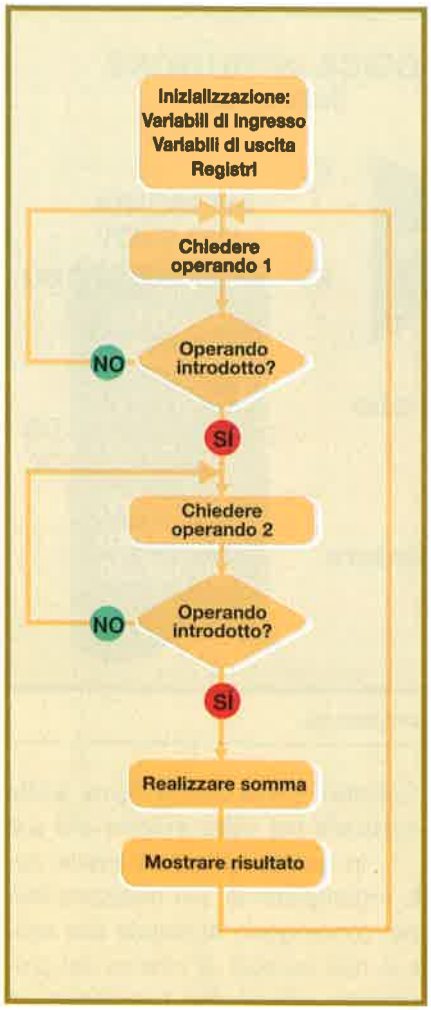


Simboli di rappresentazione di un organigramma.

CARATTERISTICHE DEGLI ORGANIGRAMMI

Dobbiamo tenere conto che il lavoro con i microprocessori per la soluzione

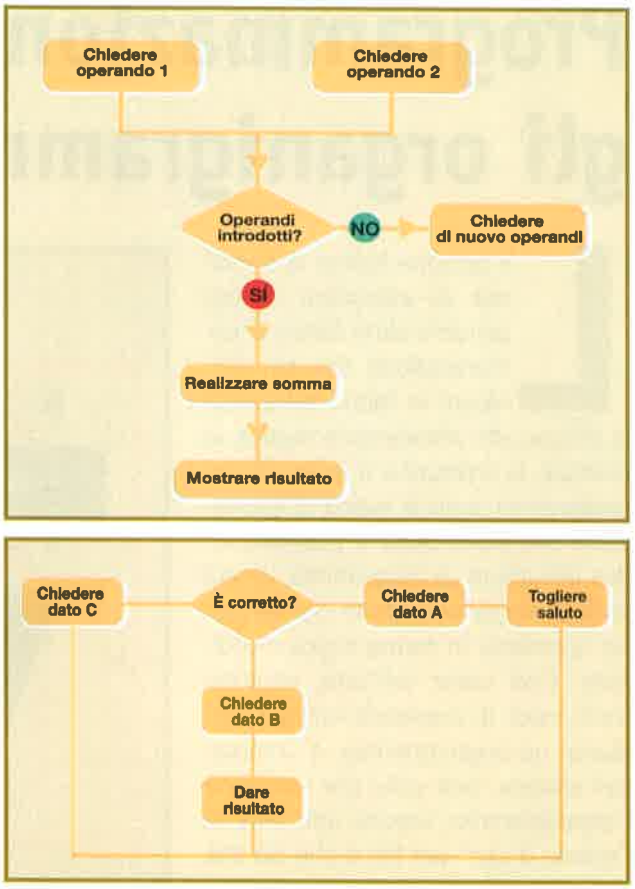
dei problemi è composto da varie fasi, che riguardano sia la parte software sia quella hardware. Normalmente quando si realizza un progetto, soprattutto se si lavora in un'equipe di sviluppo, la persona che realizza l'analisi del sistema, non è la stessa che più tardi stenderà il programma, né quella che verificherà le soluzioni, o farà le prove di funzionamento. Così un organigramma si può utilizzare in molte di queste fasi, e in ognuna di esse avrà delle norme specifiche; comunque sempre all'interno di regole base. Noi utilizzeremo solo organigrammi che si riferiscono alla rappresentazione della struttura di un programma, ed è necessario che questi siano molto precisi, perché chi più tardi sarà incaricato di realizzare quel programma, non abbia difficoltà al momento di tradurre in istruzioni il pensiero di un altro. Ecco alcune considerazioni importanti al momento di costruire un organigramma.



Meno cose si lasciano alla libera interpretazione, meglio è.

Fissare un preciso punto d'inizio, è un buon modo per non dimenticare nulla; inoltre è molto difficile, soprattutto nei programmi che si dividono in più alternative, terminare con un unico punto. Questo si ottiene quando alla fine di ogni alternativa, si arriva ad un punto comune denominato come Fine.

L'uso di alternative con due uniche scelte, risponde più alle necessità della programmazione col PIC, determinata dal tipo di istruzioni, che al tipo di organigramma in sé. Quando qualcuno disegna il suo organigramma ha bene in mente la sequenza esatta che il programma dovrà eseguire, le alternative possibili, le domande importanti eccetera, quindi si tende a dare per scontato il verso delle frecce, e a non disegnarle correttamente. Però bisogna pensare che chi dovrà tradurre la "nostra" idea, si baserà unicamente sul disegno, e meno dati lasceremo alla libera interpretazione, migliore sarà il risultato.



Organigramma di un enunciato, realizzato in forma corretta e non corretta.

Il rettangolo di inizializzazione dei registri e definizioni delle variabili servirà da promemoria al momento di mettere in pratica la stesura del programma.

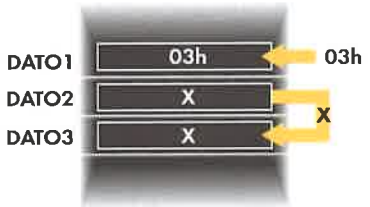
Seguendo queste considerazioni abbiamo realizzato un organigramma che rappresenta la richiesta dei dati per l'esecuzione di una somma, e la presentazione del risultato. Nelle figure in alto, vediamo un esempio di come non si deve rappresentare il medesimo programma.

Applicazione pratica: il primo programma

Anche se rimane ancora molta teoria da vedere in quanto a struttura e utilizzo del PIC, cominciamo con un primo programma, con il quale ci addentreremo a poco a poco nella programmazione e nel linguaggio assembler – che non è così complicato come tutti credono –, e più concretamente nell'assembler dei PIC. Cominceremo muovendo dati da un posto ad un altro. Leggiamo con attenzione l'enunciato dell'esercizio.

Nel grafico che accompagna l'enunciato, i rettangoli sono i registri entro i quali eseguire i movimenti dei dati, le frecce rappresentano il verso di questi movimenti; i dati all'interno appaiono nel posto che avranno dopo l'esecuzione del programma, indicato anche dalla punta delle frecce. Il valore 03 esadecimale è un valore letterale e la X rappresenta qualsiasi valore, visto che non conosciamo il valore iniziale del registro DATO2.

**Abbiamo tre posizioni che chiameremo DATO1, DATO2, DATO3.
In DATO1 si vuole immagazzinare il valore esadecimale 03,
e in DATO3 il valore che abbiamo in DATO2.**



Enunciato del programma da realizzare e la sua rappresentazione grafica.

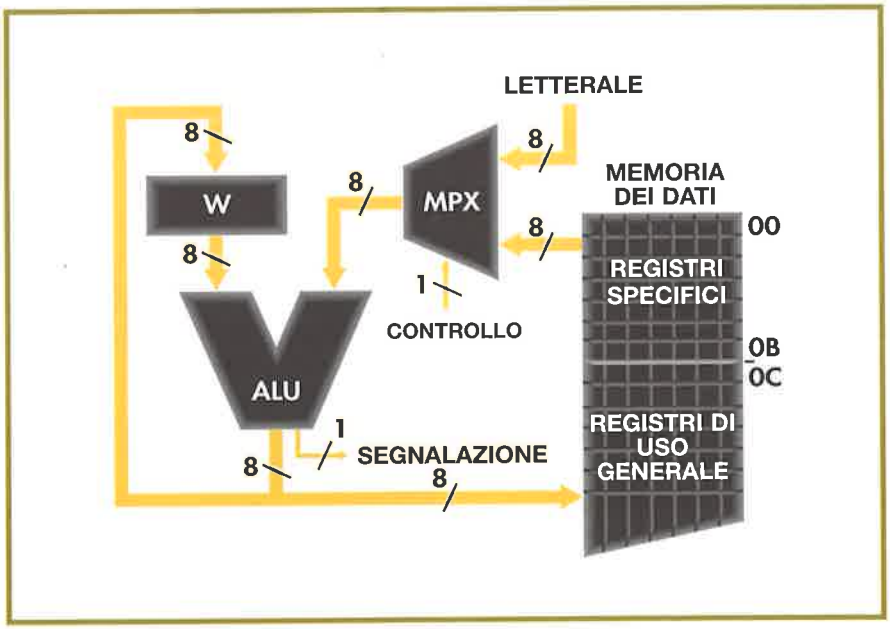
IL BLOCCO DI ELABORAZIONE DATI

Per poter realizzare il suo compito, il PIC ha al suo interno la struttura mostrata nella figura seguente. Nella memoria dei dati a partire dall'indirizzo esadecimale 0C possiamo immagazzinare i dati che ci interessano, per lavorare con essi come meglio ci conviene.

La ALU (Unità Aritmetico Logica) è incaricata di realizzare le operazioni aritmetiche e logiche, e all'interno di queste, le operazioni di movimento che sono quelle che ci interessano. W è un registro chiamato di lavoro perché interviene come appoggio alla memoria nella maggior parte delle operazioni che interessano la ALU.

Quando un'operazione aritmetico-logica utilizza due operandi, uno di questi è sempre il registro W, l'altro può essere sia un registro della memoria dei dati sia un valore letterale dato insieme al codice dell'istruzione.

La discriminazione tra i due è fatta grazie ad un multiplexor (MPX) la cui linea di controllo avrà un valore o l'altro a seconda dell'istruzione che sta eseguendo in quel momento. Il risultato dell'operazione può arrivare al registro W o alla memoria dei dati. Il numero 8 che



Percorso che seguono i dati quando si esegue un'istruzione.

• movlw	k		Muove il valore letterale k al registro di lavoro W
• movwf	f		Muove il valore contenuto in W al registro f
• movf	f ,	d	Muove il valore contenuto in f alla destinazione d, che può essere il medesimo registro f (d=1) o registro di lavoro W (d=0)

Tavola con le possibili istruzioni di movimento del PIC16F84.

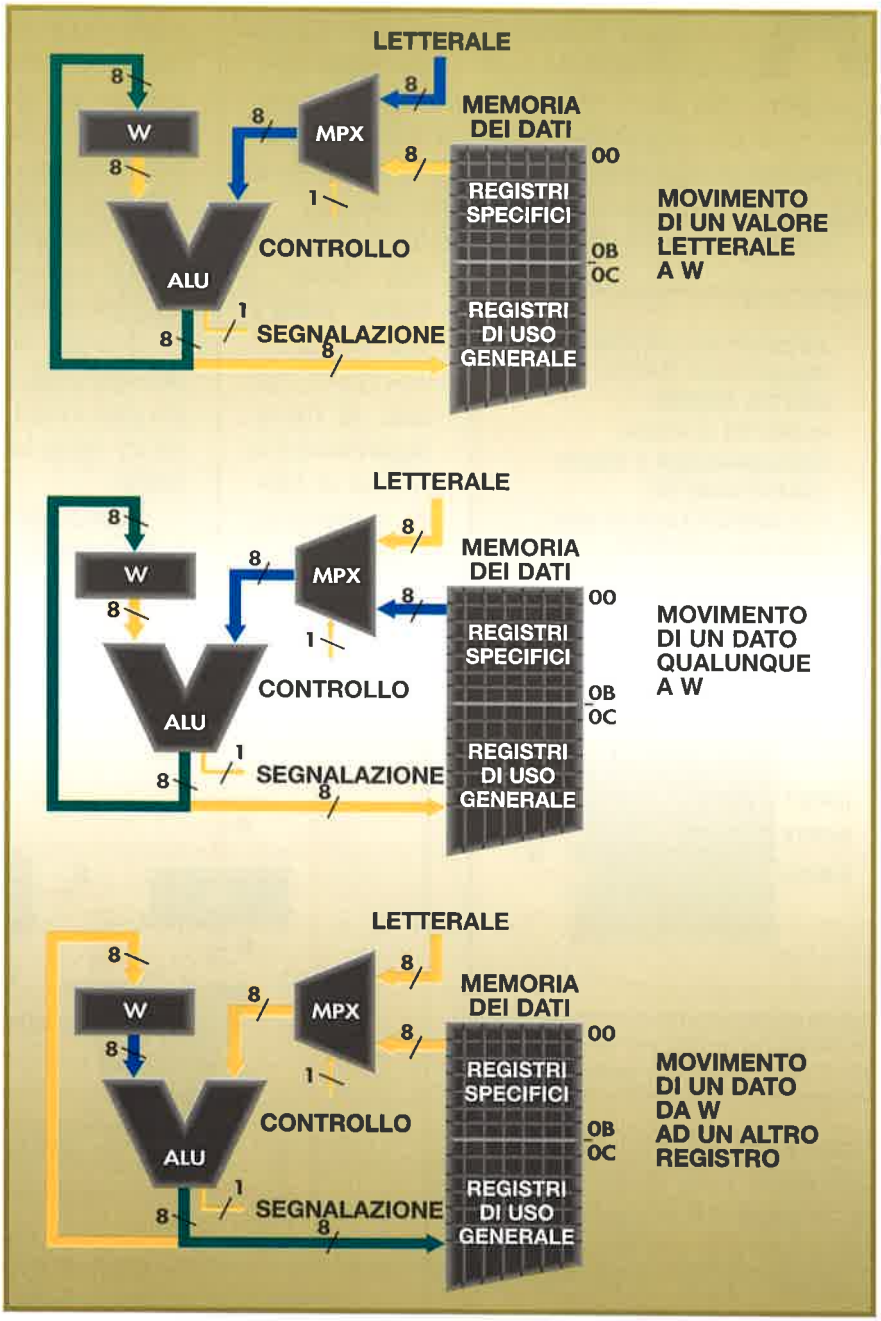
appare nel grafico ha un suo significato: indica che tutti i dati con i quali lavora il PIC 16F84 sono da 8 bit, sarebbe a dire che sia i registri che i dati con i quali opera la ALU, sia i risultati che genera, inclusi i dati che può contenere il registro di lavoro W, sono a 8 bit.

Allo stesso modo, se al posto di un 8 appare un 1, o qualunque altro numero, significa che il dato che passa per quella linea, è composto da quel numero di bit. La ALU, realizzando alcune operazioni, oltre ad un risultato, fornisce anche una segnalazione, di cui parleremo in seguito.

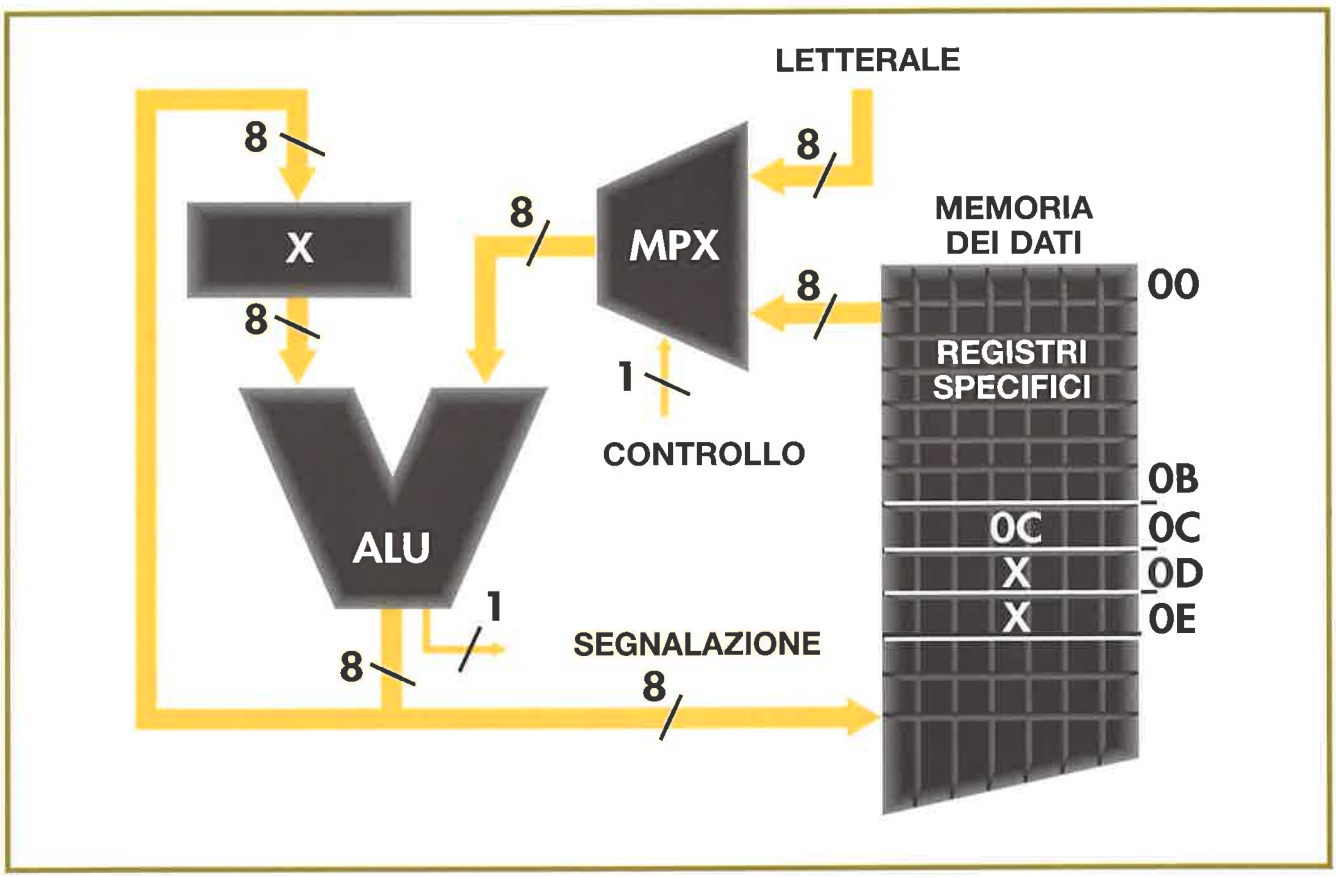
ISTRUZIONI DI MOVIMENTO

Nel PIC esistono tre istruzioni per muovere i dati. Tutte cominciano per mov e contengono a continuazione altre lettere per indicare da dove a dove si muovono i dati. Così si può muovere un valore letterale al registro di lavoro W, o muovere un valore dal registro di lavoro ad un registro della memoria dei dati o viceversa, muovere un valore dal registro della memoria al registro di lavoro W, o da W a se stesso.

Le tre istruzioni che appaiono nella figura saranno trattate in una scheda individuale più avanti, però vi anticipiamo che per tutte le istruzioni, la parte sinistra del quadro è quella che ci dice il tipo di istruzione che si sta eseguendo, quella che viene dopo e gli operandi che intervengono in questa operazione; generalmente si



Percorso dei dati, con differenti istruzioni in esecuzione.



Blocco di elaborazione dati dopo aver eseguito il programma.

rappresenta sempre con una lettera legata al tipo di istruzione.

Possiamo seguire il flusso percorso dai dati nella figura della pagina precedente. La prima rappresenta il movimento di un valore letterale a un registro, che è la parte del nostro compito nell'enunciato numero 1, muovere un valore letterale a W. Per portarlo poi da W al registro DATO1, dobbiamo seguire il cammino segnato dalla terza figura. Nella seconda parte dell'enunciato del programma dobbiamo muovere un dato da un registro ad un altro.

Anche se non esiste alcuna istruzione per muovere un dato all'interno dei registri, questo si può fare con la combinazione di due istruzioni: muovendo prima un dato da un registro qualunque a W, e dopo da W al secondo registro. Questo viene illustrato nella seconda e terza figura. Dobbiamo considerare che il registro di lavoro W è l'unico che non fa parte della memoria dei dati, per cui non viene trattato come gli altri, e per utilizzarlo bisogna fare riferimento ad esso tramite il nome dell'istruzione.

PROGRAMMA ASSEMBLER

Il programma assembler risultante è illustrato nella pagina seguente: nella prima linea con la direttiva LIST, si indica il tipo di PIC che si vuole utilizzare, nel nostro caso il 16F84. Nel caso in cui volessimo utilizzarne un altro, cambieremo il parametro che segue P=. È obbligatorio usare questa direttiva, perché tutti i nostri programmi cominceranno nella medesima forma.

Nelle linee 2, 3 e 4 si usa la direttiva EQU per definire le variabili. Nell'esercizio si chiede di muovere dati nelle differenti posizioni di memoria, senza specificare quali, la prima posizione a disposizione dell'utilizzatore è la posizione OC. Per questo si hanno codici a tre posizioni consecutive, a partire da 0C, al posto di lavorare direttamente con gli indirizzi, la direttiva EQU ci permette di dar loro un nome per far sì che l'uso sia più significativo.

La linea 5 è di nuovo un'istruzione, la ORG. Questa direttiva dice all'assembler che l'istruzione che seguirà, (nel nostro caso la movlw 03) è messa nella posizione di memoria che è indicata dal parametro (nel nostro

```
1                LIST      P=16F84

2       DATO 1   EQU       0C
3       DATO 2   EQU       0D
4       DATO 3   EQU       0E

5                ORG       0

6                movlw     03
7                movwf     DATO 1
8                movf      DATO 2.0
9                movwf     DATO 3

10               END                  ; Fine del programma
```

Programma assembler dell'enunciato proposto.

caso 0). Questa istruzione, inoltre, è sempre obbligatoria, dato che è la prima istruzione del programma (fino alla movlw non sono istruzioni) deve essere nella posizione 0 della memoria di programma, dato che è lì che viene indirizzato il puntatore al reset del PIC.

Comincia così il programma in sé. Nella linea 6, si muove il valore letterale 03 a W per proseguire col valore al registro DATO1, così come abbiamo detto nell'enunciato. Per muovere il dato contenuto in DATO2 al registro DATO3 si usa come registro intermedio W (istruzioni 8 e 9).

Nella linea 10 si termina il programma con l'istruzione END.

Questa istruzione è sempre obbligatoria.

Una volta eseguito il programma il blocco di elaborazione dati del PIC resterà come segue, tenendo conto della posizione di memoria che gli abbiamo assegnato.

NORME PER SCRIVERE UN PROGRAMMA

Molti lettori si staranno chiedendo perché alcune cose sono sul margine sinistro e altre sono tabulate, se i cambi tra maiuscole e minuscole hanno significato eccetera. Ecco qui delle norme, alcune obbligatorie, altre solo come raccomandazioni, da seguire al momento di scrivere un programma per il PIC.

- **Le etichette e i nomi di variabili (LIST, DATO1) di solito si scrivono in maiuscolo. Non succede niente se vengono scritte in minuscolo, però così si distinguono meglio dalle istruzioni. Dobbiamo ricordarci di scrivere le etichette in forma sempre uguale, dato che l'assembler è sensibile a maiuscole e minuscole.**
- **I nomi (mnemonici) delle istruzioni si scrivono in minuscolo. Non accade nulla nello scriverle in maiuscolo, però si stanca di più la vista e si fa più fatica a leggerli.**
- **I programmi devono essere tabulati e ben allineati per vedere rapidamente la struttura del programma stesso. Le uniche cose che si scrivono nel margine sinistro sono le definizioni delle variabili (DATO1, EQU, OC) e le etichette (che vedremo in seguito). Aiutano nella presentazione di questa struttura gli spazi bianchi che si lasciano tra le parti del programma.**
- **L'utilizzo di commenti per spiegare le istruzioni o parte di queste, aiuta nell'elaborazione e nella successiva modifica o riutilizzazione di un programma. Per questo si utilizza il punto e virgola (;) e in seguito il commento che si intende inserire (; Fine del programma).**

Norme di scrittura del programma in assembler per il PIC.

Schede delle istruzioni: concetti generali

Poco a poco vi mostreremo le differenti istruzioni che formano il repertorio del PIC16F84. Per questo è necessario chiarire prima una serie di termini, che saranno quelli che utilizzeremo d'ora in poi nel corso dell'opera.

Esempio di alcuni dei termini spiegati.

Mnemonico:
è il nome dell'istruzione che noi useremo per ordinare al microcontroller cosa deve fare. Dobbiamo scriverlo tale quale senza cambiarlo.

Parametri:
operandi che necessita l'istruzione per essere eseguita. Per entrare nei dettagli di un'istruzione si specificano i parametri che essa ammette che dovranno essere poi sostituiti da valori reali.

Parametro **f:** rappresenta un registro della memoria dei dati, operando originato da un'istruzione.

Parametro **d:** rappresenta un registro, operando destinatario di un'operazione, con due possibili valori.
Se si sostituisce con uno 0 significa che il registro di destinazione è il registro di lavoro W o accumulatore e se è sostituito da un 1 significa che è il medesimo operando originato dall'istruzione.

Parametro **b:** rappresenta un valore (da 0 a 7) di un registro.

Parametro **k:** rappresenta un valore letterale, sarebbe a dire, un valore riferito al mnemonico dell'istruzione.

Registro di lavoro o accumulatore W:
registro speciale che non fa parte della memoria dei dati, e che si utilizza nella maggior parte delle operazioni aritmetico-logiche.

Ciclo di istruzione:
è il tempo base che impiega un'istruzione ad essere eseguita. Per il PIC16F84, funzionante a 4 MHz, questo ciclo vale 1 microsecondo (µs).

Unità Aritmetico-Logica:
è incaricata di realizzare le operazioni aritmetiche e logiche del PIC.

Blocco di elaborazione dati:
insieme dei registri che eseguono le istruzioni.

Flags:
bits contenuti in determinati registri che servono come segnalatori di eventi nell'esecuzione delle operazioni.

Codice OP:
traduzione di un'istruzione realizzata dall'assembler, perché possa essere compresa dal PIC, che non lavora con codici mnemonici, ma con 1 e 0. La longitudine dei codici OP in un PIC16F84 è di 14 bits.

Termini utilizzati parlando di istruzioni.

Schede delle istruzioni: l'istruzione movlw

L'istruzione "mov" è tipica in tutti i processori, dato che è sempre necessario il movimento di dati da un posto ad un altro, siano questi dei risultati, valori fra registri o valori immediati.

Consta di due operandi, dove il dato contenuto nel primo è trasferito, o spostato al secondo. Con questa istruzione si realizza una copia dei dati, poiché non si cancella il dato originale.

Ci sono tre tipi di istruzioni "mov", le quali si differenziano nei loro operandi di origine e destinazione. Nel caso della movlw, l'origine è un valore letterale e la destinazione è il registro di lavoro W.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
m o v l w	k	

OPERAZIONE: Movimento del valore letterale **k** al registro di lavoro **W**

CICLI: 1

CODICE OP: 11 00xx kkkkk kkkk

FLAGS: Nessuno

Caratteristiche dell'istruzione movlw.

ESEMPI CON L'ISTRUZIONE MOVLW

Questa istruzione è stata utilizzata precedentemente nella realizzazione di un programma, dove abbiamo visto che la programmazione in assembler è più facile di quel che appare. Ora vedremo quello che succede in questo esempio passo a passo.

Si inizia muovendo il valore letterale 5 nel registro di lavoro W e si prosegue muovendo il valore letterale 3 al medesimo registro W.

La figura a lato rappresenta il valore che assume W dopo l'esecuzione di ognuna delle istruzioni.

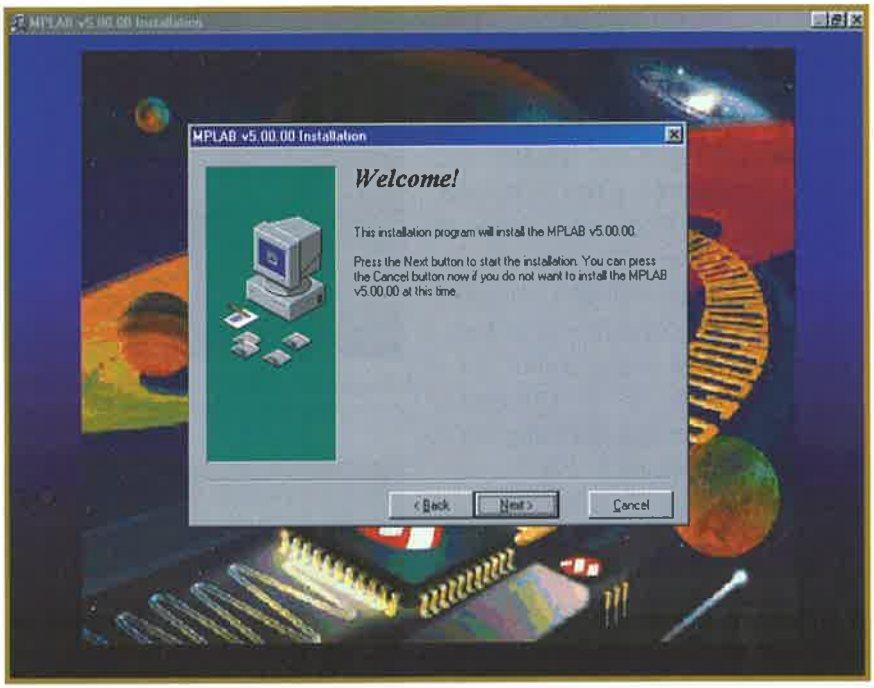
All'inizio il valore di W sarà indeterminato, e quindi si rappresenta mediante "x". Dopo la prima istruzione avremo caricato in W il valore 5, il cui codice binario si rappresenta con "101". Dato che i registri del PIC hanno una dimensione di 8 unità (bits), le celle non direttamente interessate, sono messe a zero, iniziando da sinistra. Dopo la seconda istruzione il valore di W varia, cancellando il primo valore e passando ad essere "00000011".

valore iniziale	xxxxxxxx	W
movlw 5	00000101	W
movlw 3	00000011	W

Valore di W all'interno di ogni istruzione movlw.

Strumenti di lavoro: l'ambiente MPLAB

L'MPLAB è un ambiente di sviluppo integrato per il lavoro con la famiglia PIC16/17 di Microchip. Essere integrato significa che dal suo interno si può editare un programma, compilarlo, tradurlo in linguaggio per il microcontroller, simularne il suo funzionamento, emularlo su un hardware, che abbia le funzioni del sistema reale, e anche scriverlo nel microcontroller, anche se per queste due ultime possibilità, è necessario disporre di un hardware adeguato, oltre all'applicazione.



Finestra di inizio dell'installazione di MPLAB.

INSTALLAZIONE DI MPLAB

L'ambiente MPLAB si trova in una cartella con lo stesso nome nel CD di Robots, che vi abbiamo fornito. Per eseguire l'installazione accederemo a questa cartella, dove troveremo un file che dovremo eseguire.

Facendo questo si aprirà una finestra, come quella della figura, che ci comunicherà che stiamo per procedere all'installazione di MPLAB.

Accettando (Next) apparirà una seconda finestra per scegliere quali componenti vogliamo installare.

I componenti che selezioneremo corrispondono ai programmi, ai files e alle loro funzioni relative che utilizzeremo:

MPLAB IDE Files: sono i files relativi all'ambiente di sviluppo in

Finestra di selezione dei componenti da installare.

sé, a cui si fa riferimento con la sigla IDE (Integrated Development Environment).

MPASM/MPLINK/MPLIB Files: sono il compilatore, i linker e i files di libreria accessibili tramite MPLAB. Il compilatore sarà necessario per tradurre i programmi che editeremo in linguaggio macchina, in questo caso quello del microcontroller. Se il programma è realizzato in diversi files tra loro relazionati, ed ognuno di essi contiene vari sottoprogrammi, sarà necessario il linker per renderli una sola unità coerente. I files di libreria, dal canto loro, facilitano il lavoro di programmazione fornendo "frammenti" di programmi già elaborati e molto utilizzati che potremo includere all'interno del nostro codice.

MPLAB-SIM: è il simulatore software che fornisce Microchip. Con esso si può verificare il corretto funzionamento delle istruzioni del programma che si sta eseguendo, vedere come risponde agli ingressi del sistema, così come le uscite esterne che questo genera.

MPLAB-ICD Debugger Support Files: come indica il suo nome non si tratta di un'applicazione in sé, ma di files di supporto necessari per la funzione di "debug" o ricerca guasti. Questa funzione permette, non solo di vedere i risultati finali di un programma, come farebbe l'utilizzatore, ma di accedere ai registri del sistema mentre si sta eseguendo l'applicazione, eseguendo le istruzioni passo a passo eccetera. Tutto questo lo vedremo in maniera più dettagliata trattando questi strumenti così importanti.

Help Files: sono i files che ci permetteranno di ottenere aiuto dalle funzioni, menù, finestre eccetera, che integrano MPLAB.

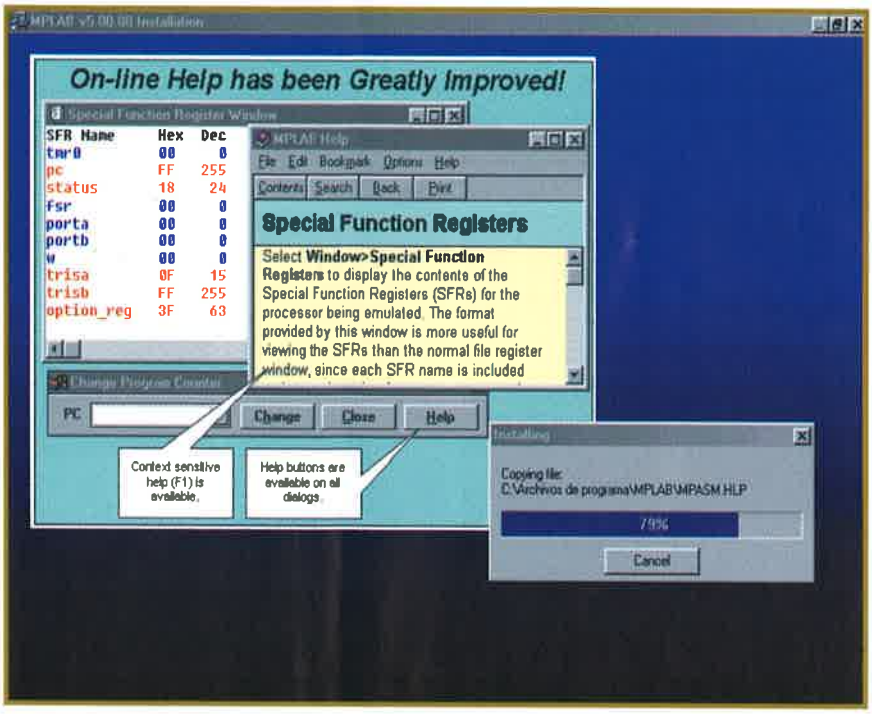
Finestra di selezione dell'ambiente dei componenti.

Finestra di selezione dell'ubicazione del linker.

I restanti componenti danno supporto software a strumenti hardware per emulare e scrivere i programmi commercializzati dalla stessa ditta Microchip, dei quali non disponiamo, per cui al momento non parleremo di loro. La programmazione del PIC nei programmi che realizzeremo si farà con il software e l'hardware specifico in quest'opera.

Una volta scelti i componenti, continuando (Next) apparirà una seconda finestra per selezionare l'ambiente nel quale si eseguiranno i programmi. Noi abbiamo scelto l'ambiente Windows perché in questo momento è il più diffuso sul mercato.

In seguito verrà chiesto un indirizzo per l'installazione. Si può scegliere quello proposto, oppure uno che vi sembra più adeguato all'organizzazione del nostro disco fisso, dato che questo non influirà in seguito sull'esecuzione dell'applicazione. Nella seguente finestra è offerta l'opzione di fare una copia di sicurezza dei files già esistenti sul nostro computer che saranno sostituiti, e in seguito introdurre nel Menù di Inizio di Windows l'accesso rapido all'applicazione in modo che non sia necessario entrare nelle sottocartelle o nei files per ricercarla.



Durante l'installazione sono mostrate una serie di caratteristiche dell'applicazione.

Quando appare la finestra per scegliere l'indirizzo del linker, dovremo selezionare una delle due possibili opzioni. Se avremo installato una versione precedente di MPLAB, la scelta sarà fatta di conseguenza, per rendere compatibili i vecchi progetti con quelli nuovi. Comunque per un utilizzatore nuovo si raccomanda di installare il linker in MPLAB\Lkr.

In seguito dovremo scegliere dove installare gli archivi di sistema. Se lavoreremo in Rete le scelte verranno fatte di conseguenza.

Dopo tutte queste scelte inizia l'installazione di MPLAB. Durante il processo di installazione appariranno una serie di caratteristiche nel video, oltre alla barra di stato dell'installazione.

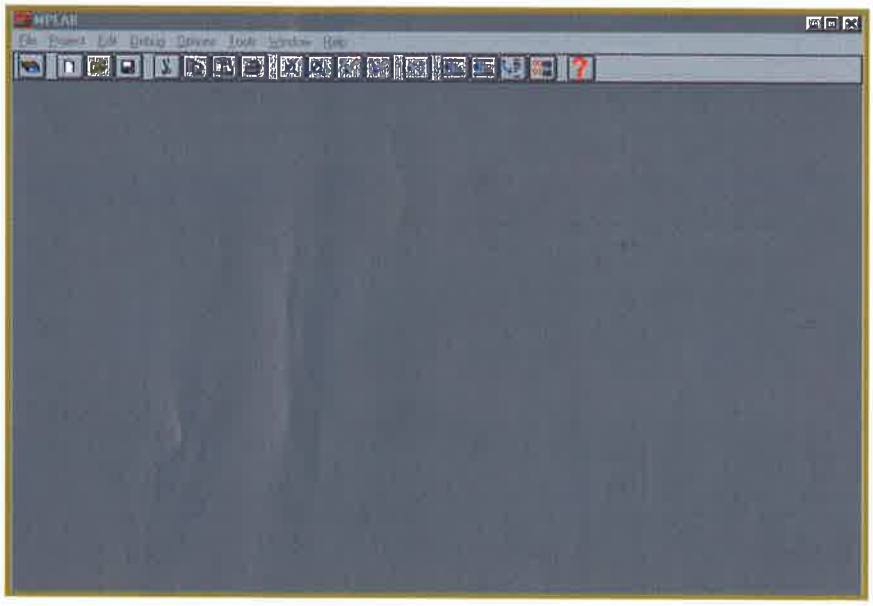
Una volta terminata l'installazione, si potranno leggere i files di testo associati ai componenti installati, oppure lasciarli per la fase successiva.

Tutti questi passi del programma MPLAB si trovano nella cartella "C:\Archivio di programma\Mplab".

FUNZIONI DI MPLAB

Possiamo già eseguire un programma, aprendolo apparirà una finestra come quella della figura riprodotta qui in basso.

Come si può osservare l'aspetto è come quello

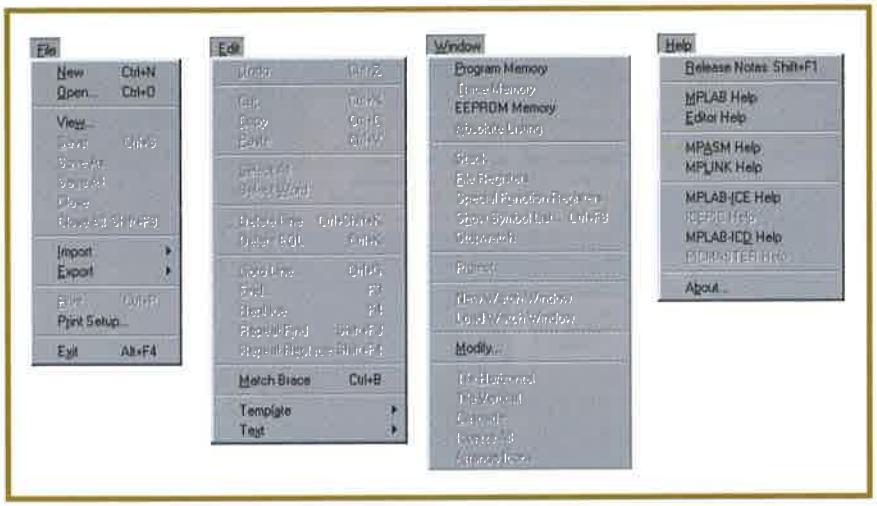


Finestra di inizio di MPLAB alla sua esecuzione.

della maggior parte dei programmi che funzionano nell'ambiente Windows. Nella parte superiore della finestra appariranno diversi menù dentro i quali incontreremo i tipici "Files", "Modifica", "Finestra", "Aiuto" e altri che, invece, sono specifici di questa particolare applicazione.

Sotto i menù le icone che rappresentano le funzioni più utilizzate. Se si seleziona l'icona di sinistra il gruppo di icone varierà.

All'interno di questo ambiente si potrà:

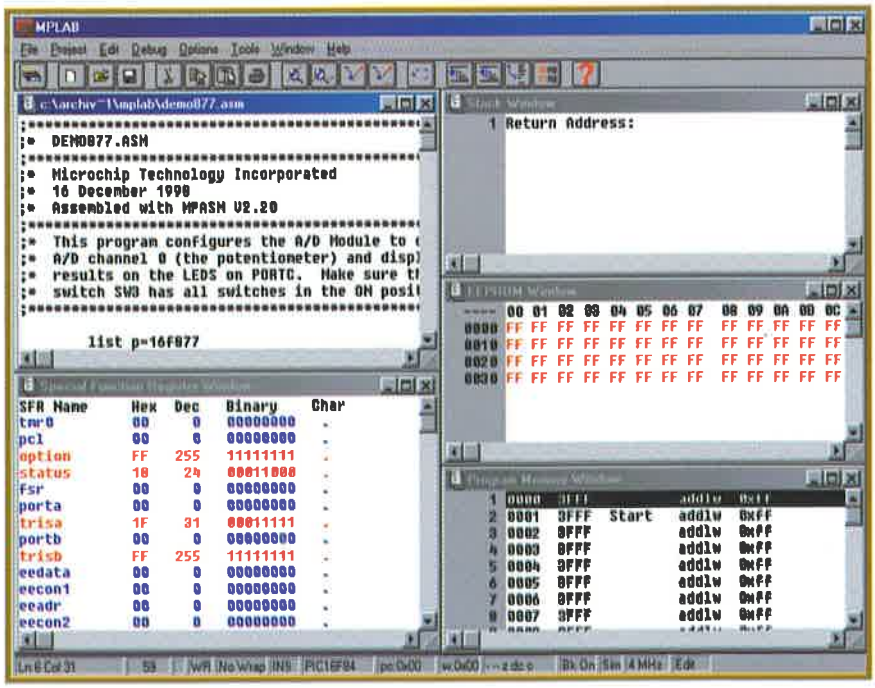


In Assembler ogni istruzione si traduce come un solo codice macchina.

Gruppi di icone che rappresentano le funzioni più comuni.

- **Organizzare codici di una applicazione in progetto.**
- **Editare i programmi fonte per il PIC 16/17 così come altri files di testo di cui potremo aver bisogno.**
- **Compilare i codici fonte ottenendo i files da memorizzare sul microcontroller.**
- **Simulare i codici una volta compilati e prima di memorizzarli nel microcontroller, verificare che il lavoro realizzato sia adeguato.**

Tabella delle funzioni da realizzare nell'ambiente MPLAB.



Finestra che mostra i differenti tipi di informazioni fornite da MPLAB.

Poco a poco vedremo come si realizzano queste opzioni. Per il momento diamo un'occhiata alla seguente figura per verificare il tipo di informazione che ci permette di vedere l'applicazione, e cominciamo a giocare un po' con i diversi menù.

La finestra superiore a sinistra mostra l'edizione del programma in corso. Sopra di essa, in un'altra finestra, si può vedere il valore di ognuno dei registri specifici del PIC nel loro formato esadecimale, decimale e binario. Nelle finestre di destra, cominciando dall'alto verso il basso, troveremo la finestra dello Stack delle istruzioni del PIC, la EEPROM dei dati e la memoria di programma.

Schede delle istruzioni: l'istruzione movwf

È la seconda variante dell'istruzione "mov". In essa il dato contenuto nel registro di lavoro W è trasferito al registro f, che qui appare come parametro, in modo che si cancelli da W.

Dobbiamo solo indicare qual è l'operando di destinazione, visto che quello d'origine, W, è implicito nel codice di istruzione.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
m o v w f		f

Operazione: movimento del contenuto del registro di lavoro **W** al registro **f**
Cicli: 1
Codice OP: 00 0000 1 fff ffff
Flags: nessuno

Caratteristiche dell'istruzione movwf.

ESEMPI CON L'ISTRUZIONE MOVWF

Questa volta vogliamo caricare il valore del registro il cui indirizzo è 0E (ricordiamoci che a partire da 0C disponiamo dei registri generali che possiamo utilizzare come ci conviene), e che chiameremo OPERANDO, con valore decimale 8. Per questo dovremo impiegare le due istruzioni che già conosciamo: movlw e movwf.

Nella figura si vedono i passi da realizzare.

	OPERANDO	W
valore iniziale	xxxxxxxx	yyyyyyyy
movlw di 8	xxxxxxxx	00001000
movwf OPERANDO	00001000	00001000

Passi da realizzare per caricare il registro OPERANDO con il valore decimale 8.

I registri, all'inizio, si considerano con valore indeterminato, X e Y. Notiamo che al momento di utilizzare le istruzioni il valore letterale non viene sempre rappresentato nello stesso modo; dipenderà dalle situazioni o dall'utilizzatore, comunque al momento di memorizzare i dati nei registri e quando lavora la ALU la base utilizzata sarà la binaria. Il codice esadecimale è quello che si usa normalmente, in assenza di altre specifiche, e comprende i valori da 0 alla f. Per utilizzare gli altri codici dovremo esprimere il numero entro "virgole semplici" e precederlo da una lettera che indichi il codice in cui si trova. In questo modo, avremo diversi codici molto impiegati, come ad esempio il decimale e il binario. Dopo le operazioni, entrambi i registri hanno il medesimo dato, visto che questo non viene cancellato nel registro di origine.

Schede delle istruzioni: l'istruzione movf

È la terza e ultima variante dell'istruzione "mov". In questo caso il contenuto del registro f, che è l'operando fonte, si può trasferire nel registro di lavoro W, o al medesimo registro f, a seconda del valore di "d". È necessario specificare tanto l'origine come la destinazione.

Muovere un dato da un registro al registro di lavoro W ha senso, dato che l'obiettivo può essere trasferirlo ad un secondo registro a partire da W, o operare nel medesimo registro di lavoro, però muovere un dato da un registro allo stesso registro, sembra non avere senso. Tutto ha una ragione, ed è che quando si realizza questa operazione resta coinvolto un segnalatore, o flag, chiamato Z. Il flag Z è un bit del registro di STATO che viene messo a 1 se il risultato di una operazione è 0. Così, muovendo un registro a sé medesimo e verificando il valore del flag Z, possiamo sapere se questo registro ha valore 0, o no, visto che Z sarà messo a 1 nel caso che questo valga 0. Ciò è molto utile, e si vedrà, parlando di cicli, che per ripetere un'azione un determinato numero di volte, si può caricare il valore in un registro e decrementarlo fino a che questo raggiunga 0.

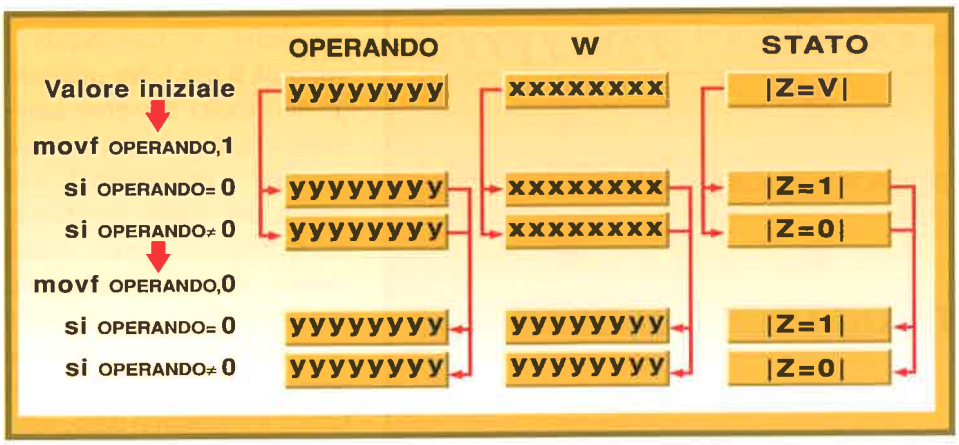
MNEMONICO	OPERANDO FONTE	OPERANDO DI DESTINAZIONE
m o v f	f	d

Operazione: movimento del contenuto del registro f alla destinazione d. Se d = 0 si muove il registro di lavoro **W** e se d = 1 si muove allo stesso registro **f**
Cicli: 1
Codice OP: 00 1000 dfff ffff
Flags: Z

Caratteristiche dell'istruzione movf.

ESEMPI CON L'ISTRUZIONE MOVF

Nella figura sono rappresentati i valori che prenderanno i registri all'esecuzione dell'istruzione movf nei due diversi casi, dipendendo dal valore degli operandi di destinazione. Questo ci darà due combinazioni, però se uniamo il valore del flag Z otterremo quattro casi distinti. Nei primi due avremo mosso il registro su se stesso e Z si sarà attivato, o no, a seconda che il valore del registro sia 0 nel momento del movimento. Negli ultimi due esempi, oltre all'attivazione del flag Z, il registro di lavoro W ha alla fine dell'istruzione, il medesimo valore dell'operando fonte.

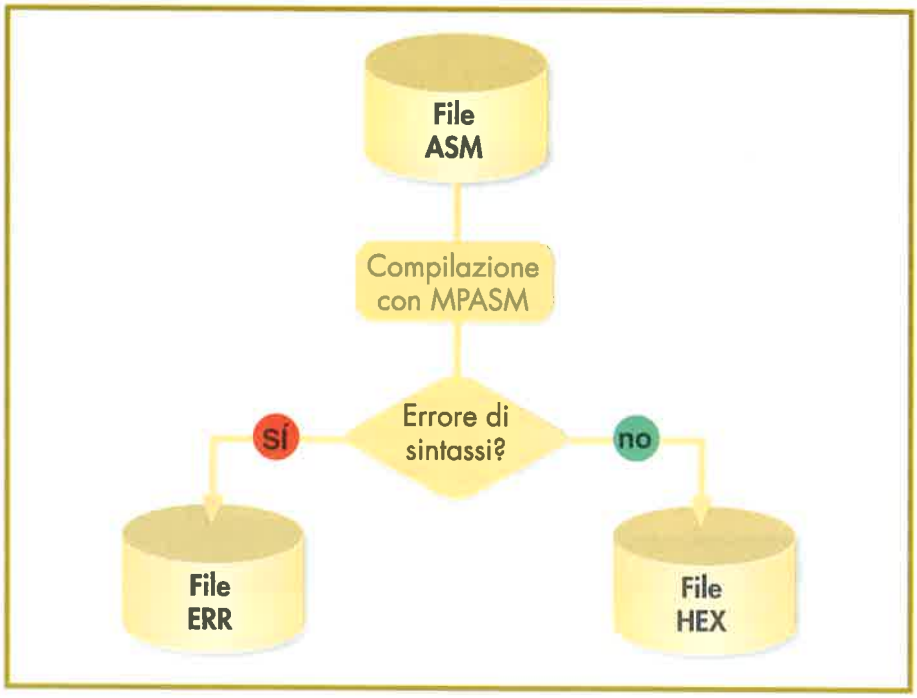


Valori dei registri nei differenti casi con esecuzione dell'istruzione movf.

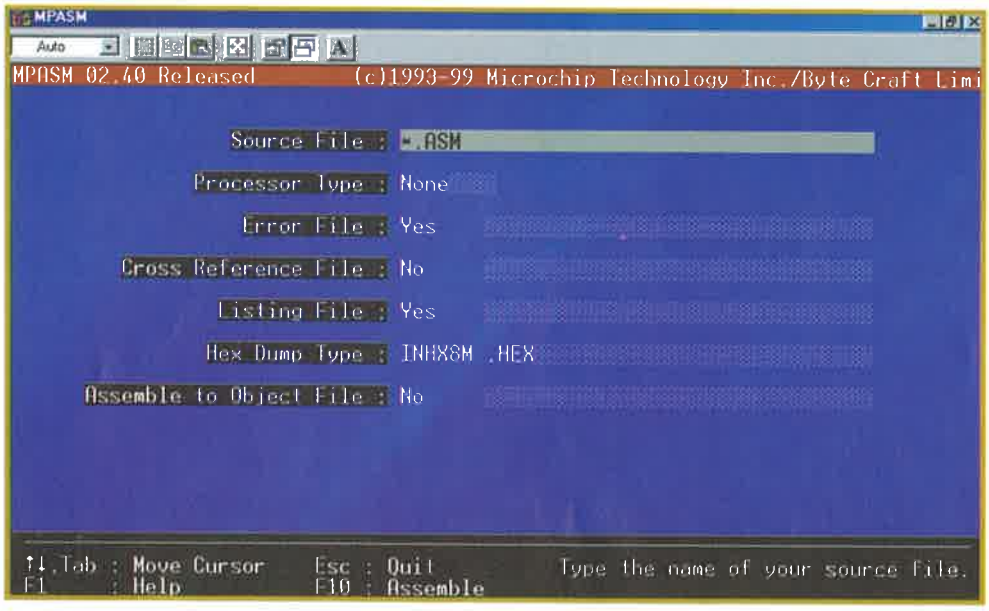
Strumenti di lavoro: il compilatore MPASM

L'MPASM è un compilatore per PIC. Delle tre versioni che esistono attualmente, noi utilizzeremo quella per Windows, che abbiamo già caricato installando MPLAB. Questo offre il vantaggio di poter compilare senza uscire dall'ambiente di sviluppo, cosa che risulta molto comoda, oltre a fornire un'interfaccia molto più gradevole rispetto alla versione per MS-DOS.

L'MPASM deve lavorare con file con estensione ASM, li compilerà e, se non sono stati commessi errori, darà come risultato altri files, questa volta con estensione HEX. Il codice HEX è quello che potrà essere programmato nel PIC per essere eseguito. Se il programma contiene errori sintattici, ad esempio se è stata scritta male un'istruzione, o non vengono rispettate le norme del compilatore, al posto del file esadecimale verrà prodotto un file di errore ERR, che sarà possibile leggere e nel quale saranno contenuti i codici degli errori trovati che dovremo correggere prima di proseguire. Lo schema della figura riprodotto in alto, mostra la sequenza seguita e i files coinvolti; questo schema anche se ha forma simile non deve essere confuso con un



Files più importanti in relazione all'MPASM.



La versione di MPASM per MS-DOS ha questo aspetto.

organigramma, perché in questo nuovo schema sono mescolati differenti tipi di dati.

I files mostrati nella figura sono i più importanti, ma non gli unici, dato che vengono generati anche altri tipi di file che per il momento non menzioneremo.

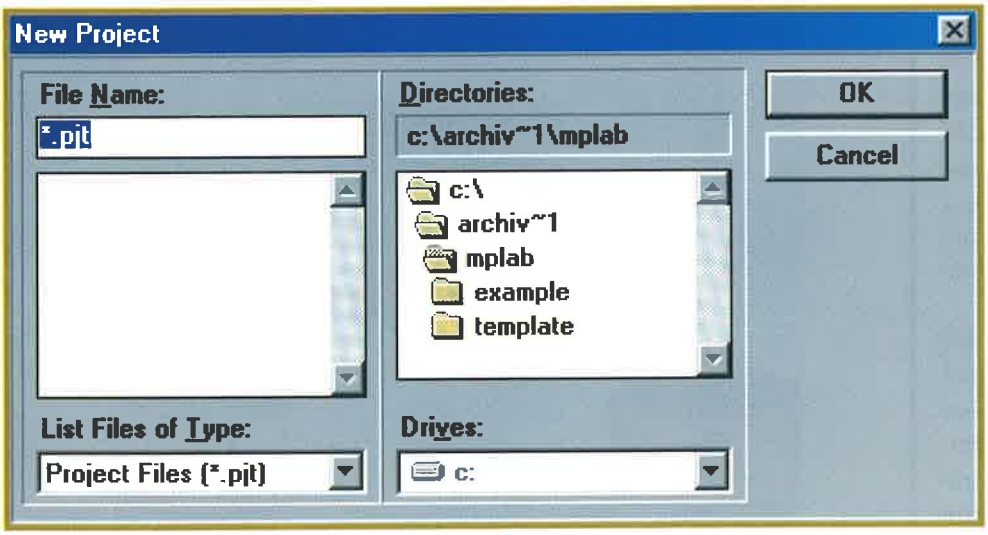
Ovviamente le finestre che illustrano questo capitolo di "Programmazione" non le troverete sul CD, ma servono unicamente come esempio di procedure da seguire per realizzare il vostro progetto.

CONFIGURAZIONE DELL'MPASM

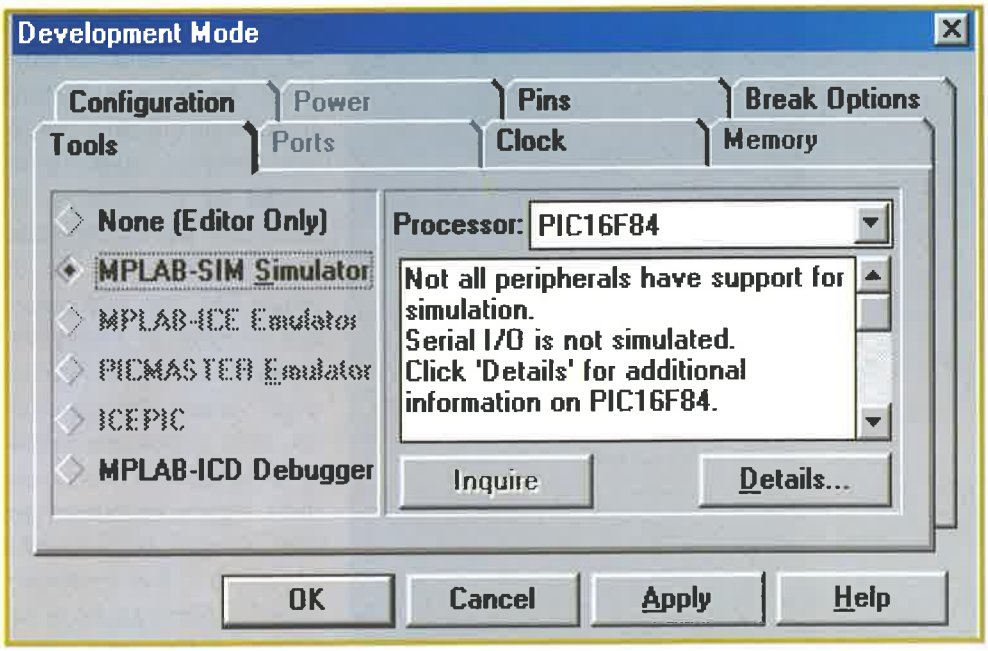
L'MPASM che utilizzeremo è parte dell'ambiente di sviluppo di MPLAB, per cui parlando del primo dobbiamo continuamente far riferimento a quest'ultimo. Così, per esempio, in MPLAB tutto è basato sui progetti, i files si chiamano nodi (nodes), e sono raggruppati per formare il progetto, così MPASM si dovrà configurare in base alle esigenze di ogni progetto.

Per configurare MPASM per poterlo utilizzare in MPLAB bisogna seguire i seguenti passi:

1. Si creerà un nuovo progetto (Project>New Project), dopo di che apparirà una finestra di dialogo come quella nella figura.



Finestra di dialogo che appare quando si crea un nuovo progetto.



Finestra per il cambio di ambiente da editazione a simulazione.

2. Si dovrà riempire il campo indicato con "nome file" (File Name) con il nome che vogliamo assegnare al progetto (ad es. "prova") e cliccare OK. Apparirà una seconda finestra con le opzioni.

3. All'inizio il modo di sviluppo (Development Mode) sarà di sola scrittura (Editor Only 16F84). Per il momento è sufficiente, però possiamo ugualmente provare a cambiarlo, con il pulsante "Change", selezioniamo il modo simulazione (MPLAB-SIM Simulator), osservando quanti parametri dovremo tenere in considerazione al momento di simulare il nostro programma.

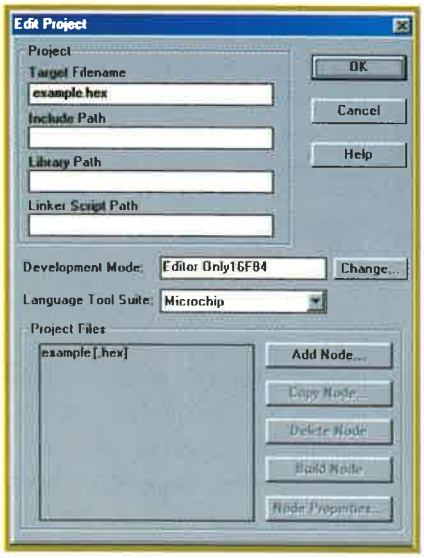
4. Torniamo però alla finestra delle opzioni del progetto: nella parte inferiore della finestra troviamo attivo il pulsante per aggiungere un nodo (Add Node), cliccandolo farà apparire una finestra che ci permetterà di realizzare questa operazione.

Una volta selezionato il file, cliccare su "OK". Nel

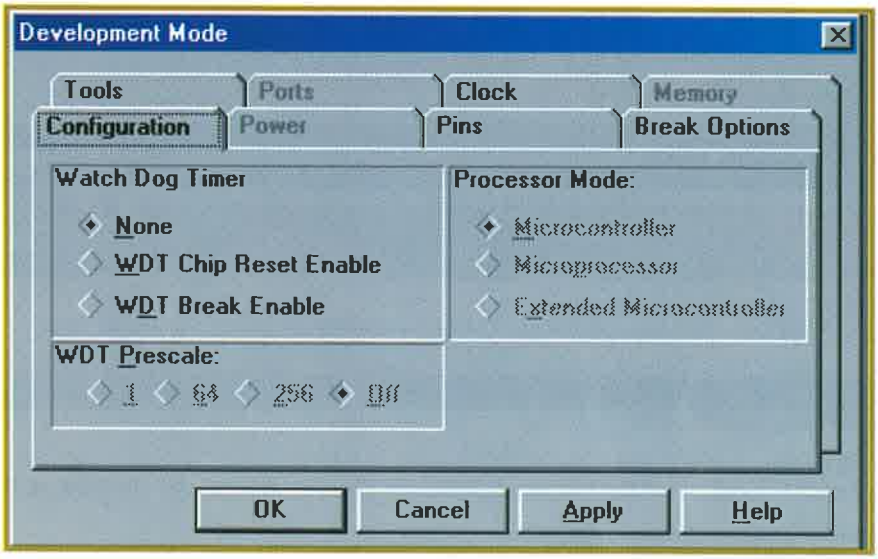
caso che nessuno dei files già editati faccia al caso nostro, ne possiamo creare uno nuovo con l'opzione File>New.

5. Se si seleziona il nome del progetto, (prova.hex), del riquadro Project File, si attiverà anche il pulsante di proprietà del nodo (Node Properties), che, di conseguenza dovremo cliccare.

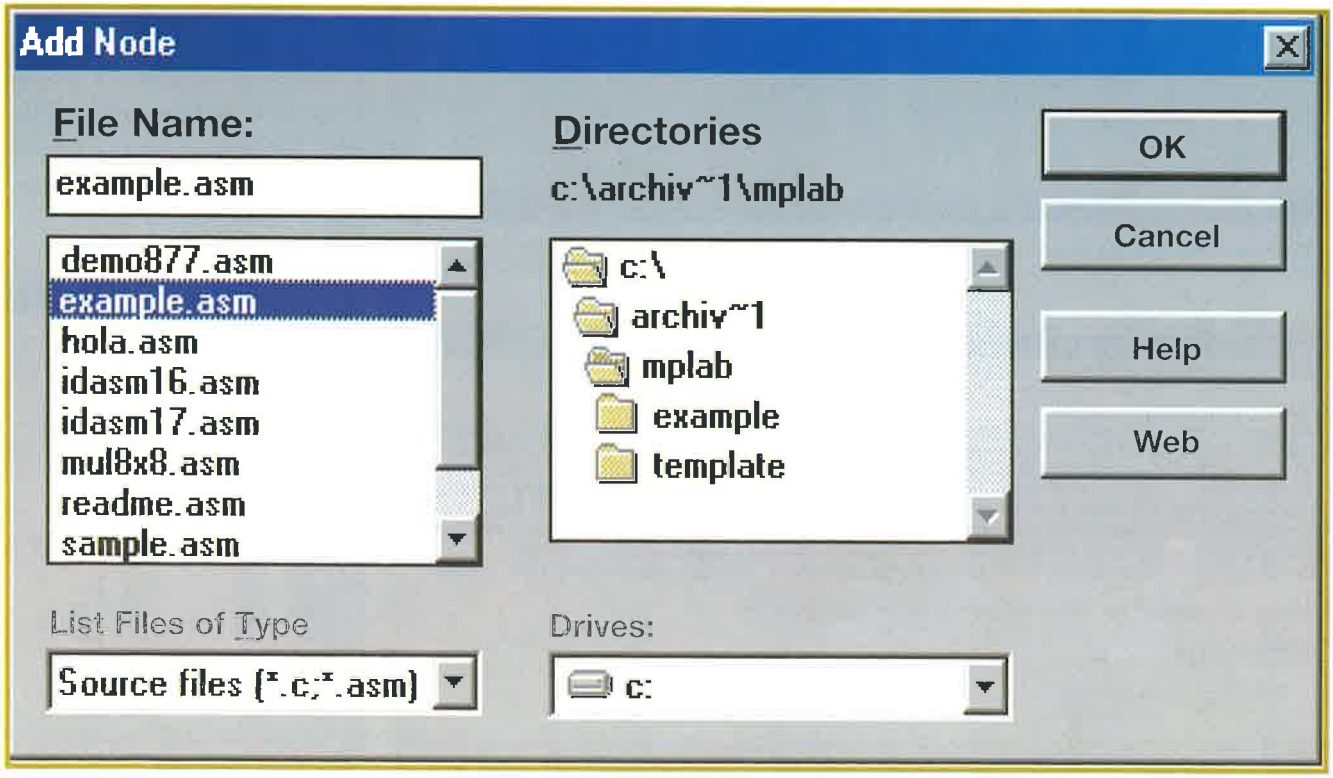
6. In questa finestra la cosa più importante è selezionare MPASM come linguaggio di programmazione (Language Tool), all'interno delle possibili scelte. Fatto questo, cliccheremo OK. Tornerà la finestra delle opzioni del progetto.



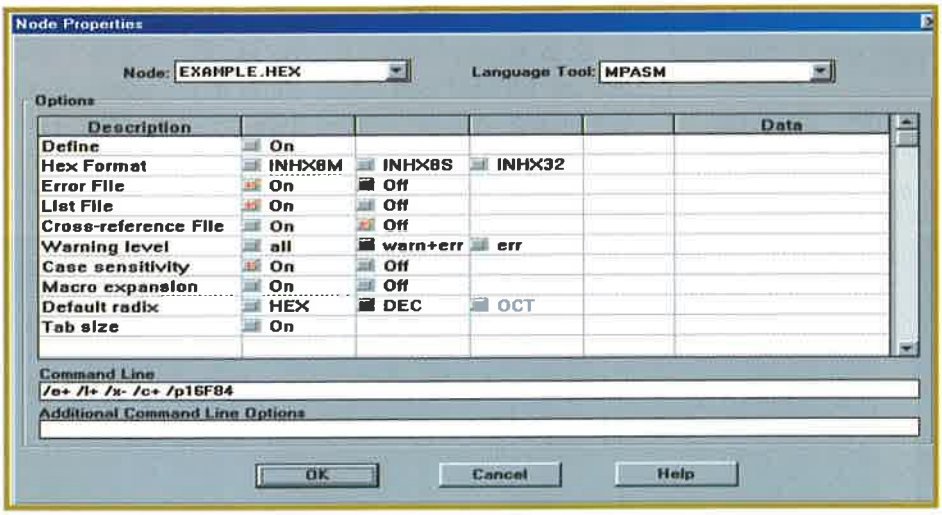
Finestra per inserire le opzioni del progetto.



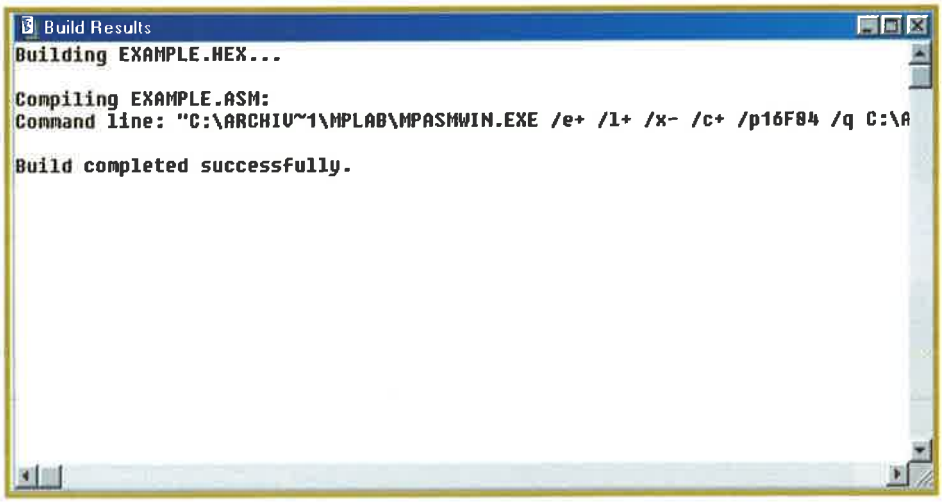
Alcuni parametri da tenere in considerazione quando si simulerà il programma.



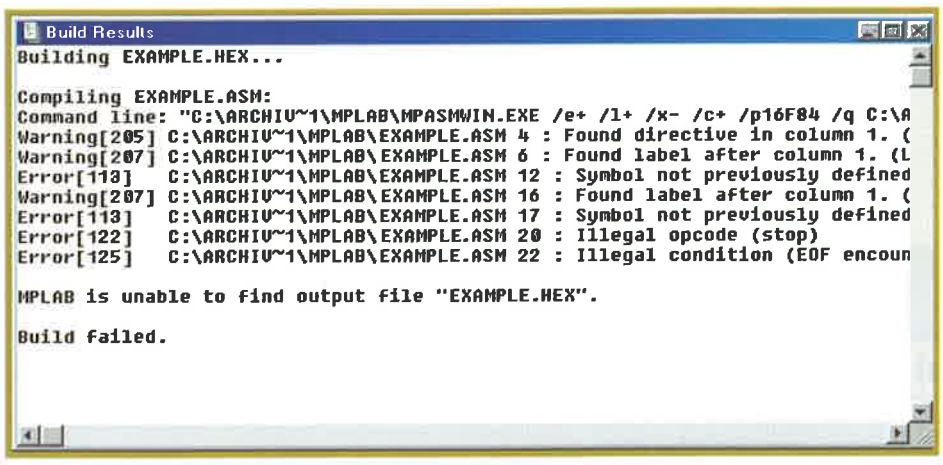
Finestra per aggiungere nodi al progetto.



Finestra delle proprietà del nodo.



Questa finestra indica che non sono stati commessi errori durante la compilazione.



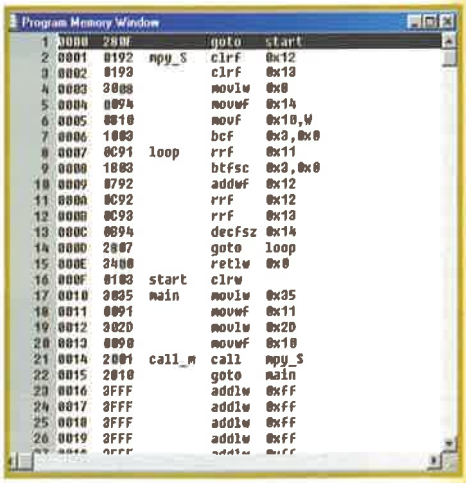
Finestra che si genera se ci sono errori di compilazione.

Abbiamo quindi definito MPASM come compilatore per questo progetto. All'interno di questa stessa finestra possiamo ancora scegliere alcune opzioni, come il tipo di errore di cui vorremmo essere avvisati dall'assembler, il sistema di numerazione, ecc. e variare i parametri che rimangono. Le opzioni scelte sono mostrate nella linea di comando (Command Line).

COMPILAZIONE DEI FILES

7. Una volta definite le proprietà del progetto e i suoi files, per compilarlo dovremo scegliere l'opzione Project>Make Project, tramite la quale, se non sono stati commessi degli errori, verrà generato il file HEX da caricare nella memoria di programma.

8. Durante la generazione del file HEX, il compilatore MPASM ci può avvisare di tre tipi di eventi: errori (errors), avvisi (warnings) e messaggi (messages), i quali possono impedire che si crei il file HEX. Nell'help in linea di MPLAB sono descritti ad uno ad uno questi tipi di eventi.



Memoria di programma con il file HEX caricato.

Scheda delle istruzioni: l'istruzione bsf

Si può dire che insieme alla "bcf" è una delle istruzioni più utilizzate nella programmazione del PIC. Il suo obiettivo è di porre a 1 un bit di un registro. Il nome o la posizione del registro all'interno della memoria dei dati, sarà fornito come primo parametro di istruzione, e il numero del bit come secondo parametro. Sia uno che l'altro iniziano a contare da 0.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
b s f	f	b

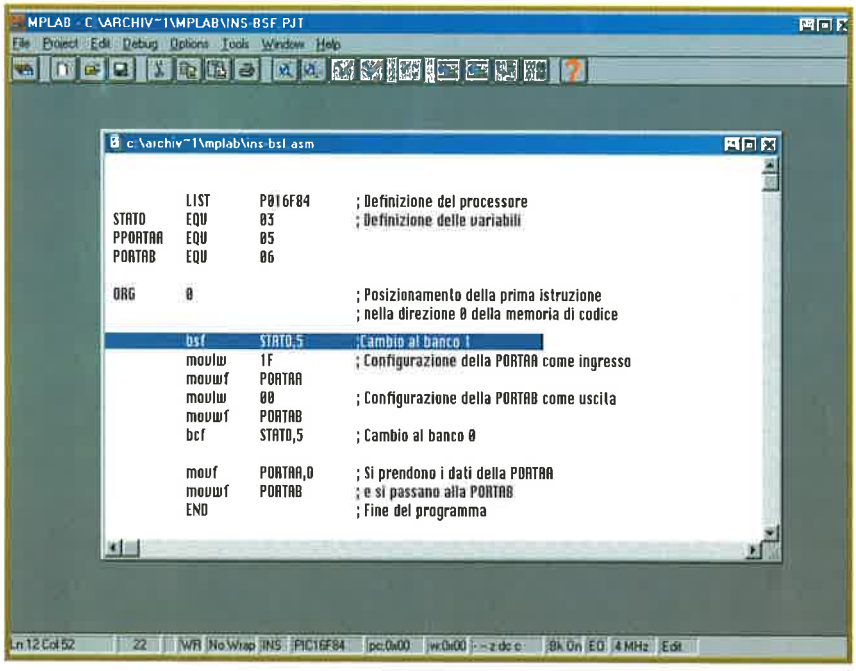
Operazione: Messa a 1 di un bit di un registro. ☐
Cicli: 1.
Codice OP: 01 01 bb bfff ffff
Flags: Nessuno.

Caratteristiche dell'istruzione bsf.

ESEMPIO CON L'ISTRUZIONE bsf

A differenza delle istruzioni di movimento, questa istruzione non varia completamente il contenuto di un registro, dato che serve per modificare bits individuali, lasciando il resto così com'è.

Questo è molto utile nell'uso di alcuni registri specifici, dove ogni bit ha un significato e deve essere trattato in modo singolo, al momento giusto, senza che gli altri debbano variare. Un esempio tipico è il cambio di banco; normalmente si lavora nel banco 0, però nella maggior parte dei programmi è necessario accedere al banco 1 per configurare una serie di registri, e dopo tornare al banco 0 per lavorare. Il cambio fra questi banchi si realizza modificando il bit 5 del registro STATO (registro che occupa la posizione 03 della memoria dei dati). Il bit è conosciuto come RP0 e, dato che la numerazione dei bits inizia da 0 da destra verso sinistra, occupa la posizione del sesto bit. Nell'esempio si vede questo cambio di banco; la linea di istruzione bsf è stata evidenziata in un altro colore.



Uso dell'istruzione bsf per il cambio di banco.

Scheda delle istruzioni: l'istruzione bcf

Questa istruzione realizza l'operazione contraria alla "bsf"; pone a 0 un bit di un registro. Il nome o la posizione del registro all'interno della memoria dei dati è anche fornito come primo parametro dell'istruzione, e il numero del bit all'interno del registro come secondo parametro. Le sue caratteristiche sono illustrate nella figura a fianco.

Come si può osservare il mnemonico di ambedue le istruzioni si differenzia solo da una lettera, la "s" della bsf è sostituita dalla "c" della bcf. Entrambe le lettere indicano il cambio di comportamento delle istruzioni, dato che il loro significato è porre a uno (set) e porre a zero (clear) rispettivamente, in questo caso un bit (b) del registro (f). Questo modo di nominare le istruzioni lo troveremo molto spesso nell'utilizzo del PIC.

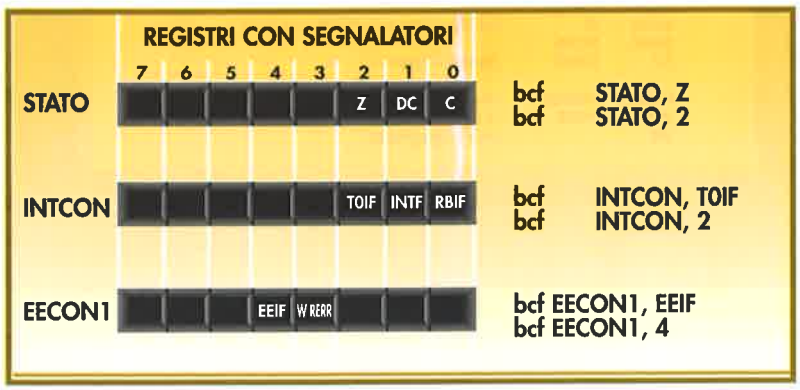
MNEMONICO	PARAMETRO 1	PARAMETRO 2
b c f	f	b
Operazione:	Messa a 0 di un bit di un registro. ☐	
Cicli: ☐	1. ☐	
Codice OP:	01 00 bb bfff ffff	
Flags: ☐	Nessuno.	

Caratteristiche dell'istruzione bcf.

ESEMPI CON L'ISTRUZIONE bfc

Riguardo l'istruzione bsf abbiamo già presentato un esempio, in cui entrambe le istruzioni si utilizzano in maniera congiunta, una per passare al banco 1 della memoria dei dati e l'altra per tornare al banco 0.

Però anche se questa combinazione si trova nella maggior parte dei programmi con il PIC, queste istruzioni non sono solo utilizzate per quello e non è detto che debbano essere utilizzate insieme. Per esempio un altro uso molto importante che si può fare con l'istruzione bcf è la messa a 0 o reset dei segnalatori, flags o bandiere, quando queste si attivano. Normalmente questi segnalatori si attivano in modo singolo, quando si produce un evento, che potrebbe essere il termine di un'operazione, il salto ad un'interrupt, eccetera. Questo viene detto attivazione via hardware, però la loro disattivazione deve essere fatta via software, sarebbe a dire, noi dovremo mettere a 0 questi flags tramite un'istruzione.



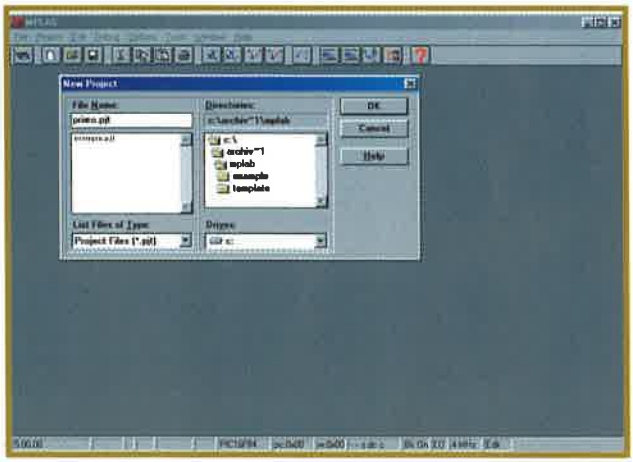
Un uso molto comune dell'istruzione bcf è il reset dei segnalatori.

Per esempio, all'attivarsi del contatore TMR0, avremo un bit all'interno del registro INTCON che si pone a 1 per avvisarci di questo fatto. Se vogliamo utilizzare il TMR0 più volte all'interno di un programma, dobbiamo rimettere a 0 questo segnalatore ogni volta che si attiva, in questo modo tornerà ad avvertirci della sua messa a 1 quando il processo si ripete.

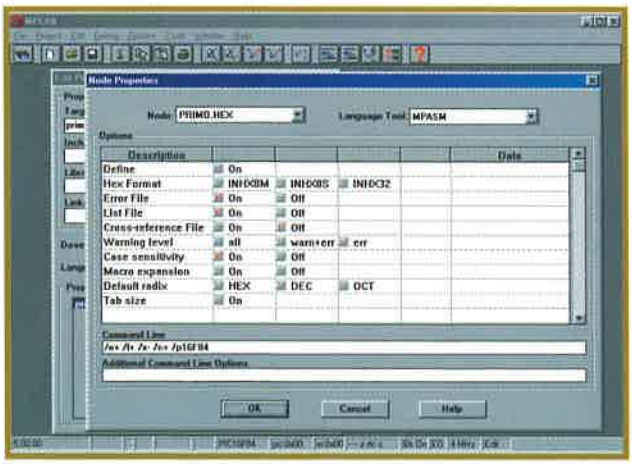
Questo si fa con l'istruzione bcf, all'interno del programma, dove il registro INTCON sarà il primo parametro e il bit di attivazione TMR0 il secondo.

Applicazione pratica: compilazione del primo programma

Una volta visto il primo programma e l'uso del compilatore MPASM nelle sezioni precedenti, seguiremo ora le fasi che abbiamo descritto per l'elaborazione di un progetto e la compilazione di un programma, basandoci su un esempio concreto.



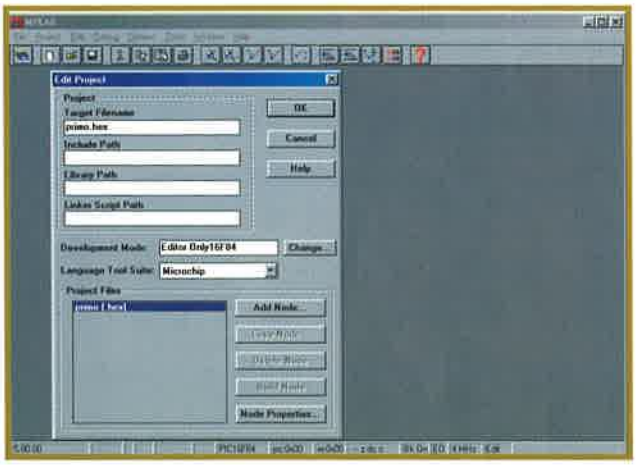
Creazione di un progetto.



Finestra di proprietà del nodo.

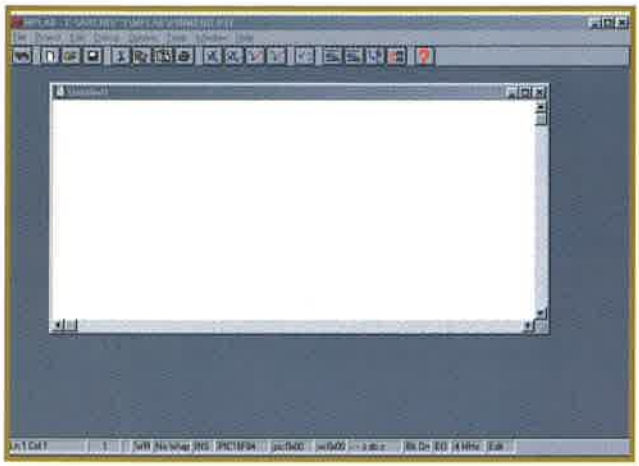
CREAZIONE DI UN PROGETTO

1. Eseguire l'ambiente di sviluppo MPLAB, e creare un progetto di nome primo.pjt.
2. Selezionare il nome del progetto in modo che si attivi il pulsante di proprietà del nodo.
3. Entrare nella finestra di proprietà del nodo.

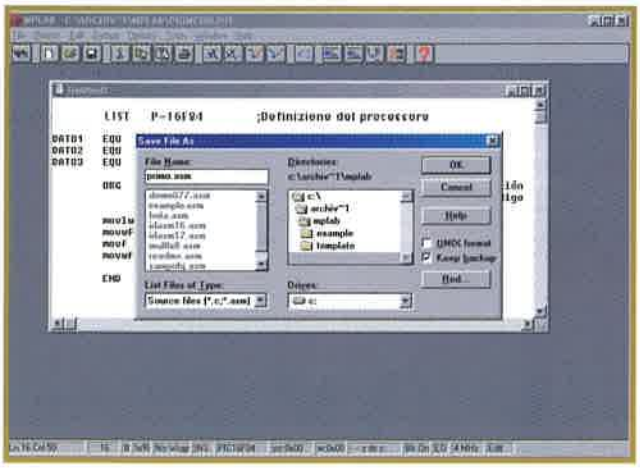


Attivazione del pulsante di proprietà del nodo.

Assicurarsi qui, che il linguaggio scelto (Language Tool) sia MPASM. Dato che non abbiamo editato il programma non possiamo sommare nessun nodo da assemblare. Uscire dalla finestra di Edit Project.



Finestra del nuovo file.

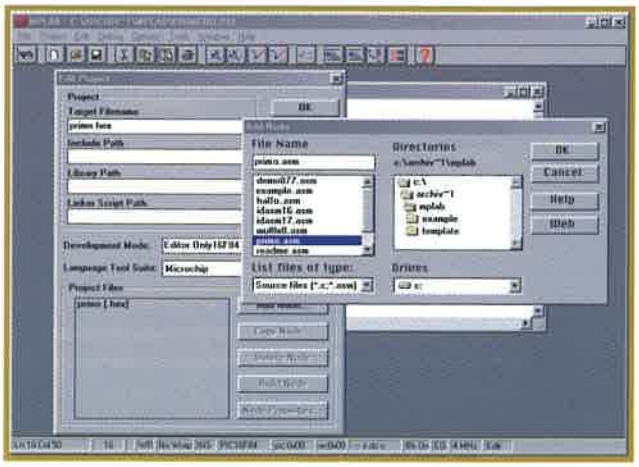


Assegnazione di un nome al programma.

SCRITTURA DEL PROGRAMMA

4. Creare un nuovo file (File>New).
5. Scrivere nella finestra corrispondente il programma che diamo come esempio di primo programma in assembler.

Scrittura del programma.

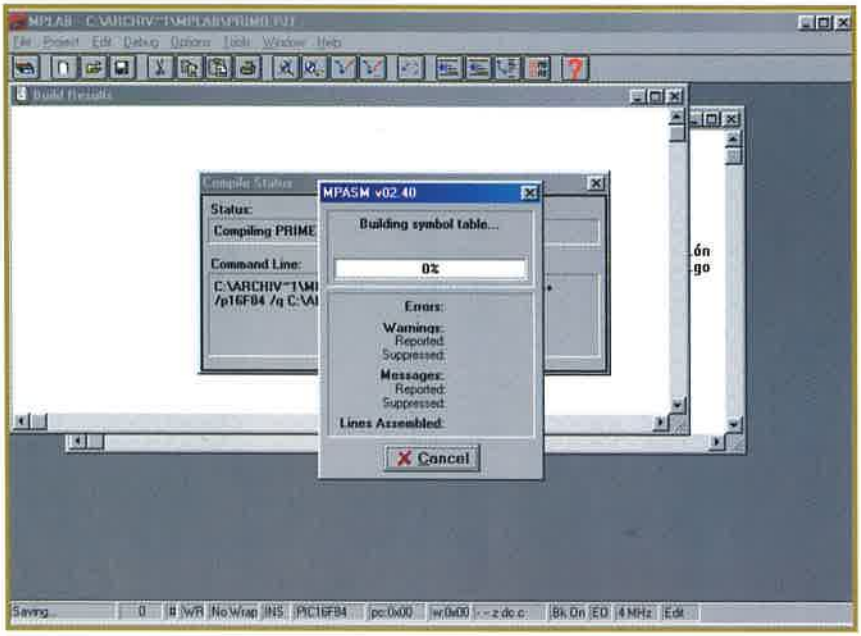


Somma di un file al progetto.

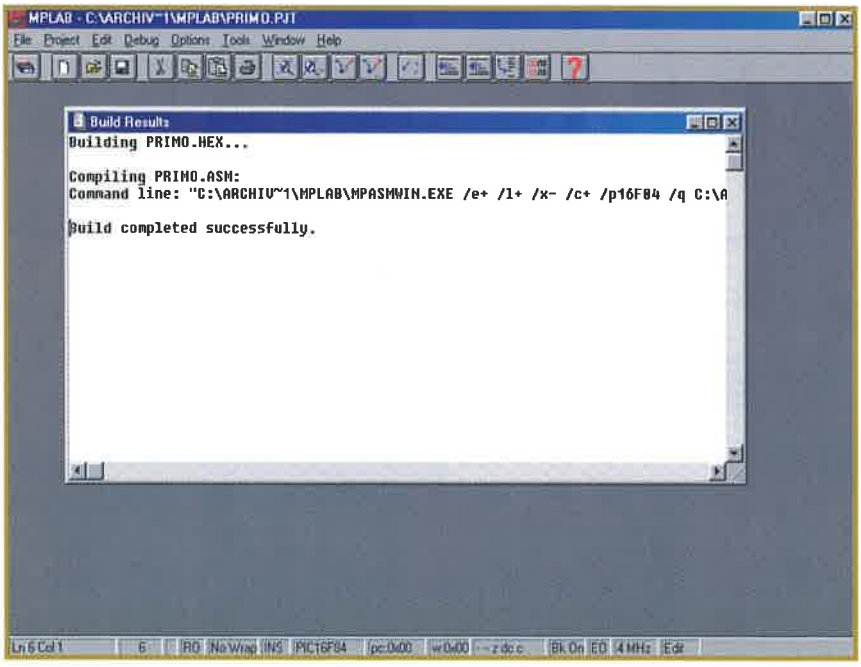
Questo sarà il programma che più tardi sommeremo al progetto.
6. Salvatelo (File>Save As...) con il nome di primo.asm nella directory dove avete installato MPLAB.
7. Ora che disponiamo di un file (nodo), possiamo aprire la finestra di opzioni del progetto (Project>Edit Project) per aggiungerlo.

COMPILAZIONE DEI FILE DI UN PROGETTO

8. Ora manderemo a comporre il progetto (Project>Make Project), mentre il software compila tutti i file del progetto, nel nostro caso uno solo; apparirà una finestra di comunicazione.
9. Se non sono stati commessi errori nella scrittura del



Compilazione dei file.

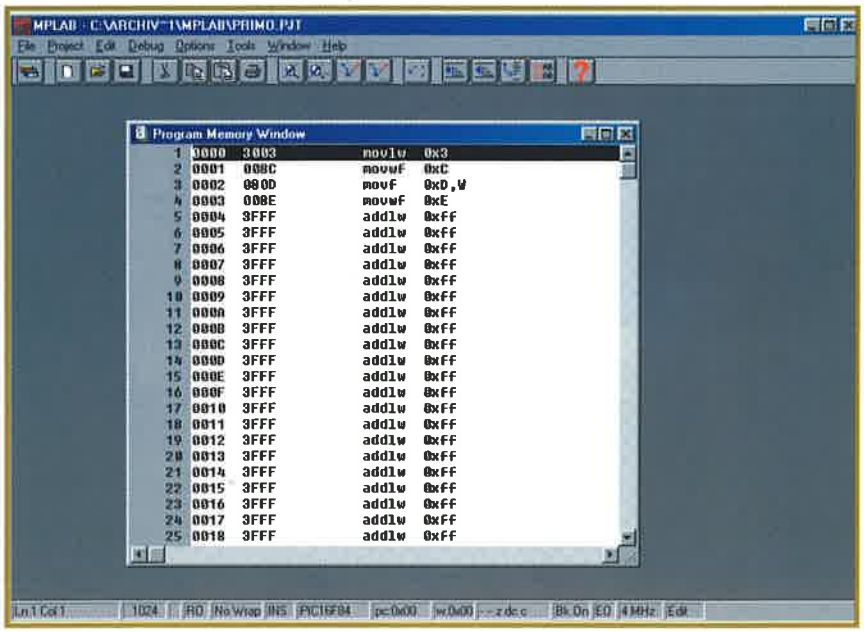


Finestra di progetto compilato con successo.

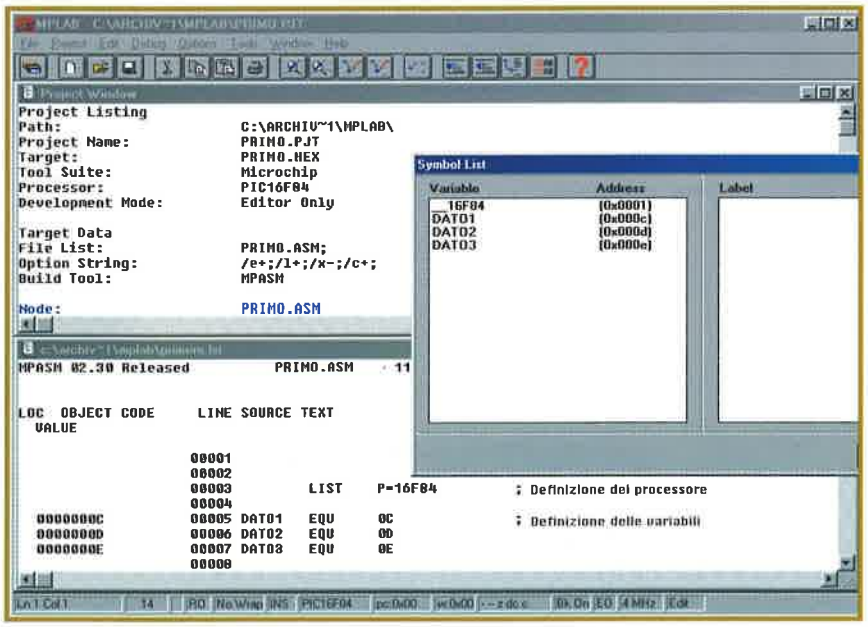
programma, deve apparire una finestra come quella della figura, che ci informa della compilazione conclusa con successo.

OSSERVAZIONI SUL CODICE GENERATO

10. Se si apre la memoria di programma (Window>Program Memory), si può osservare come verrà caricato il file nel PIC. Anche se le linee compilate nella memoria di programma sono 17, vengono caricate solo le istruzioni, sarebbe a dire quattro indirizzi di memoria. A partire dalla linea 5 (indirizzo 0004 di memoria) quello che appare ha un codice fisso, per segnalare che non c'è alcun codice caricato.
11. All'interno del menù Window possono essere aperte altre finestre che forniscono diverse informazioni sul progetto e sui file.



Finestra della memoria delle istruzioni con il programma caricato.



Altre finestre che ci forniscono dati sul progetto.

Schede di istruzioni: l'istruzione addlw

L'istruzione "add" è una delle due istruzioni aritmetiche che può eseguire un PIC16F84: la somma. L'altra è la sottrazione. Il PIC16F84 non sa moltiplicare, né dividere, né tanto meno può fare radici quadrate, potenze; nulla di tutto questo. Questo è dovuto ad una delle caratteristiche essenziali del suo insieme d'istruzioni: la loro semplicità, che fa sì che siano molto rapide nell'esecuzione. Nonostante questo se si desidera eseguire alcune di queste istruzioni complesse, si possono costruire a partire dalle istruzioni semplici.

La somma nel PIC ha due modalità: "addlw" e "addwf". La "addlw" somma un valore letterale al contenuto del registro di lavoro W, e lascia il risultato in quest'ultimo.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
a d d l w	k	

Operazione:	Somma del valore letterale **k** al registro di lavoro **W**
Cicli:	1
Codice OP:	11 111x kkkk kkkk
Flags:	C, DC, Z

Caratteristiche dell'istruzione addlw.

ESEMPI CON L'ISTRUZIONE ADDLW

Supponiamo che il registro di lavoro W contenga il valore 05 prima di realizzare l'operazione, e si vuole sommare il valore esadecimale FF. Vediamo quello che succederà dopo l'esecuzione dell'istruzione.

Nella seguente figura si rappresenta questo fatto mostrando i valori in diversi codici numerici, al fine di facilitarne la comprensione.

Il registro di lavoro, così come gli altri registri dei dati, ha solo 8 bits, il che significa che il massimo valore che può contenere è FF (255). Quando dopo un'operazione come in questo caso il risultato è maggiore del suddetto numero, si genera quello che si conosce come riporto o carry, e per avvisare di questo fatto si attiva un flag chiamato nel medesimo modo (C).

		ESADECIMALE	BINARIO	DECIMALE
valore iniziale		05	00000101	5
↓				
addlw FF	+	FF	11111111	255
↓				
valore finale W		04	00000100	4
	C=1			

Risultato dell'istruzione addlw con i valori dati.

Schede di istruzioni: l'istruzione addwf

L'istruzione addwf è la seconda modalità di somma esistente nel repertorio delle istruzioni del PIC16F84. La somma si realizza fra il contenuto di un registro qualsiasi della memoria dei dati, e il registro di lavoro W.

Il risultato dell'operazione può essere lasciato nel medesimo registro o nel registro di lavoro W, secondo il valore del parametro d.

Così come con l'istruzione addlw sono interessati i flags C, DC e Z.

Utilizzo del flag DC dopo l'operazione di somma.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
a d d w f	f	d

Operazione: Somma del contenuto del registro **f** con il contenuto del registro di lavoro **W**. Se **d=0** il risultato si lascia nel registro di lavoro **W**, e se **d=1** si lascia nello stesso registro **f**

Cicli: 1

Codice OP: 00 0111 dfff ffff

Flags: C, DC, Z

Caratteristiche dell'istruzione addwf.

ESEMPI CON L'ISTRUZIONE ADDWF

I flags C e Z sono stati trattati negli altri esempi, per cui rimane solo da spiegare il comportamento del flag DC. Questo flag va a 1 quando si produce un riporto nel 4° bit, anziché nell'8°, come succede con il flag C. Viene utilizzato, se necessario, quando bisogna tener conto in un registro, il superamento del valore massimo di un digit.

Per esempio, se un numero è scomposto in tre digits esadecimali, e ogni digit è memorizzato in un registro, quando si produce un riporto al superamento del valore F, il flag DC verrà messo a 1; il che significa che dovremo mettere a 0 il registro e sommare 1 al digit seguente. Questo è mostrato graficamente nella figura in alto.

Strumenti di lavoro: il simulatore MPLAB-SIM

L'ambiente di sviluppo MPLAB possiede tra i suoi strumenti un simulatore, il MPLAB-SIM. Un simulatore permette di eseguire nel computer il programma che più tardi sarà registrato nel microcontroller. Si può vedere così se il programma si comporta come noi pensiamo, il valore dei registri in ogni momento, e in generale tutto quello che succede dentro il microcontroller al momento di eseguire il programma. Il MPLAB-SIM, in particolare, è progettato per due funzioni che sono illustrate nel riquadro seguente.

Nonostante questo, non dobbiamo confondere un simulatore con un emulatore in circuito. I simulatori rappresentano la situazione ideale e non un'esecuzione reale al 100% di un programma, ed hanno delle limitazioni. Permettono di eseguire un codice, modificarlo, ed eseguirlo nuovamente, simulare gli input sulle linee d'ingresso, seguire l'esecuzione linea a linea del programma e i suoi registri...., però si differenziano dagli emulatori come il MPLAB-ICE in tre cose fondamentali:

Tempo di Ingressi/Uscite.

Gli stimoli esterni nel MPLAB-SIM sono elaborati una volta ogni ciclo d'istruzione, perciò i segnali di durata inferiore non possono essere captati, e pertanto simulati, cosa che è permessa dagli emulatori in circuito.

Velocità di esecuzione.

La velocità dei simulatori software è sempre minore rispetto a quella degli emulatori. Questo può essere un vantaggio o uno svantaggio al momento di programmare, dipende dal programma e dall'utente. In MPLAB-SIM si possono eseguire istruzioni ad una velocità dell'ordine del millisecondo per istruzione, mentre l'esecuzione reale di un'istruzione nel PIC è di un microsecondo.

Costi.

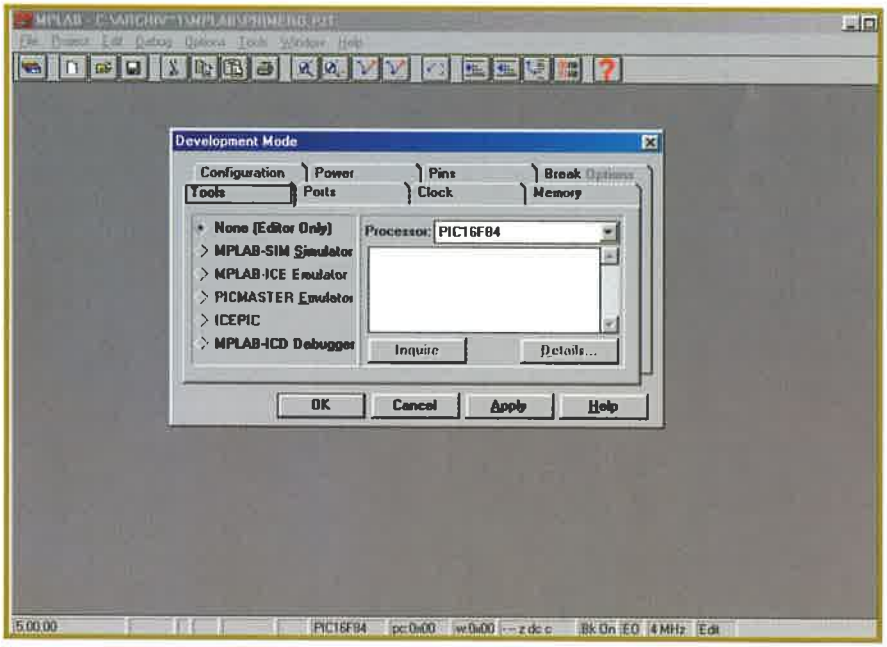
I simulatori hanno il vantaggio di non necessitare di hardware esterno al PC per il loro utilizzo, per questo se non è necessaria un'esecuzione in tempo reale, di solito sono sufficienti per trovare e correggere gli errori della maggioranza dei programmi.

Differenze fondamentali tra i simulatori e gli emulatori in circuito.

Permettere l'esecuzione dei programmi scritti per i microcontroller PIC di Microchip, come ad esempio le famiglie PIC12Cxx, PIC16C5x, PIC16Cxx.

Assistere l'utente nella verifica dei programmi.

Funzioni realizzabili con MPLAB-SIM.



Opzioni per cambiare l'ambiente di sviluppo.

CONFIGURAZIONE DEL MPLAB PER LAVORARE CON MPLAB-SIM

Aprendo MPLAB, il modo di sviluppo standard è di scrittura. Questo significa che dobbiamo configurarlo prima di fare una simulazione, altrimenti le opzioni non saranno accessibili. La procedura per cambiare il modo di lavoro è contenuta nel menù opzioni (Options> Development Mode). Eseguendolo, a seconda del programma installato, deve apparire una finestra con in parte opzioni accessibili, così com'è mostrato nella figura della pagina precedente.

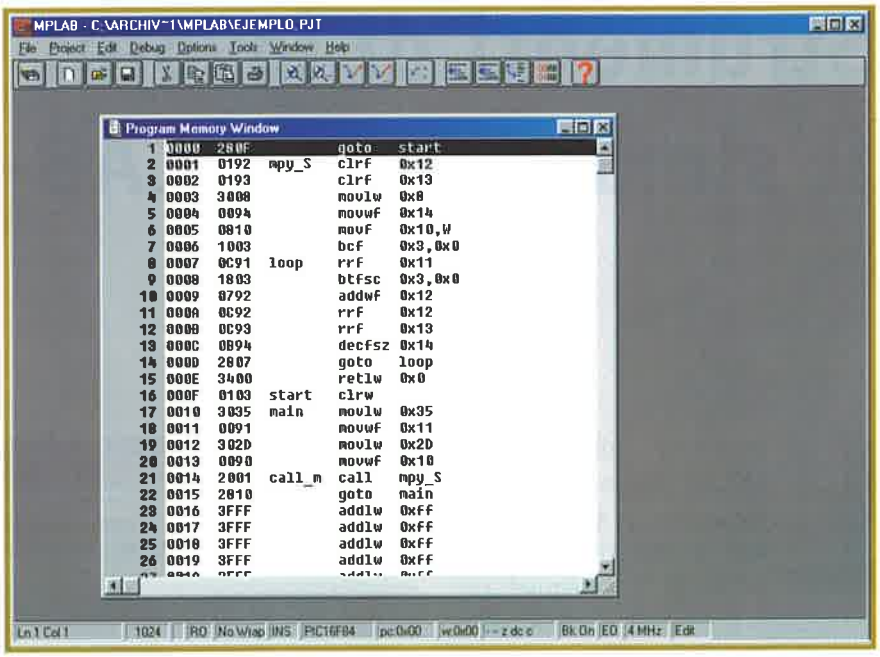
None (Editor Only)

Permette solo la scrittura e la compilazione dei programmi, e le funzioni di progetto.

Simulatore MPLAB-SIM (MPLAB-SIM Simulator).

Permette la simulazione dei programmi oltre la scrittura e la compilazione. Dobbiamo scegliere il microcontrollore che si va a simulare nella parte destra della medesima finestra.

Caratteristiche dei modi disponibili.



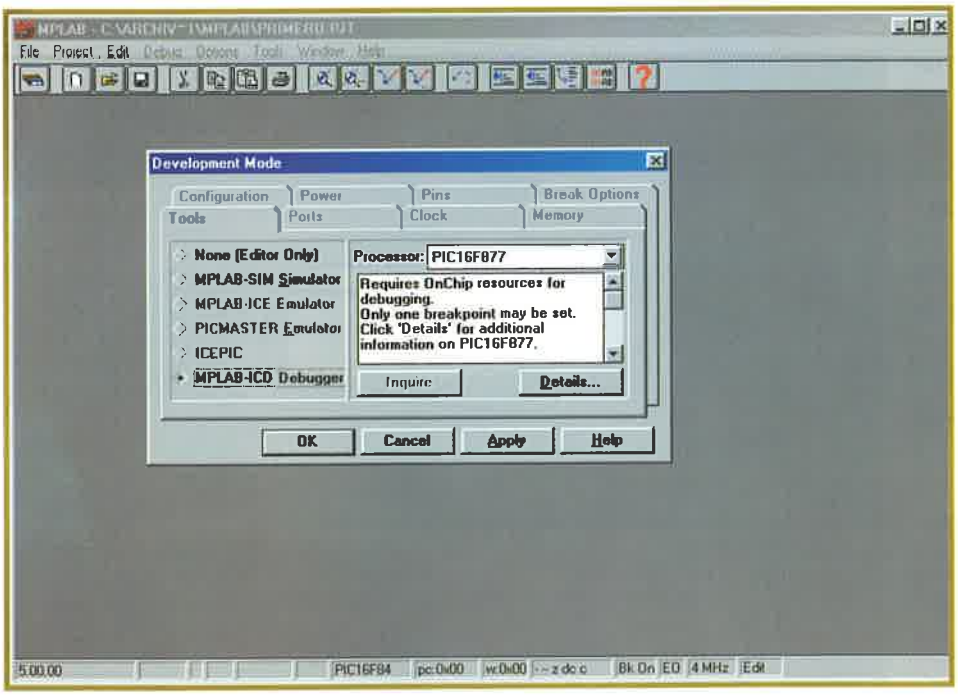
Finestra della memoria d'istruzioni con caricato il programma con il quale si sta lavorando.

In questo caso possiamo scegliere fra differenti modi, secondo le necessità.

L'MPLAB-ICD Debugger si vede evidenziato, perché è stato installato sul software come componente di MPLAB (vedere fascicolo dedicato a MPLAB), se proviamo a selezionarlo ci apparirà un errore dovuto al fatto che non abbiamo hardware associato. Inoltre, tra i microcontrollori che ammettono questa opzione non appare il PIC16F84.

Gli altri modi non sono selezionabili, dato che non sono stati installati nel software, e nel caso lo fossero non potrebbero essere scelti, poiché non disponiamo dell'hardware.

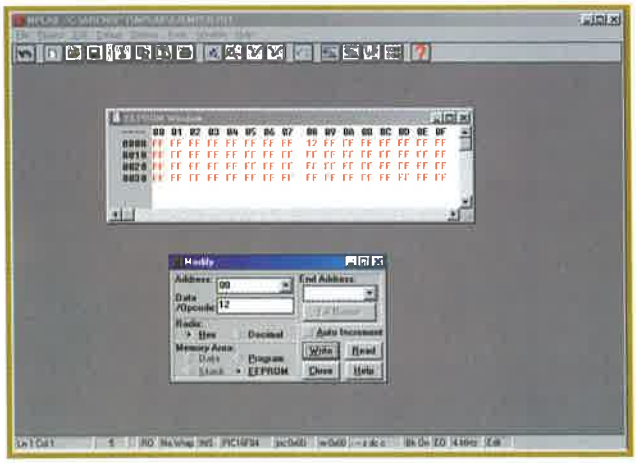
Per realizzare delle simulazioni dobbiamo selezionare la seconda opzione proposta. Possiamo effettuare una prova e verificare, per ognuna delle opzioni, le possibilità che ci offrono i menù di Debug e Window.



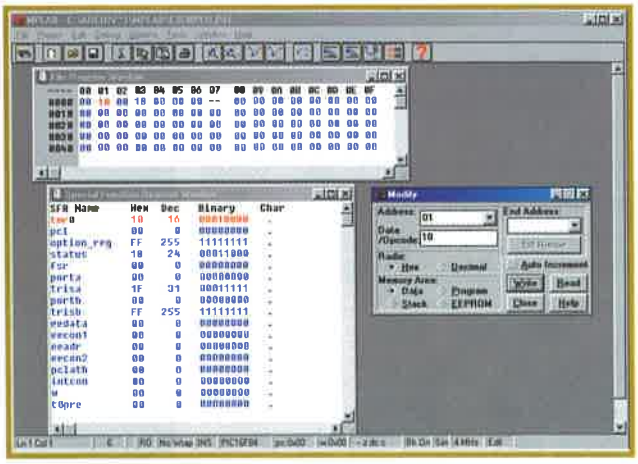
Messaggio d'errore che appare selezionando il MPLAB-ICD Debugger.

FINESTRE ACCESSIBILI NEL MODO SIMULAZIONE

Durante la simulazione del programma può essere interessante vedere in che posi-



Finestra di memoria EEPROM tramite la quale si modifica un valore.



Finestre dei registri con un dato appena modificato.

zione si trovano le istruzioni che si stanno eseguendo.

Questo è possibile attraverso la Finestra di Memoria del Programma (Program Memory Window).

Le informazioni disponibili si possono ottenere in differenti formati, da scegliere cliccando nell'angolo superiore sinistro della finestra.

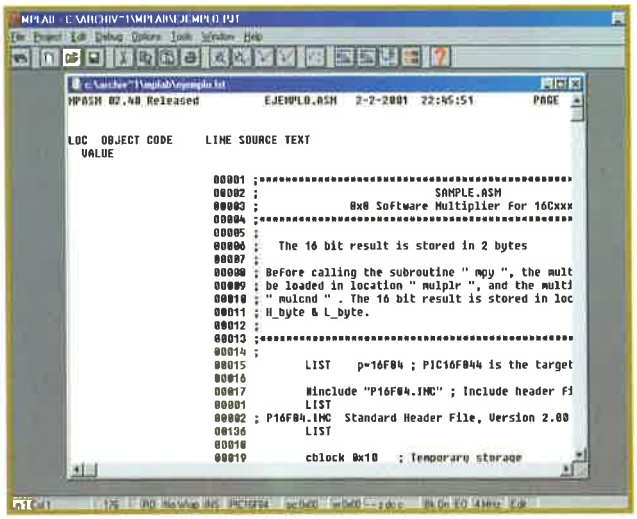
Questi formati sono quelli che ci mostrano il codice delle istruzioni in esadecimale oppure così com'è stato scritto, facilitandone la verifica. Inoltre, appaiono informazioni addizionali sul numero d'ogni istruzione e sulla posizione all'interno della memoria di programma. Se il nostro programma lavora con la memoria EEPROM dei dati, possiamo vederne i valori che contiene nella EEPROM Window.

Dall'opzione Window>Modify potremo modificare il valore degli indirizzi della EEPROM che ci interessano, per fare la simulazione.

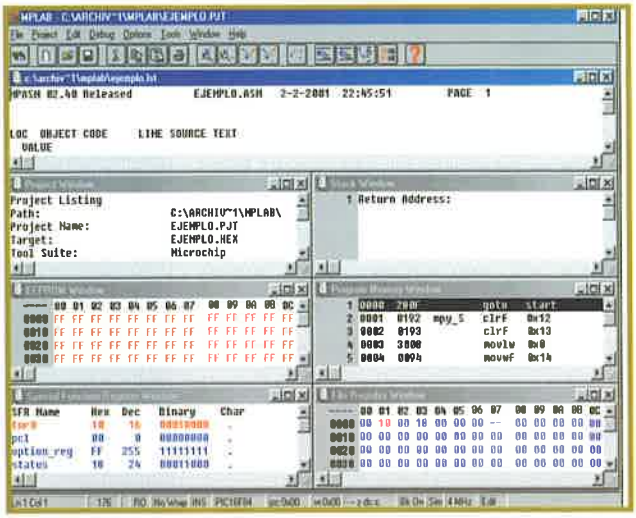
Nella figura corrispondente abbiamo modificato il valore della posizione 08 introducendo il valore 12 esadecimale.

Un'altra possibilità è decidere se il programma che vogliamo simulare deve essere visualizzato così come lo abbiamo scritto o con informazioni addizionali che fornisce il simulatore alla compilazione, come gli indirizzi d'ogni istruzione, le etichette definite, il numero d'indirizzi della memoria d'istruzione non utilizzati, ecc. Questa informazione è disponibile nella finestra del listato (Absolute Listing).

La finestra generale dei registri (File Register Window) e quella dei registri speciali (Special Funtion



Informazioni addizionali del programma assemblato fornito.

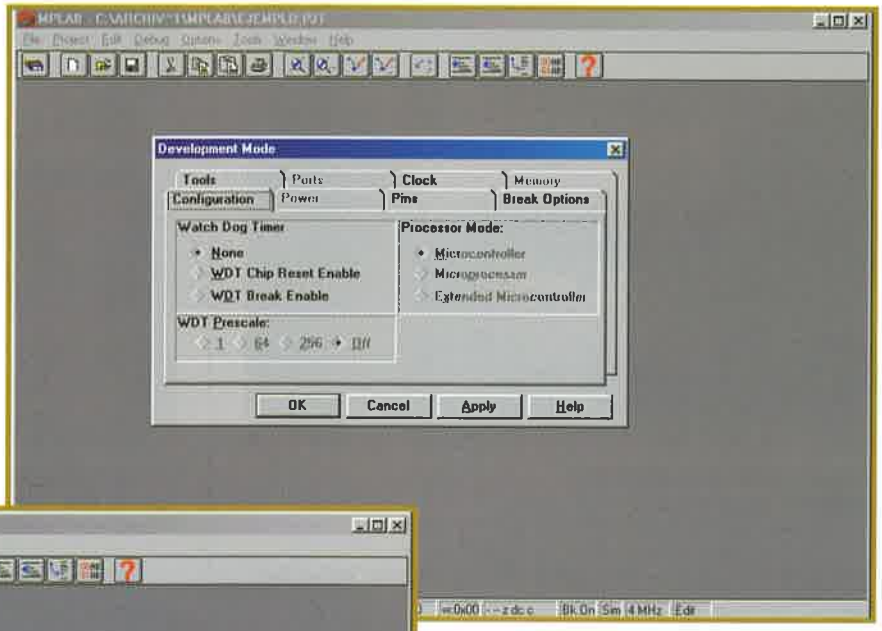


Alcune finestre utili per realizzare la simulazione.

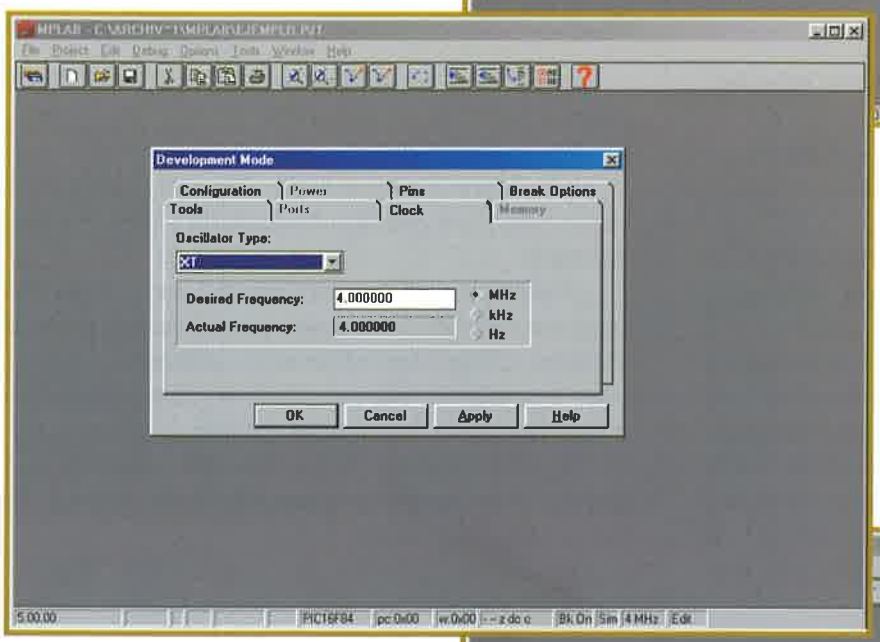
Register Window) generalmente è molto utile durante una simulazione.

La finestra dei registri speciali contiene solo quelli specifici, mentre quella generale riunisce sia i registri specifici sia i generali. Un cambiamento di valore all'interno di una finestra, Window > Modify, influenza ambo le finestre se gli indirizzi esistono in entrambe.

Sono interessanti anche altre opzioni come lo stack delle istruzioni (Stack Window) o informazioni del progetto (Project Window). Normalmente si tengono aperte diverse finestre nello stesso tempo. Le utilizzeremo, così come i diversi modi d'esecuzione di un programma (menù Debug), a seconda di come procederemo con i concetti e la programmazione.



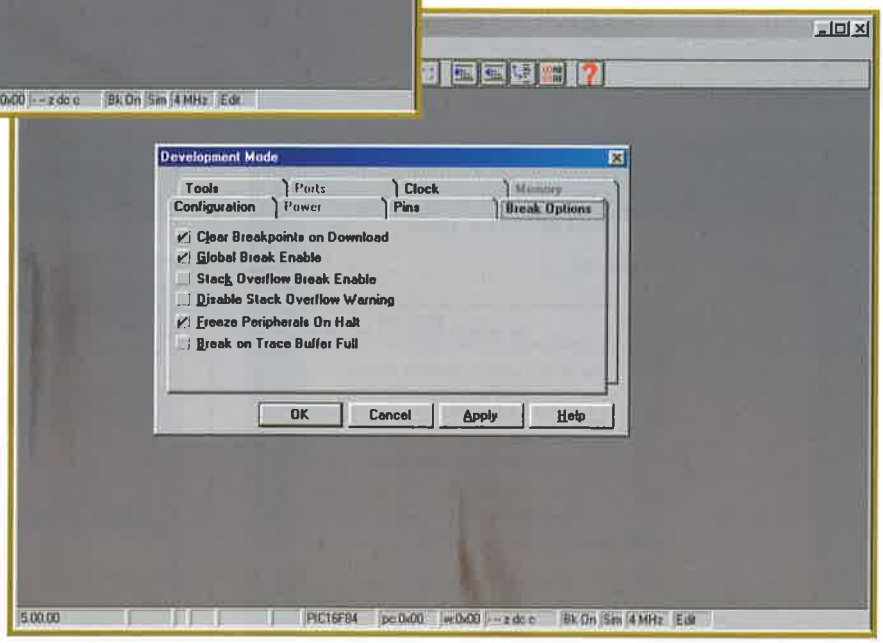
Finestra per l'abilitazione del Watchdog.



Finestra per la determinazione del tipo di oscillatore.

OPZIONI DA VARIARE NELLA SIMULAZIONE

Prima di realizzare delle simulazioni, dobbiamo tornare alle finestre del modo sviluppo, per vedere che tipo d'oscillatore è stato utilizzato e per determinare la velocità del processore, l'abilitazione o no del Watchdog, o il modo di collocare i punti in cui vogliamo fermare la simulazione (break points).



Differenti opzioni dei break points.

L'istruzione INCF

Questa istruzione può considerasi un caso speciale dell'istruzione add, dato che somma 1 al contenuto di un registro, cioè incrementa di una unità il contenuto del medesimo. Il risultato può essere portato nel registro di lavoro W al posto del registro di origine, in modo che quest'ultimo non venga modificato. Questo dipende, come sempre, dal valore del parametro d.

Osserviamo che l'unico flag interessato a questa istruzione è Z, per cui se ci interessa controllare i flag C e/o DC non potremo usare questa istruzione.

OPERAZIONE	UTILIZZO DELL'ISTRUZIONE inc	UTILIZZO DELL'ISTRUZIONE add
Incremento di un'unità del registro CONTATORE	incf CONTATORE,1	addlw 01 addwf CONTATORE,1
Incremento di tre in tre unità del registro CONTATORE	incf CONTATORE,1 incf CONTATORE,1 incf CONTATORE,1	addlw 03 addwf CONTATORE,1

Comparazione dell'uso dell'istruzione di somma e incremento.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
incf	f	d

Operazione: incremento di una unità del contenuto del registro **f**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro f.
Cicli: 1
Codice OP: 00 1010 dfff ffff
Flag: Z

Caratteristiche dell'istruzione incf.

ESEMPI CON L'ISTRUZIONE INCF

L'uso più comune di questa istruzione è nei contatori. Normalmente i contatori si incrementano di una unità per volta, per cui quest'istruzione permette un sostanziale risparmio di tempo se la compariamo con quella di somma, poiché ogni incremento comporta solo un'istruzione (un ciclo di istruzione). Infatti per la stessa operazione di incremento realizzata con l'istruzione di somma, è necessario caricare un valore (un 1) nel registro di lavoro W, e poi sommare il registro utilizzato come contatore, al registro di lavoro W. Il risultato sono due istruzioni, per cui il tempo sarà il doppio.

Se il contatore che vogliamo implementare deve essere incrementato di più di un'unità per volta, è meglio impiegare istruzioni di somma.

L'istruzione DECF

Questa istruzione è quella opposta alla "incf". Con essa si può decrementare il valore di un registro e portare il risultato al registro stesso o al registro di lavoro W. Il suo flag associato è nuovamente Z, che andrà a 1 quando il registro in questione arriva a 0.

Anche qui, come con l'operazione di incremento, quando si deve decrementare di più di un'unità un registro, è meglio farlo con un'altra istruzione, in questo caso la sottrazione.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
d e c f	f	d

Operazione: decremento di un'unità del contenuto del registro **f**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro **f**.
Cicli: 1
Codice OP: 00 0011 dfff ffff
Flag: Z

Caratteristiche dell'istruzione decf.

ESEMPI CON L'ISTRUZIONE DECF

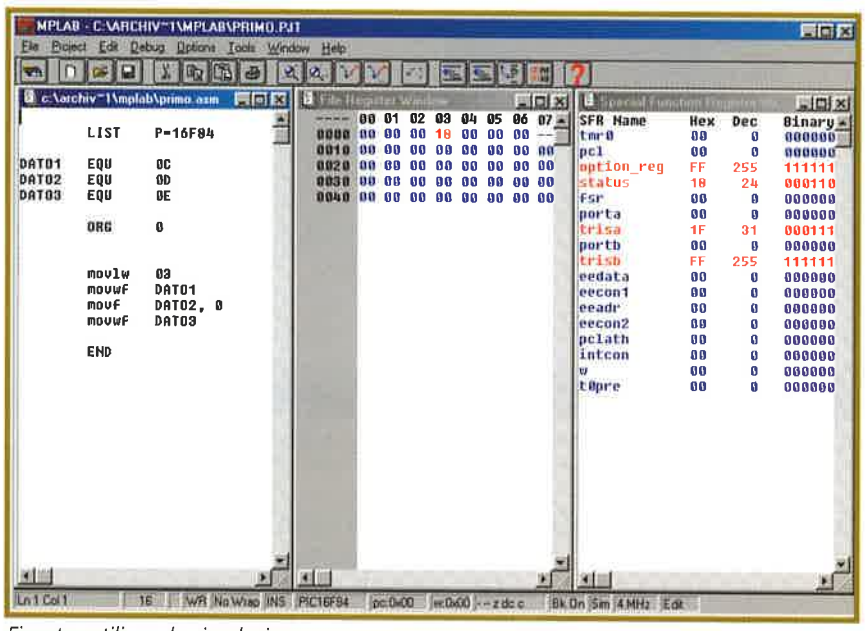
L'uso di questa istruzione è simile all'istruzione "incf", per cui non verrà ripetuta. In ogni caso, parleremo ancora dell'utilizzo di questi parametri in altre istruzioni. Normalmente quando impieghiamo un registro, facciamo riferimento a questo con un nome, e non con l'indirizzo che occupa dentro la memoria di programma. In realtà è la stessa cosa, dato che abbiamo dovuto "indicare" al compilatore che sono uguali. Questo è possibile grazie alle direttive EQU, che fanno corrispondere un nome o un'etichetta con un valore numerico. Quindi, dovremo fare attenzione all'uso che faremo in seguito di questi "nomi", dato che, a seconda dell'ambiente in cui li utilizzeremo, cioè: il parametro di istruzioni a cui faremo riferimento, significheranno una cosa o l'altra.

```
W           EQU   0
F           EQU   1
CONTATORE   EQU   0C

            decf  CONTATORE,W  ; Decremento del registro contatore portando
                               ; il risultato al registro di lavoro W
            decf  CONTATORE,F  ; Decremento del registro contatore portando
                               ; il risultato allo stesso registro F
            decf  F,CONTATORE  ; Caso errato, dato che CONTATORE non
                               ; corrisponde con 0 o con un 1, che è il
                               ; parametro che serve all'istruzione decf
            decf  W,F          ; Decremento del registro che occupa la
                               ; posizione 0 della memoria dei dati (non W)
                               ; portando il risultato allo stesso registro 0
```

Differenti significati di etichetta a seconda del luogo dove si utilizzano.

Applicazione pratica: simulazione del primo programma

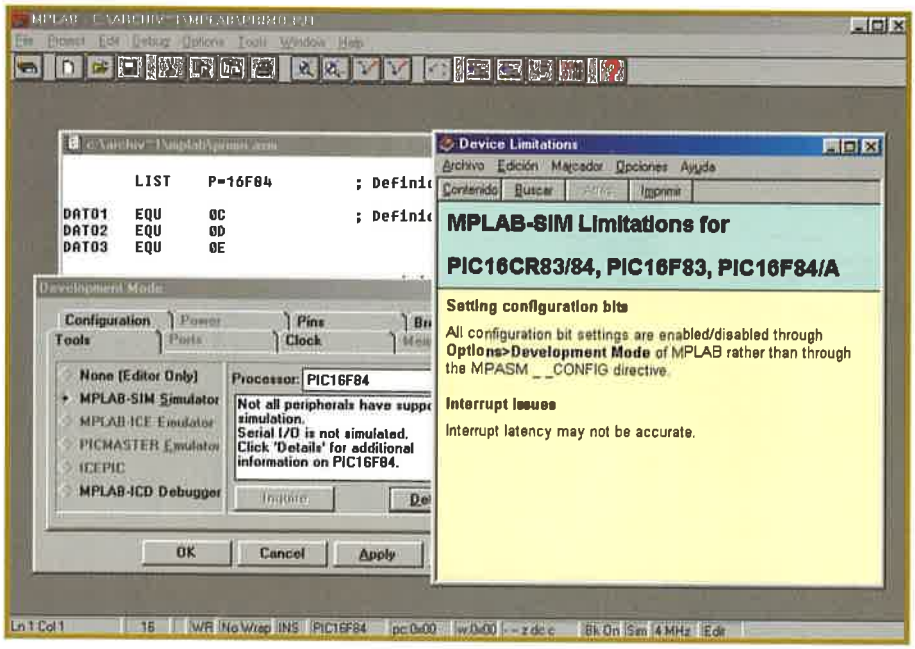
Dopo aver visto in cosa consiste il simulatore integrato MPLAB, è arrivato il momento di simulare il primo programma che abbiamo fatto e che è stato compilato con MPASM. Ora diamo un'occhiata a queste sezioni per ricordare l'enunciato del programma, i passi e gli aspetti di cui tenere conto. Fatto questo saremo pronti per cominciare la simulazione.



Finestre utili per la simulazione.

CARICARE IL PROGETTO

1 • Una volta dentro a MPLAB, la prima cosa da fare è aprire il progetto che abbiamo realizzato, primo.asm, cosa che si fa mediante il menù di progetto (Project>Open Project).



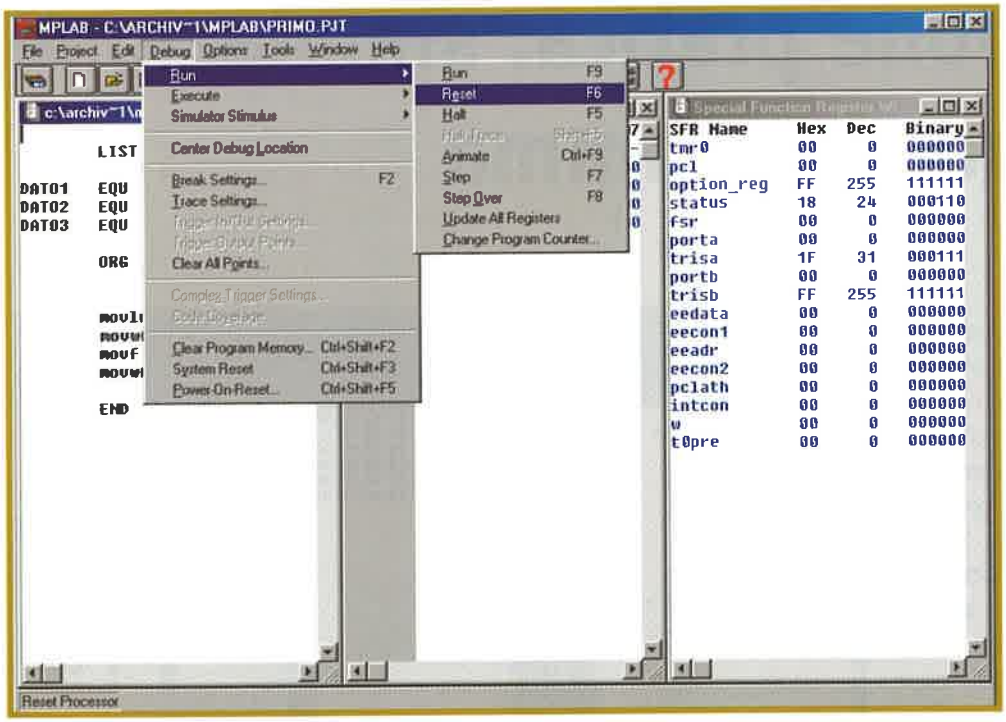
Limitazioni di MPLAB-SIM per il PIC16F84.

2 • In seguito, se non l'ha fatto l'apertura del progetto, apriremo i file o nodi associati al progetto tramite il menù dei file (File>Open).

FINESTRE UTILI PER QUESTA SIMULAZIONE

Per ogni progetto dovremo scegliere quali finestre sono utili per farci vedere quello che il programma sta facendo e quali rappresentano solo una distrazione, non fornendo informazioni essenziali. Alcune opzioni possono dipendere dalle preferenze personali, e sarà proprio l'utilizzatore che determinerà con l'esperienza il suo modo di lavoro.

3 • Per prima cosa dovremo scegliere come modo di sviluppo



Si deve fare un Reset ogni volta che si desidera iniziare una simulazione.

la simulazione, opzione inclusa dentro a Options>Development Mode. All'interno della finestra che ci appare selezionando Details, potremo vedere le limitazioni che ha MPLAB-SIM per il PIC che vogliamo simulare, nel nostro caso il PIC16F84.

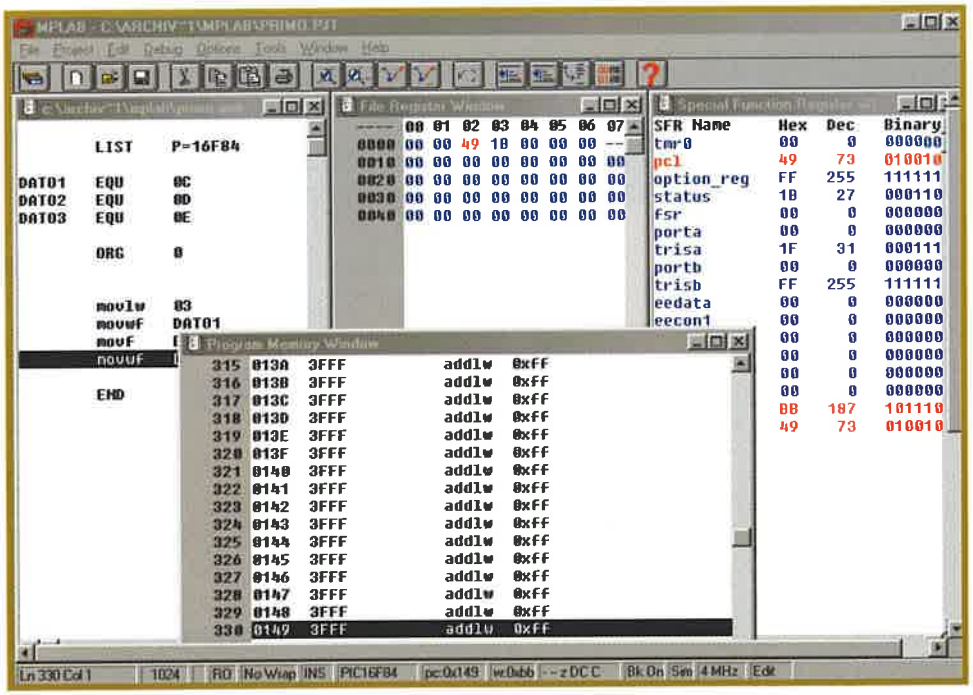
4 • Dopo aver scelto il modo, sono già disponibili diverse finestre, fra queste sceglieremo di lasciare aperte solo quella del programma, quella dei registri generali, e quella dei registri specifici. Ora dobbiamo disporre le finestre sullo schermo. Se la disposizione si effettua tramite Windows>Tile Horizontal dobbiamo ottenere una serie di finestre dove poter verificare i dati fondamentali.

OPZIONI DELLA SIMULAZIONE

Le opzioni della simulazione scelta dipendono da cosa ci interessa ottenere dal programma. Potrebbe interessarci solamente il risultato finale di un registro oppure vedere lo sviluppo di un pezzo del programma su cui abbiamo dei dubbi sul funzionamento, o vedere passo a passo tutta o parte dell'esecuzione del programma. Vedremo alcune di queste opzioni con il nostro primo esempio.

5 • Prima di eseguire il programma faremo un Reset del sistema con l'opzione Debug>Run>Reset. Ogni volta che facciamo una simulazione e vogliamo reinizializzare il sistema, dovremo effettuare questo Reset.

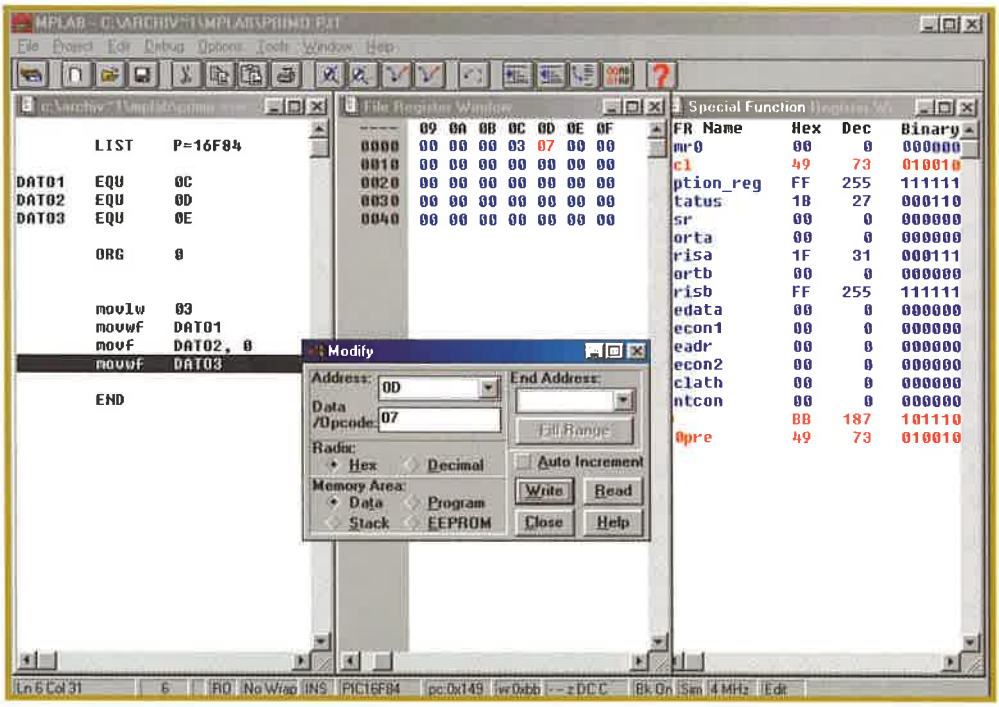
6 • Iniziamo con la prima simulazione. Selezioniamo Animate all'interno del menù Debug>Run. Come si può vedere, questa opzione non è di grande utilità, almeno in questo esempio, dato che le nostre tre istruzioni si eseguono rapidamente e poi il cursore prose-



Terminato il nostro programma, si prosegue eseguendo posizioni non valide.

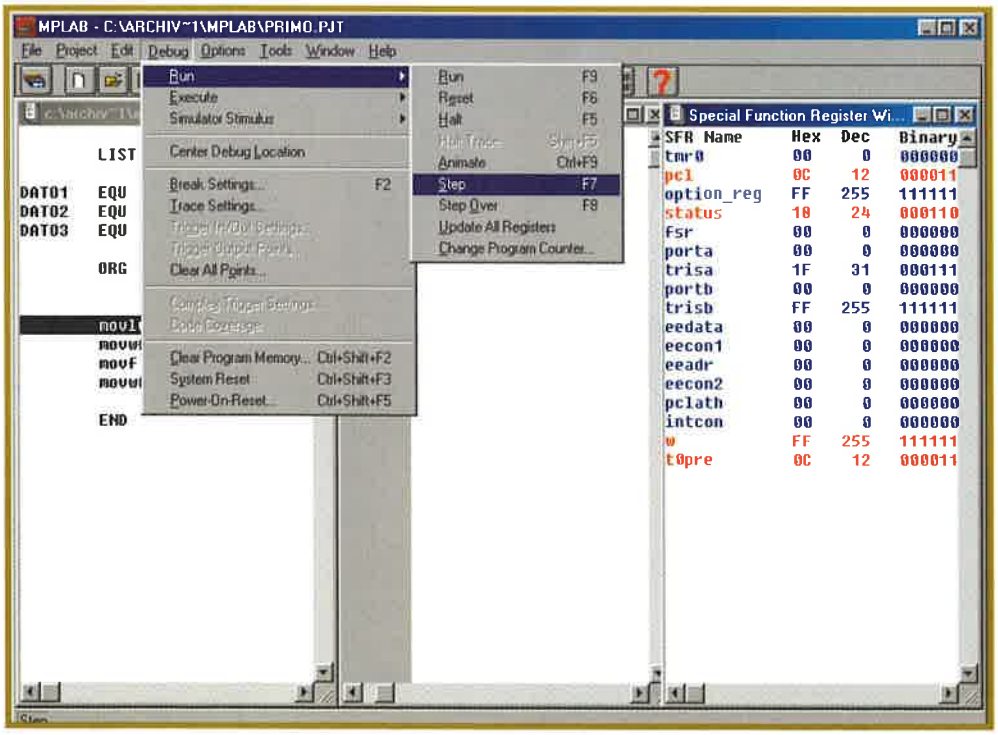
gue per il resto della memoria di programma, che non contiene valori validi. Nella figura precedente sono visibili le finestre che si possono vedere con questa simulazione.

7 • Utilizzare Debug>Run>Halt per fermare la simulazione. Ora possiamo verificare che il valore dei registri corrisponda con l'enunciato del programma: il registro 0C contiene il dato 03 e i registri 0D e 0E contengono un valore uguale. Verificare queste informazioni spostandosi all'interno delle varie finestre.



Modifica del valore di un registro.

8 • Ripetiamo la stessa simulazione, prima però cambiamo il valore del registro 0D con, ad esempio, 07. Così, eseguendo il programma, noteremo meglio che i registri 0D e 0E prendono lo stesso valore. Il cambio del valore di un registro, si fa tramite la finestra Window>Modify. Dovremo scrivere l'indirizzo (Address) e il dato (Data) che si desiderano, e cliccare il pulsante di scrittura (Write) per far sì che la variazione abbia effetto.



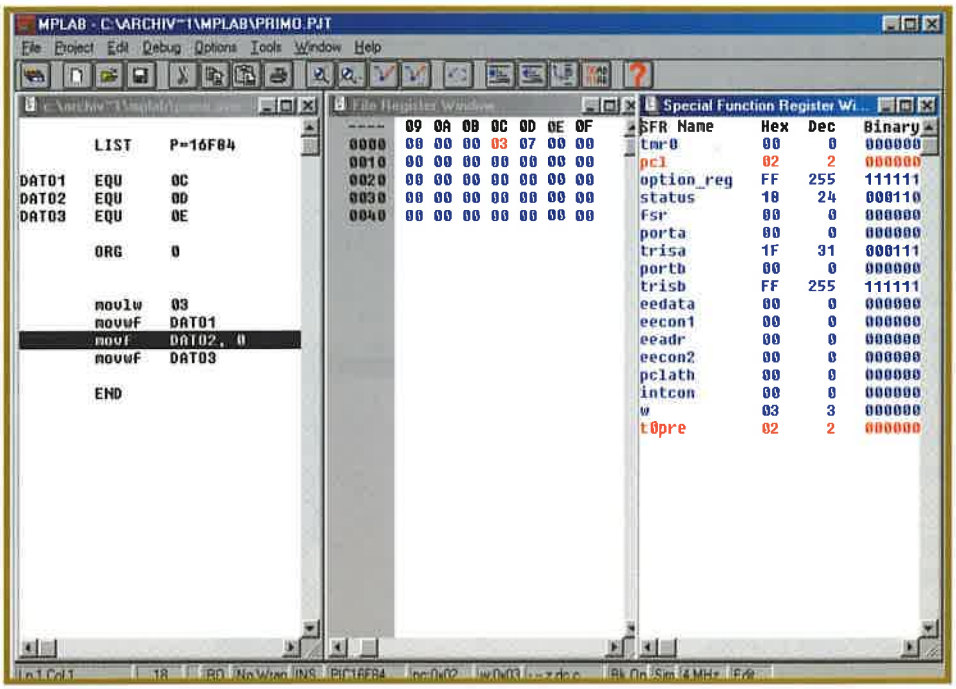
Scelta del modo di simulazione passo a passo.

9 • Il seguente tipo di simulazione lo troviamo in Debug>Run>Run. Per notare l'effetto dobbiamo tornare a mettere 00 nei registri 0C e 0E, seguendo i passi del punto precedente di modifica dei registri, poi eseguiremo il programma (Debug>Run>Run) e lo fermeremo (Debug>Run> Halt). Ci appariranno i registri secondo quanto predisposto dal programma. La differenza con il tipo di simulazione precedente, Animate, sta nel fatto che noi non vediamo come si modificano i registri, anche se internamente il processo è lo stesso, e il PC, eseguito il nostro codi-

ce continua con gli indirizzi successivi della memoria di programma.

10 • In ultimo eseguiamo il programma istruzione per istruzione. Questo modo viene anche detto tracciare, dall'inglese "to trace". Inizializzeremo i registri come nel caso precedente, oltre a resettare il sistema. Andiamo all'opzione Debug>Run>Step. Notiamo che il tasto funzione è F7.

11 • Ogni volta che premiamo F7 verrà eseguita una sola istruzione e potremo osservare il valore preso dai registri di volta in volta. Le variazioni all'interno dei registri sono segnalate dal colore rosso, così come l'istruzione che sarà eseguita in seguito. La figura mostra il processo dopo l'esecuzione della seconda istruzione; a DATO1 è già stato spostato il valore 03, il PC (pcl) punta ancora la seconda posizione della memoria delle istruzioni, e appare evidenziata la terza istruzione, che sarà la prossima ad essere eseguita.

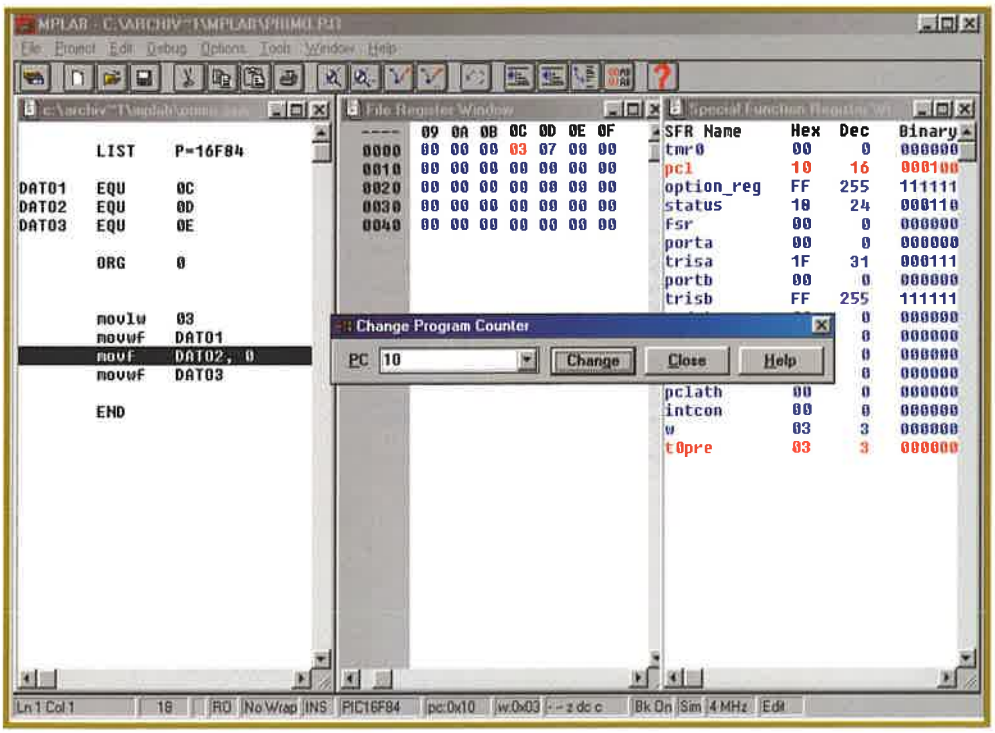


Informazioni ottenute con l'esecuzione passo a passo.

12 • Se in un determinato momento ci interessa vedere solo quello che succede passo a passo in una parte del programma, e non nel programma completo, si può modificare il valore del PC, per farlo iniziare dal punto che ci interessa. Questo si fa con l'opzione Debug>Run>Change Program Counter.

L'uso di questa opzione è tipico in programmi molto lunghi, dove la simulazione completa passo a passo risulterebbe molto noiosa.

L'opzione di simulazione passo a passo, generalmente è la più usata quando si inizia a programmare, per collaudare le nuove funzioni delle istruzioni. Introducete ora in questo stesso programma le istruzioni apprese sinora, assemblatele e realizzate la loro simulazione, vedrete che in pochissimo tempo dominerete il loro funzionamento.



Modifica del PC per iniziare la simulazione da un altro punto.

L'istruzione GOTO

Siamo arrivati all'istruzione tabù della maggior parte dei linguaggi di programmazione che la possiedono. Molti libri di programmazione strutturata, hanno naturalmente un capitolo che spiega a fondo questa istruzione, alla fine del quale si raccomanda di utilizzarla il meno possibile. Noi non abbiamo altra scelta che utilizzarla, dato che nel PIC è l'unico modo per fare un salto incondizionato, cioè, andare ad un'altra parte del programma in modo incondizionato, inoltre è la base principale dei salti condizionati, come vedremo più avanti. Pertanto, vale la pena imparare l'uso corretto di questa istruzione, per non dover poi esclamare: "chiaro! quando si usa questa istruzione nessun programma gira".

Il ciclo di esecuzione di questa istruzione, sarebbe a dire il tempo necessario ad essere eseguita, è il doppio di quello normale, questo significa che un PIC funzionante a 4 MHz impiega 2 ms al posto di uno. Questo perché l'architettura segmentata del PIC mentre esegue un'istruzione punta alla prossima da eseguire, che si suppone sia in sequenza. Però la goto salta da un'altra parte del programma, rompendo la sequenza, per cui l'istruzione seguente non è conosciuta, e si impiega un altro ciclo per ottenere la nuova istruzione.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
g o t o	k	

Operazione: salto incondizionato all'indirizzo segnato come k, che potrebbe essere un'etichetta.
Cicli: 2
Codice OP: 10 1kkk kkkk kkkk
Flags: nessuno

Caratteristiche dell'istruzione goto

ESEMPI CON L'ISTRUZIONE GOTO

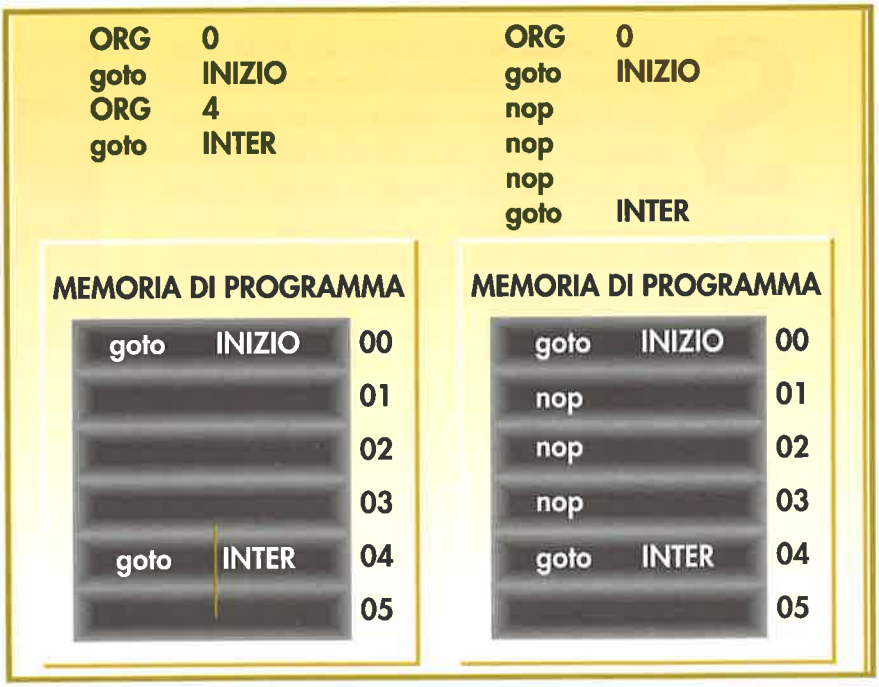
Nella figura possiamo osservare due usi dell'istruzione goto, il primo corretto e il secondo sbagliato. Come si può vedere nel quadro delle caratteristiche dell'istruzione, dopo il mnemonico della medesima si mette un parametro, che è indicato come letterale, e che normalmente è un'etichetta che rappresenta l'indirizzo di destinazione: dove si desidera andare. La programmazione strutturata, dice che se si salta da una subroutine a un'altra, bisogna andare all'inizio di questa e non a una parte interna della medesima, per non rompere la struttura del programma rappresentato dall'organigramma. Così se si considera il primo esempio dove l'etichetta 1 è all'inizio di una subroutine non c'è nessun problema; il problema c'è se, come nel secondo esempio, l'etichetta 1 è parte di un'altra subroutine.

```
goto        inizio - modulo 1          goto        mezzo-modulo
            o  o  o                                o  o  o
inizio-modulo 1  istruzione 1          inizio-modulo 1  istruzione 1
                 istruzione 2                           istruzione 2
                 istruzione 3                           istruzione 3
metà-modulo 1    istruzione 4          metà-modulo 1    istruzione 4
                 istruzione 5                           istruzione 5
                 goto  metà-modulo 1                    goto  inizio-modulo 1
                 istruzione 7                           istruzione 7
                 istruzione 8                           goto  inizio-modulo 2
                 goto  inizio-modulo 1                  goto  metà-modulo 1
```

Utilizzo corretto ed errato dell'istruzione goto.

L'istruzione NOP

L'istruzione nop è una delle più semplici da utilizzare, e nello stesso tempo è quella meno utilizzata. È semplice perché non ha parametri, e quindi non può essere impiegata male, ed anche se fosse utilizzata in modo indebito, l'unico inconveniente che potrebbe provocare sarebbe una perdita di tempo equivalente ad un ciclo di istruzione, dato che non fa niente e non varia nessun flag. Nel suo codice OP le due x indicano che in quella posizione si possono avere zeri o uno indistintamente.



Modi diversi di posizionare le istruzioni nella memoria di programma.

ESEMPI CON L'ISTRUZIONE NOP

La natura stessa dell'istruzione rende difficile immaginare un caso in cui sia necessario il suo utilizzo. Alcuni programmatori la utilizzano in sostituzione della direttiva ORG che, come ricordiamo, manda al compilatore l'indicazione che la posizione dell'istruzione successiva è all'indirizzo indicato dal suo parametro. Come abbiamo già visto, in questo modo si ottiene di mettere all'indirizzo 0 la prima istruzione del programma, cosa necessaria, dato che quando inizierà l'esecuzione del programma, è lì che si deve trovare la prima istruzione. Abbiamo anche un altro indirizzo speciale nella memoria dei codici: l'indirizzo 04. In questa posizione, il processore va a cercare la routine di servizio dell'interrupt, quando se ne produce uno, in modo che il suo posto non possa essere occupato da nessun'altra istruzione. Nell'esempio si mostrano due modi di realizzare questo, utilizzando l'istruzione nop e con la direttiva ORG. Anche se il risultato nella memoria dei codici può apparire diverso, l'esecuzione del programma è la stessa. In alcuni casi, inoltre, si utilizza questa istruzione nei cicli, come metodo per costruire delle routine di temporizzazione.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
n o p		

Operazione: è un'operazione che non fa nulla.
Cicli: 1
Codice OP: OP: 00 0000 0xx0 0000
Flags: nessuno

Caratteristiche dell'istruzione nop.

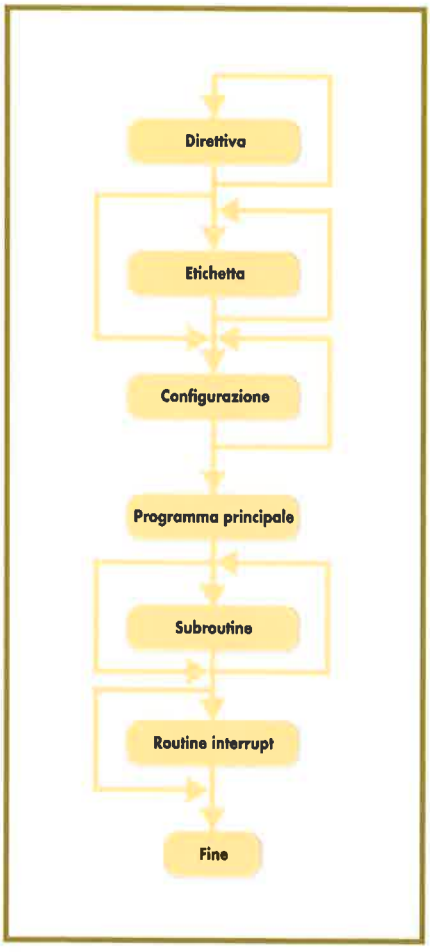
Programmazione: struttura di un programma

Quando si affronta la realizzazione di un programma, sia esso in assembler o in qualsiasi altro linguaggio di programmazione, molte volte non si sa bene da dove iniziare, soprattutto le prime volte. In questi casi potrebbe essere molto utile creare una specie di piantina che ci faciliti il lavoro, per evitare errori od omissioni indesiderate.

Questa piantina si ottiene facilmente, dato che un programma assembler, è composto da varie parti chiaramente differenziate. La tabella mostra in modo schematico queste parti. Spiegheremo per prima cosa alcune convenzioni utilizzate nella realizzazione di queste piantine. Nella colonna di sinistra scriveremo le etichette delle routine e delle subroutine, nella colonna centrale scriveremo le istruzioni vere e proprie del programma, che devono rispettare l'esatta nomenclatura prevista per il nostro microcontroller. Infine, nella colonna di destra, scriveremo i commenti ed i nomi che vorremo dare alle varie sezioni del programma.

LIST RADIX	P=tipo di processore Sistema di numerazione	Direttive
nome	EQU valore	Etichette
ORG goto ORG goto ORG etichetta-inizio	0 etichetta inizio 4 routine-inter 5 istruzioni _____ _____	Configurazioni
ciclo	_____ _____ _____ goto ciclo	Programma principale
routines	_____ _____ _____ return (rtlw)	Subroutines
routine-inter	_____ _____ retfie	Routine di trattamento dell'interrupt
END		Direttiva di fine programma

Parte di tutto il programma compilatore per il PIC.



Organigramma della parte di un programma obbligatorio, opzionale e multiplo.

```
                LIST        P=16F84
                INCLUDE     "Regx84.inc"
                INCLUDE     "Macros.inc"

LED             EQU         1
LAMPEGGIATORE   EQU         6
                ORG         0
                goto        INIZIO
                ORG         4
                goto        INTER
INIZIO
                call        CONFIGU
                avanti

CICLO
                GOTO        CICLO           ;Ciclo infinito
CONFIGU
                bsf         STATUS,RP0
                movlw       b'00000001'
                movwf       TRISB
                movlw       b'10000'
                movwf       TRISA
                movlw       b'10110000'     ;INTERRUPT, TMR0 E RB0 ABILITATI
                movwf       INTCON
                movlw       b'0111000'      ;TMR0 RA4;INTEFRONTE DI SALITA
                                            ;TMR0 SENZA PRESCALER
                movwf       OPTION_REG
                bcf         STATUS,RP0
                bcf         PORTB,LAMPEGGIATORE ;CI ASSICURIAMO CHE SIA SPENTO
                bcf         PORTB,LED
                movlw       F6              ;10 IMPULSI PER IL TIMER
                movwf       TMR0
                return
INTER           stop                        ;ROUTINE DI INTERRUPT. Se chiamata
                bifsc       INTCON,T0IF
                goto        IMPULSI
                bifsc       INTCON,INTF
                retfie

BUMPER
                bsf         PORTB,LED       ;ATTIVIAMO IL LAMPEGGIATORE
                bcf         INTCON,INTF     ;CANCELLIAMO IL FLAG DI INTERRUPT
                retfie

IMPULSI
                bsf         PORTB,LAMPEGGIATORE ;ATTIVIAMO IL LAMPEGGIATORE
                bcf         INTCON,T0IF     ;CANCELLIAMO IL FLAG DI INTERRUPT
                retfie

END
```

Programma esempio che mostra diverse parti tipiche.

SIGNIFICATO DI OGNI PARTE

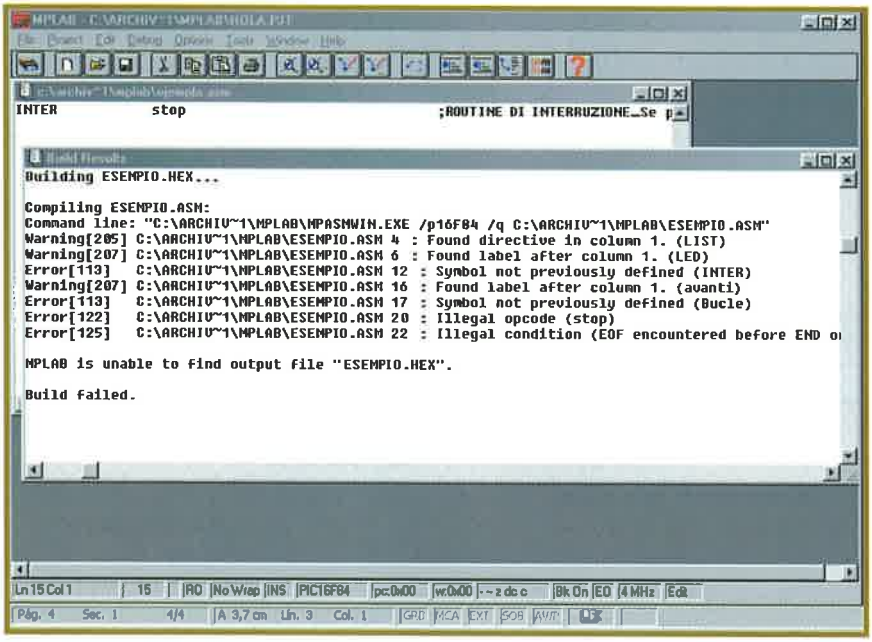
Si inizia con direttive per il processore, in pratica, "ordini" che il compilatore dovrà tenere conto al momento di realizzare il file esadecimale, a partire dalle sorgenti. Queste direttive non si traducono in istruzioni, dato che non vanno ad occupare indirizzi della memoria di programma. Nello schema appaiono due direttive molto frequenti, anche se ne esistono molte altre: la prima informa del tipo di PIC, e la seconda determina il codice standard dei dati, cioè se i dati saranno in esadecimale, decimale o in binario.... Come abbiamo visto in alcuni esempi, di seguito arrivano le definizioni delle etichette per riferirci a registri, che sono valori, con un nome al posto di un numero.

Nella zona di configurazione si inizializzano i registri che si utilizzeranno nel programma. Questo si fa nella zona chiamata "etichetta inizio" e questa configurazione si realizza con istruzioni normali. Prima di questo, considerata anche come configurazione, la direttiva ORG indica al compilatore, con differenti parametri, dove deve collocare le differenti parti del programma nella memoria del codice. Nella posizione 0 è obbligatorio mettere la prima istruzione che dovrà essere eseguita, normalmente un'etichetta come inizio, start, etc., o la chiamata a questa.

Nella posizione 4, come vedremo più avanti, bisogna porre l'inizio della routine che gestirà l'interrupt, il vettore di interrupt.

Il modo ideale per progettare un programma è situare – dopo tutto quanto menzionato in precedenza – un ciclo da cui si chiamano le routines che realmente realizzano il lavoro, normalmente in modo ciclico.

Questo è ciò che normalmente è etichettato come programma princi-



Modo di avviso degli errori da parte dell'assembler.

pale; anche se è una buona forma di programmazione, non è l'unica, e comunque, non è obbligatoria.

In seguito troveremo le differenti subroutines, da cui si rientra con l'istruzione retlw o return, a seconda del caso, e anche la routine di trattamento dell'interrupt. Si termina con la direttiva di fine del programma END.

PROGRAMMA MINIMO

Tuttavia, non tutte le parti sono obbligatorie, per cui può succedere che qualcuna non ci sia. Al limite possiamo iniziare con la direttiva LIST, non definire etichette, lavorare con gli indirizzi, mettere solo una ORG con parametro 0, che non configuri nulla e non utilizzare subroutines. Obbligatoriamente dovremo terminare con la direttiva END.

Comunque questo non è abituale, poiché lo schema in genere viene utilizzato al completo e molte di queste parti si ripetono diverse volte. L'organigramma mostra oltre all'ordine delle differenti parti di un programma, quelle che possono essere saltate e quelle che possono essere ripetute.

Se seguiamo le frecce vediamo che quelle centrali indicano la sequenza delle diverse parti. Le frecce a sinistra indicano la possibilità di saltare il quadro a cui girano intorno, passando alla parte seguente. Le frecce di destra, che puntano verso l'alto, indicano che la parte può essere ripetuta tutte le volte necessarie. Così vediamo che sono obbligatorie alcune direttive, alcune configurazioni, il programma principale e la direttiva di fine programma. Le uniche parti che devono essere uniche, sono: il programma principale, la routine di trattamento dell'interrupt e la fine del programma.

ESEMPIO PRATICO

Il programma compilatore mostrato come esempio contiene tutti gli elementi visti, anche se la sua struttura non è perfettamente uguale allo schema iniziale – così come abbiamo detto – la soluzione ad un problema non è unica. Si inizia con tre direttive una di queste ripetuta. In seguito vengono le definizioni delle due etichette. Facendo la configurazione, abbiamo una parte che non effettua direttamente fino a quando non si chiama una routine per realizzarla (CONFIGU), nella quale si configurano 4 registri, e si inizializzano tre variabili. Le parole "avanti" e "stop" sono una specie di subroutine chiamate macro, che si studieranno in sezioni successive. Il programma principale consiste unicamente in un ciclo infinito da cui si aspetta ad uscire mediante interrupt.

L'interrupt inizia con l'etichetta INTER, dove, al contrario di cosa può apparire, si lavora con una sola routine di trattamento dell'interrupt anche se è divisa in vari rami dovuti al fatto che possono esistere diverse cause che la provocano. La fine del programma è segnata dalla direttiva END.

Se siete stati capaci di identificare ognuna delle parti, anche se con qualcuna di esse non abbiamo mai lavorato, siete già in grado di realizzare il vostro programma, che sarà sempre più complesso, man mano che andremo avanti.

SINTASSI DEI PROGRAMMI

I programmi sorgente possono essere scritti con qualunque editor di testo a caratteri ASCII.

Guardando nella struttura che segue, vedremo le diverse parti che possono formarlo.

Classificheremo ora il suo contenuto in altro modo, cioè come tipo di dati che possono contenere ognuna di queste parti. Così il compilatore, durante l'esecuzione, verificherà che i dati di tutte le linee del file siano di uno di questi quattro tipi: etichette, mnemonici, ope-

randi o commenti. Questi dati devono assecondare a norme di scrittura e a norme di posizione relativa che occupano gli uni rispetto agli altri.

Come regola generale, per tutti questi, la loro scrittura non deve eccedere i 255 caratteri sulla stessa linea.

ETICHETTE

Una etichetta deve iniziare nella colonna 1 e può essere seguita da due punti (:), spazi, tabulazioni e salti in linea. Possono contenere fino a 32 caratteri, che possono essere alfanumerici, e includere un tratto basso e il segno di punto interrogativo. Però il primo carattere non può essere numerico né di segno.

Come standard si differenziano le maiuscole e le minuscole, di modo che l'etichetta "ciclo" e "CICLO" siano differenti, a meno che non si affermi il contrario mediante un comando speciale. Se si utilizzano i due punti per terminare una dichiarazione di una etichetta, questi non sono presi come parte dell'etichetta, ma come un operatore. Nella figura le etichette definite alla sinistra sono corrette, e quelle a destra sono sbagliate. Lo stesso si può dire del resto degli esempi.

MNEMONICI

Gli mnemonici delle istruzioni, le direttive del compilatore e le chiamate alle macro devono iniziare a partire dalla colonna 2. Se l'istruzione segue una etichetta nella medesima linea, deve essere messa tra due punti, spazi in bianco o tabulati. Si possono scrivere in minuscolo o maiuscolo indistintamente.

movf LIST avanti etichetta1 movf Etichetta2:movf _etichetta3? Movf Etichetta_4	movf LIST avanti Etichetta5movf

Mnemonici, direttive e chiamate a macro corrette e sbagliate.

movf STATO,W etichetta1 movf STATO,W Etichetta2:movlw 9	movf STATO,W STATO,W etichetta6 movf ESTADOW

Operandi formulati in maniera corretta e scorretta.

OPERANDI

Devono essere separati dai mnemonici da uno o più spazi o tabulazioni. Se una istruzione ha più di un operando, questi devono essere separati da una virgola.

COMMENTI

Tutto quello che si pone dopo un punto e virgola (;) è trattato come un commento fino alla fine della linea. Se il commento occupa più di una linea dovremo mettere un punto e virgola all'inizio di ognuna di esse (come possiamo vedere negli esempi pubblicati). Quando il compilatore incontra errori di questo tipo, ci avvisa e non genera il file che doveva essere eseguito.

etichetta1 etichetta2: _etichetta3? Etichetta_4	Etichetta5 ?etichetta6 7etichetta

Etichette formulate in maniera corretta e scorretta.

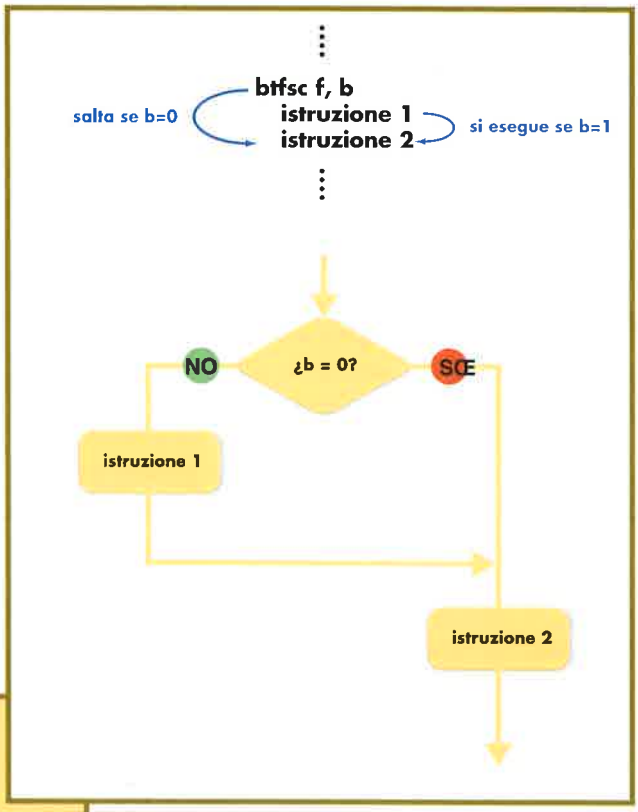
etichetta1 movf STATO,W ;movimento Etichetta2:movlw 9 ;movimento di un ;valore letterale	STATO,W mnemonico etichetta6 movf STATOW ;tutto sbagliato

Commenti formulati bene e formulati male.

Scheda delle istruzioni: l'istruzione BTFSC

Questa è una delle istruzioni chiamate "istruzioni di salto condizionato" che ha il PIC. Anche se tante consonanti di seguito appaiono difficili da ricordare dato che non hanno significato, in realtà rispondono alle sigle di quello che fa: testa (t) un bit (b) di registro (f è la denominazione che stiamo dando ai registri in generale, prima di sostituirla con il valore corretto caso per caso) e salta (s) se è zero (c). Questo salto è piuttosto un passo in avanti, dato che ignora solo un'istruzione che, pertanto, non viene eseguita.

Questa è la prima istruzione fra quelle viste finora, il cui ciclo di esecuzione può essere 1 o 2. Così se il salto non si realizza il PC conosce l'indirizzo dell'istruzione da eseguire in successione, che è quello seguente alla btfsc; però, se deve effettuare il salto, dato che il valore del bit è uguale a zero, il PC dovrà calcolare il nuovo indirizzo, per cui sarà necessario un ciclo addizionale.



Schema del comportamento dell'istruzione btfsc a seconda del valore di un bit.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
b t f s c	f	b

Operazione: nel caso che il bit segnalato dal registro abbia valore 0 l'istruzione seguente è ignorata e nel caso contrario no.
Cicli: 1-2
Codice OP: 01 10bb bfff ffff
Flags: nessuno

Caratteristiche dell'istruzione btfsc.

ESEMPI CON L'ISTRUZIONE BTFSC

Dobbiamo tenere conto che quando si compie la condizione e si esegue il salto, si esegue un'unica istruzione, però se la condizione non si compie, si eseguono entrambe le istruzioni perché dopo la prima arriva la seconda. Questo è indicato nello schema generale della figura, dove le frecce rappresentano la sequenza che seguono le istruzioni in entrambi i casi. I due grafici mostrano la stessa cosa, il primo nella forma che avrà il programma e il secondo in forma grafica.

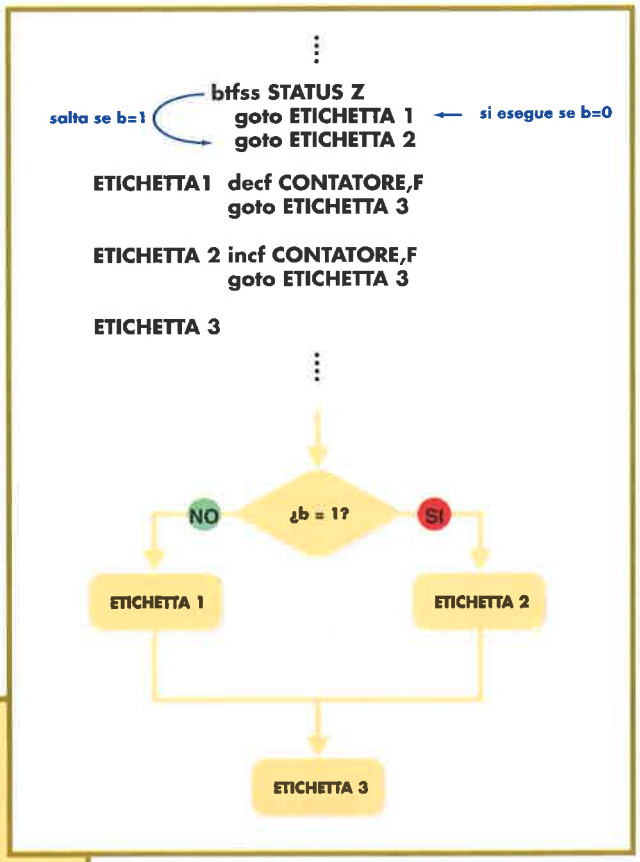
Scheda delle istruzioni: l'istruzione BTFSS

E' analoga alla precedente. Seguendo di nuovo la sua sigla vediamo che cambia solo l'ultima lettera, questa istruzione testa (t) un bit (b) di un registro (f) e salta (s) se è un uno (s è l'iniziale in inglese della parola "set" o mettere a uno).

Per quanto riguarda i cicli di istruzione si comporta come l'istruzione precedente.

ESEMPI CON L'ISTRUZIONE BTFSS

In questa occasione sono stati sostituiti i dati generali per ottenere un esempio reale. In esso, a seconda del valore del flag Z del registro di Stato, si indirizza l'esecuzione del programma ad una parte oppure ad un'altra. L'istruzione fa la stessa cosa della btfsc, però salta nel caso che il valore sia a uno e il salto riguarda solo l'istruzione immediatamente successiva. Nonostante questo, come possiamo vedere, l'organigramma è variato in modo sostanziale rispetto al caso precedente.



Schema del comportamento dell'istruzione btfss, a seconda del valore di un bit, mostrato con un esempio.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
b t f s s	f	b

Operazione: nel caso che il bit segnalato dal registro abbia valore 1 l'istruzione seguente è ignorata e nel caso contrario no.
Cicli: 1-2
Codice OP: 01 11bb bfff ffff
Flags: nessuno

Caratteristiche dell'istruzione btfss.

Questo non è dovuto alla funzione che realizza l'istruzione btfss (potrebbe essere così anche per la btfsc), ma dal fatto che fra tutte le istruzioni che si potevano utilizzare dopo la btfss (o btfsc) è stata scelta la goto.

Con questa istruzione, come abbiamo già visto, si spezza il flusso sequenziale di un programma e si salta lì dove indica l'etichetta. Questo è il modo di utilizzo più comune delle istruzioni btfsc e btfss.

Programmazione: strutture di controllo

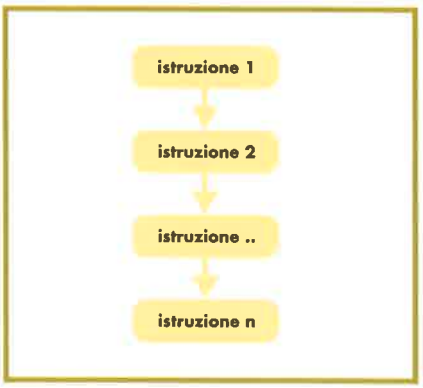
La programmazione modulare e strutturata diminuisce il costo di realizzazione e mantenimento dei programmi, e aumenta la sua affidabilità e leggibilità. Per questo si costruiscono moduli o pezzi di programmi seguendo certe regole. I moduli a loro volta, sono formati da strutture, le quali possono essere di tre tipi: sequenziali, condizionali e ripetitive. L'organigramma della figura a lato, rappresenta un programma con tre tipi di strutture, che sono quelle abituali.

STRUTTURE SEQUENZIALI

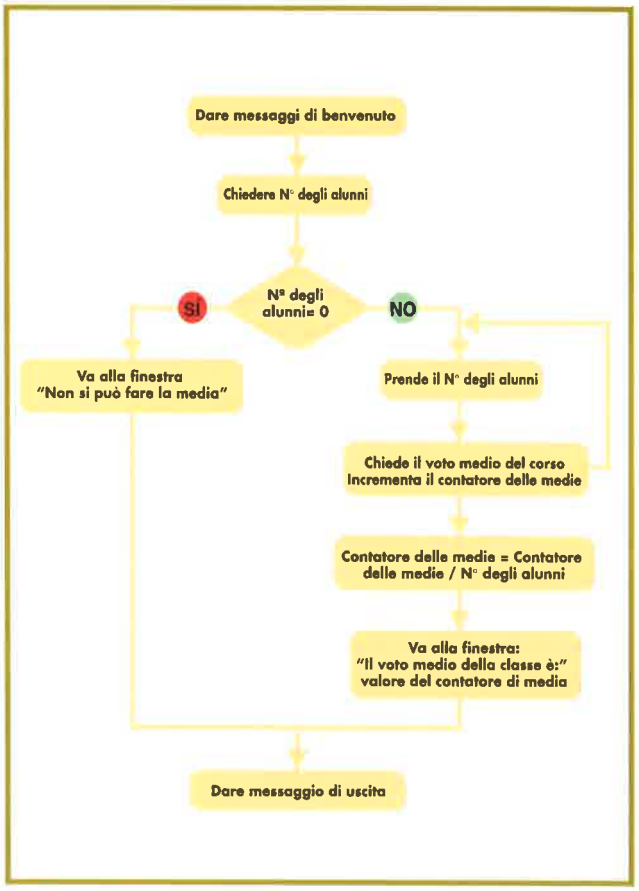
Le strutture sequenziali, o concatenate, si costruiscono scrivendo le istruzioni nell'ordine in cui più tardi saranno eseguite, sarebbe a dire una dietro l'altra.

Graficamente si può rappresentare come una serie di quadri in sequenza, ognuno dei quali è un'istruzione.

Le frecce di unione segnano la sequenza logica di esecuzione, nella quale non possiamo omettere nessuna azione.



Rappresentazione delle strutture sequenziali in forma grafica.



Organigramma di un programma con tre tipi di strutture di controllo.

STRUTTURE CONDIZIONALI

Normalmente non è sufficiente porre le istruzioni una dietro l'altra, nel caso in cui esse dipendano da eventi esterni bisogna poter prendere una decisione, e scegliere un percorso fra due possibili. È quello che viene denominato struttura condizionale o alterna-

Rappresentazione delle strutture condizionali IF-THEN-ELSE.

Rappresentazione della struttura condizionale IF-THEN-ELSE con un solo ramo.

tiva, che in pseudocodice si conosce come IF-THEN-ELSE se la condizione ha solo due ramificazioni, o CASE, se la scelta è multipla. La decisione può essere determinata dal valore di una variabile, per una chiamata di un evento esterno, ecc.

Il grafico indica i due cammini possibili di una IF-THEN-ELSE. Come possiamo vedere all'interno di una struttura condizionale può esistere, a sua volta, una struttura sequenziale, che possiamo sostituirla con il suo schema.

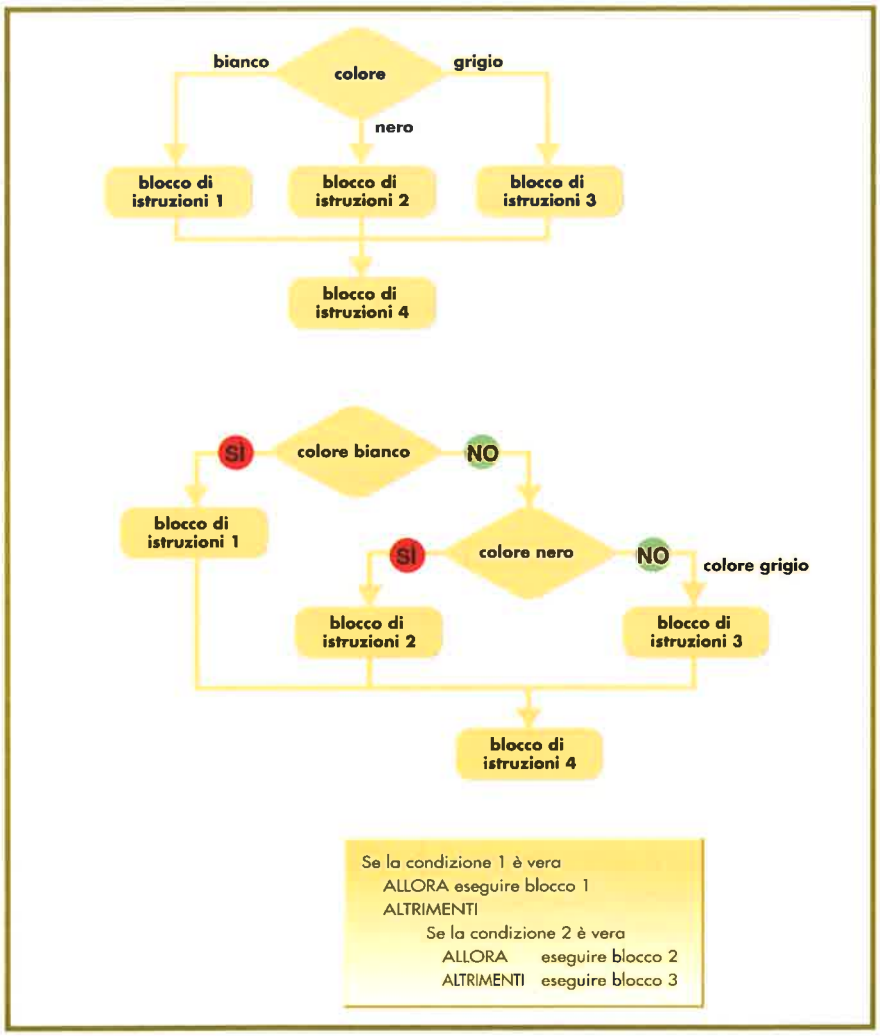
A seconda del valore della condizione, si eseguirà il blocco di istruzioni 1 o 2, però mai entrambe. Anche se l'esecuzione del blocco di istruzioni 3 non è pertinente alla struttura condizionale in sé, è stato introdotto nel grafico perché dopo una biforcazione si ritorna sempre ad un programma comune, anche se alla fine, per terminare in un unico punto.

Potrebbe succedere che manchi uno dei blocchi di istruzioni, in questo caso, a seconda della condizione, si eseguirà un blocco di istruzioni o non si farà nulla. Nella figura in alto, è stata rappresentata la mancanza del blocco di istruzione, nel caso di "condizione falsa", però potrebbe verificarsi anche il caso contrario: non fare nulla se si compie tale condizione.

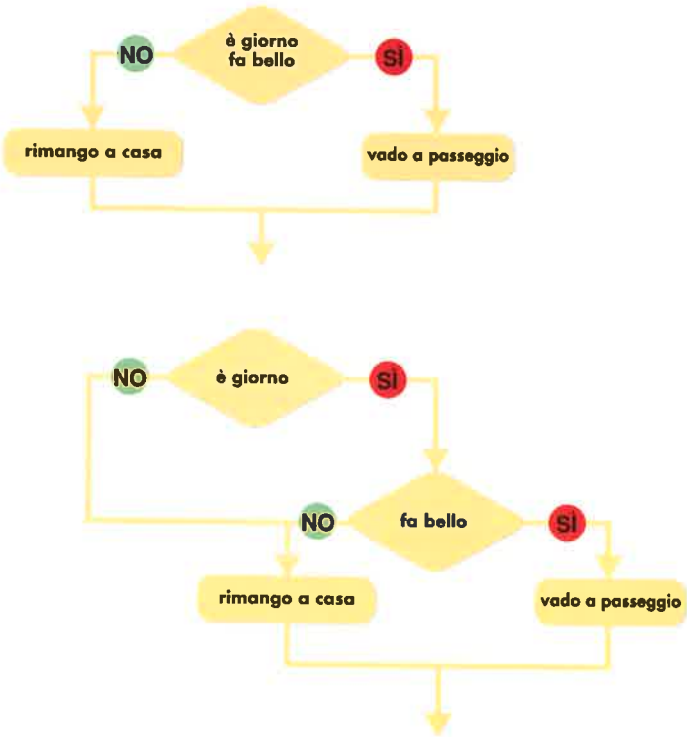
Potrebbe succedere che la decisione da prendere non sia legata a due sole risposte, ma ad un numero più alto, come succede nel CASE. In questo caso non abbiamo una traduzione immediata nell'assembler del PIC, per cui dobbiamo evitare questo schema, e per questo la cosa migliore è trasformarlo in una struttura condizionale già vista, però in forma annidata.

Nella figura corrispondente viene indicato questo annidamento con un esempio. Si vuole determinare qual è il colore scelto fra tre possibili. Il primo impulso è chiedere direttamente qual è il colore, restando alla prima figura.

Se al posto di questo immaginiamo il gioco del "vero o falso", dove le sole risposte alle domande possono essere "si" o "no", andiamo alla seconda struttura, in cui il risultato finale è lo stesso. Anche se abbiamo fatto due domande, l'annidamento può essere largo quanto vogliamo.



Esempio di trasformazione di una struttura CASE in varie IF-THEN-ELSE annidate.




```
Se la condizione 1 è vera
   ALLORA
      Se la condizione 2 è vera
         ALLORA          eseguire blocco 1
         ALTRIMENTI      eseguire blocco 2
ALTRIMENTI eseguire blocco 2
```

Implementazione dell'operatore logico AND con strutture IF-THEN-ELSE annidate.

Osserviamo che in questa occasione le alternative vere si trovano alla sinistra della figura, perché lo schema risultante è più comodo da vedere e da realizzare. L'altro caso in cui sono necessarie le strutture IF-THEN-ELSE annidate è quando si vuole analizzare un blocco di istruzioni solo se si compiono alcune condizioni nello stesso tempo. In questo caso, come nel precedente, dobbiamo evitare di fare una domanda, e al posto di questa, una volta verificata la veridicità di una condizione, formulare la domanda seguente. Questo rappresenta l'implementazione di una AND logica. Le altre operazioni logiche, inoltre, possono anche essere rappresentate con strutture annidate.

STRUTTURE RIPETITIVE

Come indica il nome stesso, le strutture ripetitive servono per ripetere un determinato numero di volte un blocco di istruzioni. Nuovamente, quindi, all'interno di esse devono essere contenute le strutture sequenziali. Il numero di volte che si ripete il blocco può essere noto, o dipendente da una variabile o da un evento. A seconda dell'autore consultato, o del linguaggio di programmazione che si prende come riferimento, il nome di questo tipo di struttura varia, di modo che conviene fissarci sull'essenza di quello che fa, piuttosto che sul nome che riceve. Se il numero di volte che deve essere eseguita un'istruzione è fisso, stiamo parlando di un ciclo, previsto nella maggior parte dei linguaggi di programmazione, e che di solito si chiama FOR; si ripete qualcosa per un numero di volte. Anche all'interno delle strutture ripe-

Struttura in pseudocodice	Struttura adattata per il compilatore del PIC

FOR numero volte DO
Istruzione 1
Istruzione 2
Istruzione 3
...
Istruzione n
END FOR
	Contatore = numero di volte
	Etichetta Istruzione 1
	Istruzione 1
	Istruzione 2
	Istruzione 3
	...
	Istruzione n
	Decremento contatore
	Se il valore del contatore è 0
	ALLORA va alla Fine
	ALTRIMENTI va a Etichetta
	Fine

Struttura FOR con pseudocodice e suo adattamento per il compilatore del PIC.

Struttura in pseudocodice	Struttura adattata per il compilatore del PIC

WHILE condizione vera DO
Istruzione 1
Istruzione 2
Istruzione 3
...
Istruzione n
 	
ENDWHILE
	Chiedere Se la condizione è vera
	ALLORA andare a Etichetta
	ALTRIMENTI andare alla Fine
	Etichetta Istruzione 1
	Istruzione 2
	Istruzione 3
	...
	Istruzione n
	Andare a Chiedere
	Fine

Struttura WHILE-DO in pseudocodice e suo adattamento per il compilatore del PIC.

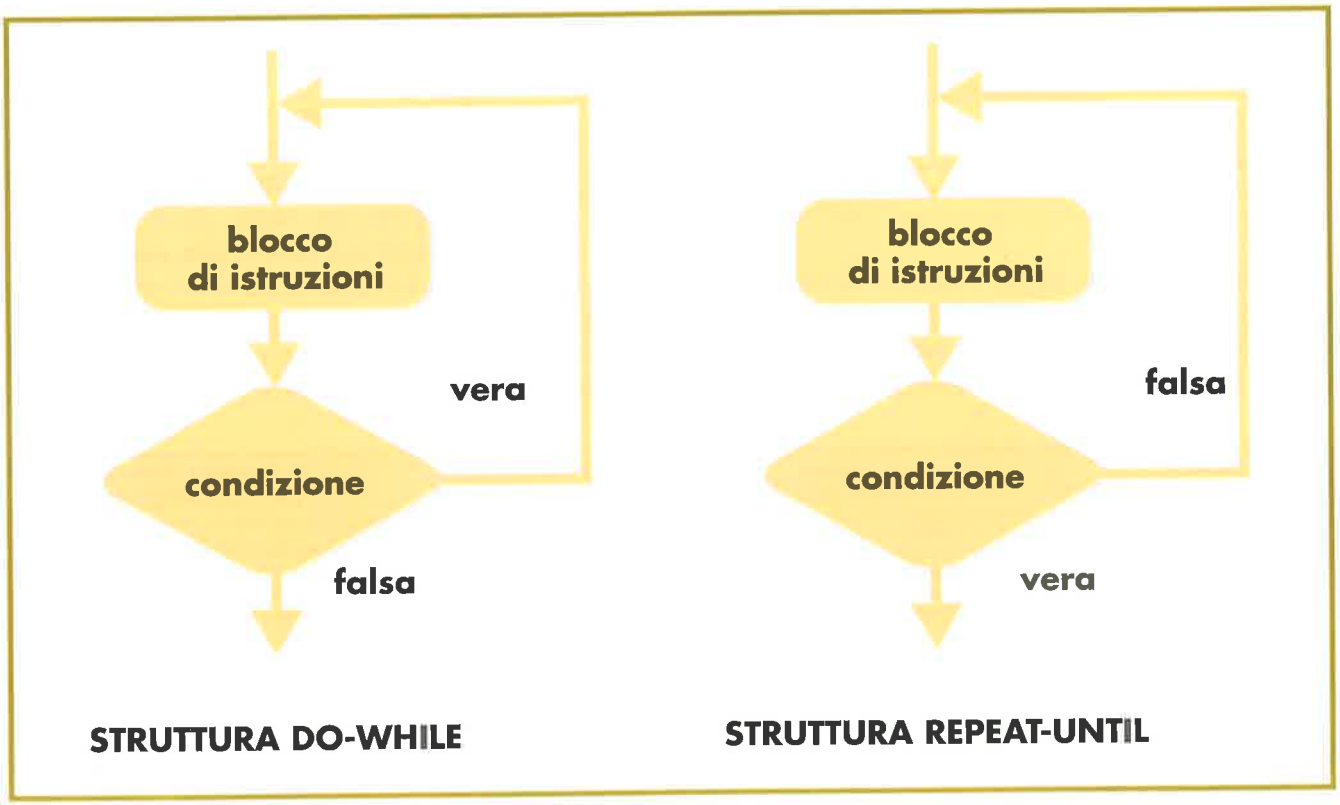
Struttura in pseudocodice	Struttura adattata per il compilatore del PIC
DO Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n WHILE condizione vera	Etichetta Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n Chiedere Se la condizione è vera ALLORA andare a Etichetta ALTRIMENTI andare alla Fine Fine

Struttura DO-WHILE in pseudocodice e suo adattamento per il compilatore del PIC.

Struttura in pseudocodice	Struttura adattata per il compilatore del PIC
REPEAT Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n UNTIL condizione vera	Etichetta Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n Chiedere Se la condizione è vera ALLORA andare a Etichetta ALTRIMENTI andare alla Fine Fine

Struttura REPEAT-UNTIL in pseudocodice e suo adattamento per il compilatore del PIC.

titive si utilizza una condizione per controllare la fine del ciclo, ma non deve essere confusa con le strutture condizionali. Quando si deve compiere una condizione per uscire dal ciclo si dice che è una WHILE-DO (mentre si compie una condizione fare qualcosa), o una DO-WHILE (fare qualcosa mentre si compie una condizione) oppure una REPEAT-UNTIL (fare qualcosa fino a che si compie la condizione). È infrequente che i linguaggi di programmazione abbiano le tre varianti allo stesso tempo, dato che una si può trasformare in un'altra. I tre tipi di cicli possono tradursi facilmente nell'assembler del PIC. La differenza fra la WHILE-DO e la DO-WHILE è che nella prima se non si compie la condizione, le istruzioni non si eseguono. Nella DO-WHILE invece, si eseguono le istruzioni la prima volta, e se si compie la condizione, si continuano ad eseguire. La REPEAT-UNTIL è simile alla DO-WHILE, dato che il blocco di istruzioni si ripete almeno una volta, e continua a ripetersi fino a che non si compie la condizione. Come possiamo osservare, se si comparano gli schemi, le differenze sono nella negazione della condizione, dato che nella DO-WHILE si suppone che sia vera e si esce quando si converte in falsa, e nella REPEAT-UNTIL si considera falsa e si esce quando diventa vera.

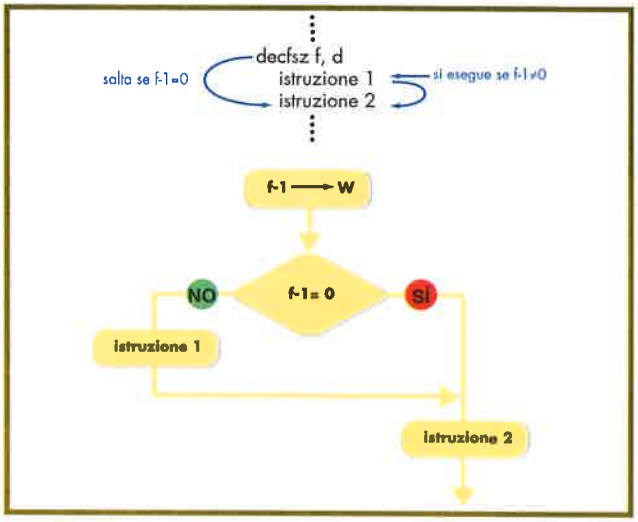


Schema di funzionamento delle strutture DO-WHILE e REPEAT-UNTIL

Scheda delle istruzioni: l'istruzione DECFSZ

Ancora una volta le lettere che compongono il mnemonico di questa istruzione ci indicano la funzione che realizza. La decfsz decrementa (dec) un registro (f) e salta (s) se il risultato è zero (z). Il risultato del decremento può essere memorizzato nello stesso registro che si prende come base per questa operazione, oppure nel registro di lavoro W, nel cui caso il valore del registro non varia.

Forse ricorderete un'istruzione simile chiamata btfsc, in quell'istruzione per fare riferimento allo zero si utilizza la c al posto della z. Questo perché esiste una differenza sostanziale, dovuta al fatto che la btfsc verifica se è a zero un bit ("clear" o cancellato), mentre nella decfsz si verifica se un registro completo ha assunto valore zero (00h). Come la btfsc, il salto che produce è di una sola istruzione, e nel caso in cui il salto venga eseguito, la sua esecuzione impiegherà due cicli anziché uno.



Schema di come si comporta l'istruzione decfsz, a seconda del risultato dell'operazione, prendendo come secondo parametro il valore d = 0.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
decfsz	f	d

Operazione: decrementa di una unità il contenuto del registro **f**. Nel caso in cui il risultato di questo decremento sia **0** l'istruzione seguente è ignorata, e nel caso contrario no. Se **d= 0** il risultato del decremento si memorizza nel registro di lavoro **W** e se **d=1** si memorizza nello stesso registro **f**.

Cicli: 1-2

Codice OP: 00 1011 dfff ffff

Flag: nessuno

Caratteristiche dell'istruzione decfsz.

ESEMPI CON L'ISTRUZIONE DECFSZ

Lo schema rappresentato ricorda quello già visto per l'istruzione btfsc, anche se con sostanziali differenze. La più importante, per quanto ci riguarda, è che l'istruzione non solo verifica un valore per decidere se saltare o meno, ma realizza anche un'operazione, il decremento, e salta a seconda del risultato generato. Se per la funzionalità del programma, ci interessa solo l'operazione di salto, non è necessario memorizzare il risultato, quindi lo porteremo nel registro di lavoro, senza eseguire ulteriori operazioni su di esso. La stessa funzione che compie questa operazione si può realizzare mediante la combinazione delle istruzioni decf e la successiva verifica del flag Z con la btfsc.

Scheda delle istruzioni: l'istruzione INCFSZ

L'istruzione è simile alla precedente, sia nella struttura che nell'uso, anche se al posto del decremento di un registro questa realizza il suo incremento, e in ugual modo salta se il risultato è zero. Nel caso della decfsz, può apparire più chiaro, però anche incrementando un registro si può passare per lo zero, dato che dopo il valore massimo (FF) si passa al minimo (00) con l'incremento successivo.

Struttura di pseudocodice	Struttura utilizzando decfsz	Struttura utilizzando incfsz
FOR numero di volte D0 Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n ENDFOR	movlw NUMERO movwf CONTATORE ETICHETTA Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n decfsz CONTATORE goto ETICHETTA goto FINE FINE	movlw C2 (NUMERO) movwf CONTATORE ETICHETTA Istruzione 1 Istruzione 2 Istruzione 3 ... Istruzione n incfsz CONTATORE goto ETICHETTA goto FINE FINE

Traduzione della struttura FOR mediante le istruzioni decfsz e incfsz.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
incfsz	f	d

Operazione: incrementa di una unità il contenuto del registro **f**. Nel caso in cui il risultato di questo incremento sia **0** l'istruzione seguente è ignorata, e nel caso contrario no. Se **d=0** il risultato dell'incremento si memorizza nel registro di lavoro **W** e se **d=1** si memorizza nello stesso registro **f**.

Cicli: 1-2

Codice OP: 00 1111 dfff ffff

Flag: nessuno

Caratteristiche dell'istruzione incfsz.

ESEMPI CON L'ISTRUZIONE INCFSZ

Lo schema di funzionamento di questa istruzione è lo stesso di quello per la decfsz, anche se al posto delle sottrazioni abbiamo le somme. Entrambe le istruzioni si utilizzano nella costruzione dei cicli in cui si deve ripetere un insieme di istruzioni un determinato numero di volte.

Nella figura è mostrata la traduzione della struttura FOR a istruzioni del PIC, impiegando la incfsz e la decfsz. Notare che l'unica differenza, a parte il cambio di un'istruzione con un'altra, è nel valore che deve essere introdotto nel registro che si usa come contatore, che nel caso dell'istruzione è direttamente il numero di volte che si vuole ripetere una sequenza, mentre per la incfsz, per contare lo stesso numero di volte, bisogna introdurre il complemento a 2 di detto valore.

comunque se utilizzate in modo coerente, i vantaggi sono maggiori degli svantaggi.

Normalmente una subroutine comunica con il programma principale mediante lo scambio di una serie di valori. Quando si chiama il subprogramma, questo riceve gli argomenti necessari alla sua esecuzione, e restituisce dei risultati. È consigliabile che le variabili a cui accedono sia il programma che il subprogramma siano indipendenti, cioè di ambito locale, di modo che al momento della chiamata al procedimento, siano le uniche variabili interessate. Però questo non è possibile con il compilatore del PIC, dato che la chiamata a subroutine (istruzione call) ammette come parametro solo il nome della subroutine stessa. Una soluzione intermedia fra lo scambio di parametri e la non comunicazione fra programma e subprogramma, consiste nell'utilizzo di variabili globali, a cui possano accedere entrambi. Così prima della chiamata ad una subroutine, verranno introdotti nei registri predisposti, i valori da utilizzare con le subroutine, e prima di ritornare dalla subroutine, saranno aggiornati i registri che devono contenere i risultati del lavoro svolto, oppure entrambe le cose se sono necessarie. Un esempio tipico di questo è l'uso delle tabelle, che abbiamo già visto parlando dell'istruzione retlw. In questo caso il registro che funzionava da intermediario era il registro di lavoro W.

```
Nome_macro      macro
                Istruzione-1
                Istruzione-2
                Istruzione-3

                Istruzione-(n-1)
                Istruzione-n
                endm

Avanti          macro
                movlw       b'00001101'
                movwf       PORTA
                endm
```

Struttura di una macro ed esempio del suo utilizzo.

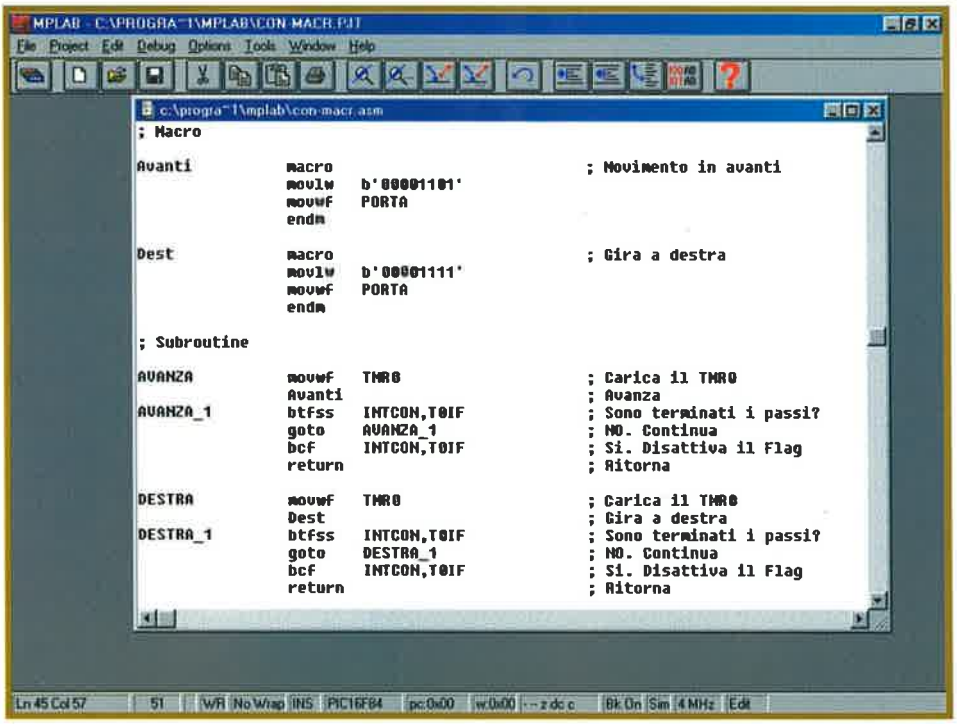
MACRO

Un altro modo di semplificare un programma consiste nell'utilizzare delle macro. Una macro definisce un insieme di istruzioni a cui si assegna un nome. Nel programma principale basterà inserire il nome della macro nel punto che ci interessa, e le istruzioni che la compongono saranno eseguite.

Nome_macro esprime il nome della macro che in seguito verrà inserita nel programma principale per aggiungere tutte le istruzioni che appaiono sotto questo nome fra le parole chiave macro e endm.

DIFFERENZE FRA MACRO E SUBROUTINE

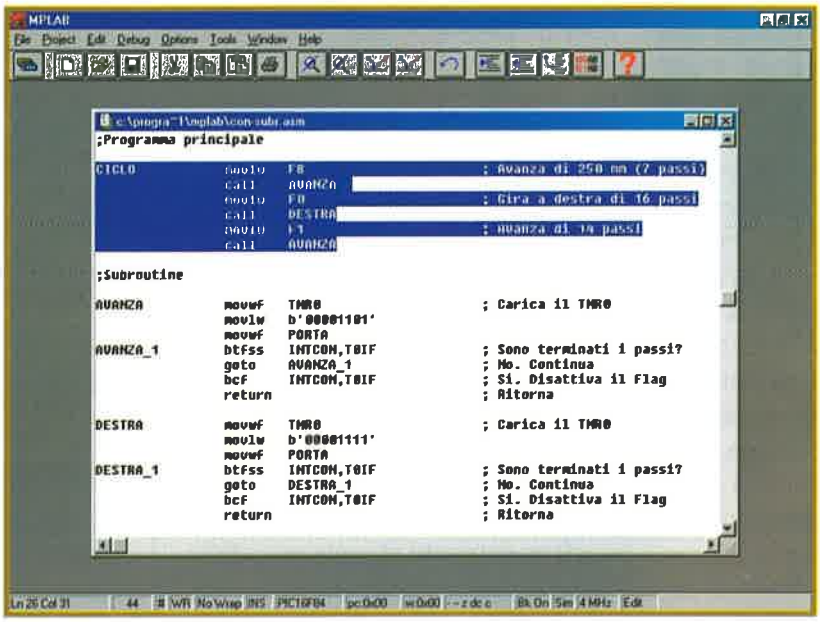
Nonostante l'utilizzo delle macro e delle subroutine si assomiglino, fra loro c'è una differenza fondamentale. Quando si compila un programma fatto con subrouti-



Sezione dello stesso programma, però con l'utilizzo di subroutine e di macro nella stesura.

Programmazione: subroutine, macro e recursività

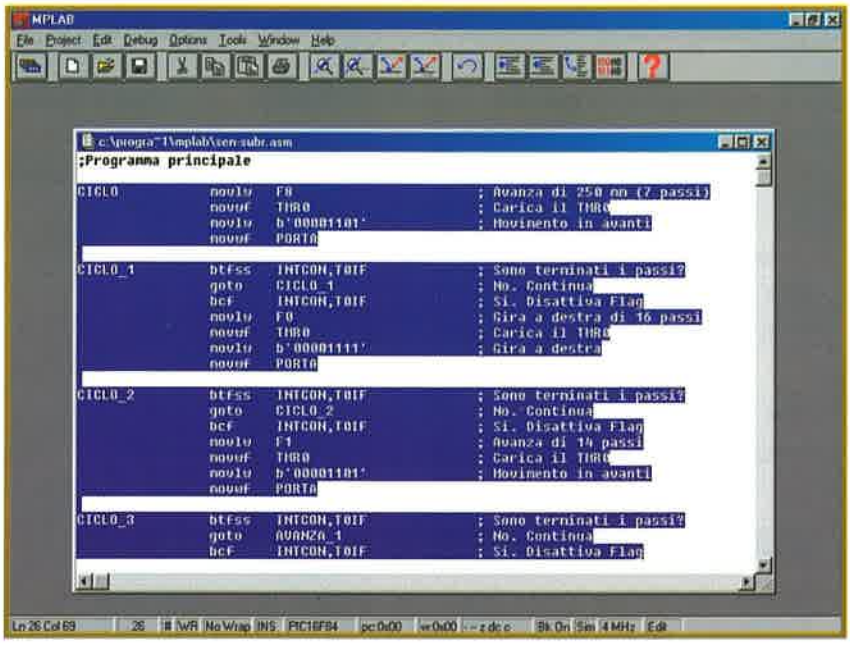
La necessità di programmare in forma strutturata e modulare è stata spiegata nella sezione precedente, così come le strutture che formano parte di questi moduli. Ora tratteremo i differenti moduli esistenti, e quali sono le differenze fra di loro. In ogni caso la modularità deve sempre permettere che un programma da principio complesso, si divida in parti più semplici, riutilizzabili, e pertanto più facili da mettere a punto e da modificare.



Il programma precedente, scritto senza l'utilizzo dei moduli.

SUBROUTINE

La decisione di utilizzare o meno le subroutine dipende dal tipo di programma che vogliamo realizzare. Quando un'azione si ripete molte volte, come può essere il "ritardo di un tempo determinato", non è consigliabile ripetere tutto il codice ogni volta che bisogna usarlo. Questo potrebbe portare gravi inconvenienti, il primo è che stiamo sprecando spazio nella memoria di programma, e questo è pericoloso, tenendo conto che il PIC16F84 dispone solo di 1024 linee di programma. Il secondo è che un piccolo cambio in una di queste parti di programma che si ripete, comporta un cambio in ognuna delle ripetizioni, il che oltre ad essere oneroso, può provocare degli errori.



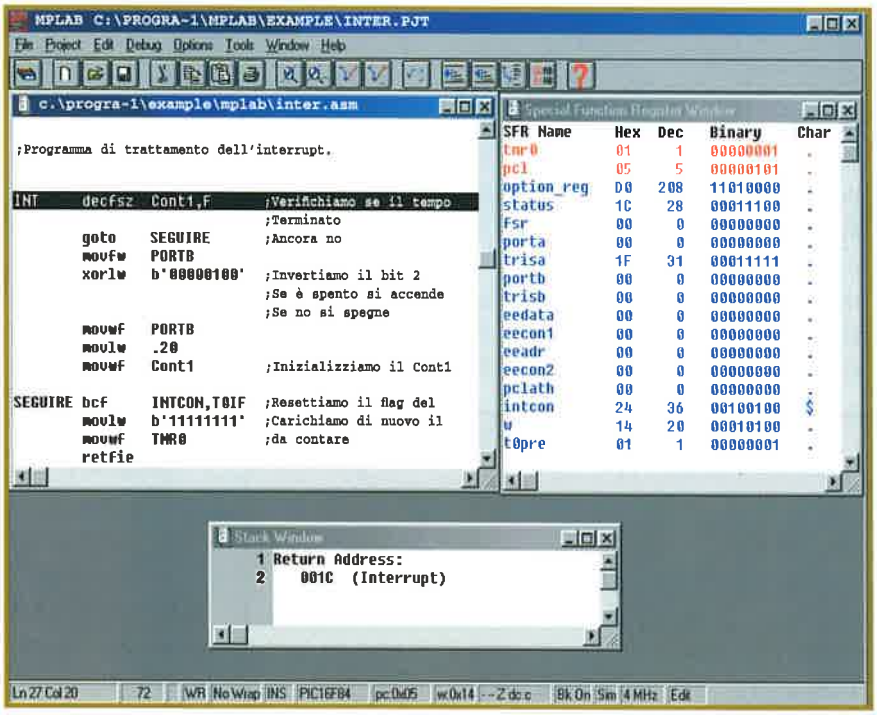
Programma che utilizza moduli nella stesura.

L'inconveniente è che sia la chiamata sia il ritorno dalla subroutine "sciupano" cicli di istruzione, in altre parole ogni chiamata ad una subroutine comporta una perdita di tempo,

Scheda delle istruzioni: l'istruzione RETFIE

È l'ultima delle istruzioni di ritorno, però questa volta da interrupt. Anche se il suo utilizzo appare simile a quello dell'istruzione return, in questo caso quando il processore ritorna al flusso principale del programma, recuperando l'indirizzo memorizzato nello Stack, pone il bit GIE del registro INTCON a valore 1.

Questo bit che abilita il microcontroller ad accettare gli interrupt, è posto a zero automaticamente entrando nella routine di servizio all'interrupt, per fare in modo che durante la sua esecuzione non ne vengano accettati altri. Mettendo di nuovo il bit GIE a 1 si torna a permettere gli interrupt. Se si esce da una routine di servizio all'interrupt con return, invece che con retfie, questa possibilità viene annullata.



Programma esempio per l'uso dell'istruzione retfie.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
r e t f i e		

Operazione: ritorno dalla subroutine di servizio dell'interrupt.
Cicli: 2
Codice OP: 00 0000 0000 1001
Flag: nessuno

Caratteristiche dell'istruzione retfie.

ESEMPI CON L'ISTRUZIONE RETFIE

Nella figura abbiamo tre finestre aperte: quella del codice del programma, quella dei registri specifici e quella dello Stack. È stato eseguito il programma passo a passo, e l'immagine è stata presa nel momento in cui si entra nell'interrupt. Nello Stack è rimasto l'indirizzo di ritorno, per quanto riguarda i registri specifici, possiamo notare che nel registro INTCON, e più esattamente il bit 7 o bit GIE, è appena stato messo a zero, per non permettere altri interrupt. Come ultima istruzione nelle subroutine dedicate agli interrupt, si trova la retfie.

Scheda delle istruzioni: l'istruzione RETLW

L'uso di subroutine è molto comune in qualsiasi programma del PIC. Dei due tipi di subroutine che possiamo distinguere, a seconda delle istruzioni di chiamata e ritorno, abbiamo già visto quella che corrisponde alle istruzioni call e return. L'istruzione con la quale lavoreremo in questa sezione, la retlw, è anche un'istruzione di ritorno, che ha similitudini e differenze rispetto all'istruzione return, pertanto si usa in differenti tipi di subroutine, a cui si accede sempre con l'istruzione call. Quando si torna da una subroutine con l'istruzione return, il valore che si trova nella prima posizione dello Stack viene caricato nel PC, in modo che questi torni a puntare l'indirizzo da cui era partito. Se al posto dell'istruzione return si utilizza l'istruzione retlw, oltre a portare di nuovo il valore dello Stack nel PC, in W rimane memorizzato il valore che appare come parametro dell'istruzione.

TABELLA	Dato1	Dato2	Dato3	Dato4	...	DatoN
	0	1	2	2	...	n-1

Array di nome "Tabella" con indice da 0 a N.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
r e t l w	k	

Operazione: ritorno dalla subroutine lasciando un valore letterale in **W**.
Cicli: 2
Codice OP: 11 01xx kkkk kkkk
Flag: nessuno

Caratteristiche dell'istruzione retlw.

ESEMPIO CON L'ISTRUZIONE RETLW

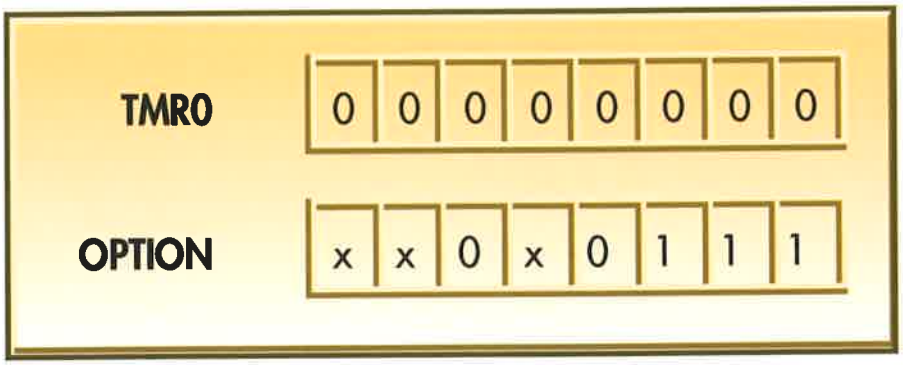
Questa istruzione si utilizza per formare le tabelle, negli altri linguaggi di programmazione si chiamano array, con indici consecutivi che iniziano la loro numerazione da 0, così come mostrato nell'immagine in alto. Per accedere ad un dato dell'array dobbiamo conoscere l'indice: il registro di lavoro W viene caricato con il valore dell'indice, e poi si esegue una chiamata alla tabella come se fosse una normale subroutine. Tornando dalla subroutine, in W rimarrà caricato il dato che corrisponde all'indice fornito. Per costruire una tabella dobbiamo porre il nome della medesima, una prima istruzione addwf con i parametri PCL e 0 e tante istruzioni retlw, quanti indici ha la tabella, ognuna con il parametro associato all'indice. La prima istruzione, che è obbligatoria, fa sì che al PC (la sua parte bassa PCL), venga sommato il valore che in quel momento si trova scritto in W, così da provocare un salto all'indice adeguato. Di tutte le istruzioni retlw, ad ogni chiamata alla tabella se ne eseguirà solo una, quella corrispondente all'indice utilizzato. Prendiamo l'esempio della figura: se prima di eseguire la chiamata alla tabella si carica il registro W con valore 1, al PC, che a causa della chiamata alla subroutine stava puntando all'istruzione addwf, verrà sommato 1 sulla sua parte bassa; questo più l'autoincremento normale che caratterizza questa risorsa del PIC, portano il PC a puntare all'istruzione retlw b'00001111' che è quella che si stava cercando.

```
            call    tabella
            ...
tabella     addwf   PCL,0
            retlw   'A'
            retlw   b'00001111'
            retlw   56
```

Frammento di programma nel quale si mostra l'utilizzo delle tabelle.

```
                LIST    P=16F84         ;Definizione del processore

TMR0            EQU     01              ;Definizione delle variabili
OPT             EQU     81
STATUS          EQU     03
PORTAB          EQU     06
INTCON          EQU     0B
DATO1           EQU     0C
DATO2           EQU     0D
DATO3           EQU     0E
CONT            EQU     0F

                ORG     0               ;Posizionamento della prima istruzione
                                        ;all'indirizzo 0 della memoria di codice

                bsf     STATUS,5        ;Passaggio al banco 1
                movlw   00              ;Configurazione della porta B come
                movwf   PORTAB          ;uscita
                movlw   b'00000111'     ;Configurazione di OPTION per il TMR0
                movwf   OPT
                bcf     STATUS,5        ;Passaggio al banco 0

                movf    DATO1,0         ;Visualizzazione dei dati prima del
                movwf   PORTAB          ;movimento dei dati
                call    RITARDO         ;Chiamata alla routine di ritardo
                movf    DATO2,0
                movwf   PORTAB
                call    RITARDO         ;Chiamata alla routine di ritardo
                movf    DATO3,0
                movwf   PORTAB
                call    RITARDO         ;Chiamata alla routine di ritardo

                movlw   03              ;Movimento di un valore a W
                movwf   DATO1           ;Movimento da W a DATO1
                movf    DATO2,0         ;Movimento del DATO2 a W
                movwf   DATO3           ;Movimento di W a DATO3

                movf    DATO1,0         ;Visualizzazione dei dati dopo il
                movwf   PORTAB          ;movimento dei dati
                call    RITARDO         ;Chiamata alla routine di ritardo
                movf    DATO2,0
                movwf   PORTAB
                call    RITARDO         ;Chiamata alla routine di ritardo
                movf    DATO3,0
                movwf   PORTAB
                call    RITARDO         ;Chiamata alla routine di ritardo
                goto    FINE            ;Terminare programma

RITARDO         movlw   d'15'           ;Ciclo esterno che si ripeterà 15 volte
                movwf   CONT
CICLO1          movlw   00              ;Inizializzazione del ciclo interno
                movwf   TMR0
CICLO2          btfss   INTCON,2        ;Ha finito di contare?
                goto    CICLO2          ;No. Continua nel ciclo interno
                bcf     INTCON,2        ;Si. Azzera il flag
                decfsz  CONT,1          ;Decrementa il contatore del ciclo esterno
                goto    CICLO1          ;Dato che non è zero torna al ciclo interno
                return                  ;Il contatore esterno è arrivato a zero, esci dalla routine

FINE            END                     ;Fine del programma
```

Programma con la routine di temporizzazione integrata.



Configurazione del registro OPTION e del TMR0 per contare 256x256.

inserire il valore indicato nella figura in alto. Le x sono valori che in questo specifico caso possono essere 1 oppure 0 indifferentemente. I tre bit meno significativi (0-2) messi a 1 indicano un divisore di frequenza di 256, valore adatto a questa temporizzazione. Il bit 3 assegna questo divisore al TMR0. Il bit 5 a zero fa sì che il TMR0, dei due modi in cui può contare, lo faccia utilizzando gli impulsi del clock interno.

ROUTINE DEL TMR0

La routine di temporizzazione funzionerà come mostrato nella figura. Inizia con il nome della routine, che abbiamo chiamato "ritardo". Come abbiamo spiegato in precedenza, il tempo desiderato si otterrà con un ciclo all'interno di un altro, con quello interno che realizza una temporizzazione di 256x256 ms, grazie al TMR0, e un altro esterno che fa ripetere 15 volte quello interno. L'introduzione del valore 15 fra virgolette semplici, e con una 'd' davanti, fa sì che il valore sia considerato come decimale. All'inizio di ogni ciclo dobbiamo inizializzare i contatori associati; verificando il valore del bit 2 del registro INTCON, si sa se il TMR0 ha terminato il conteggio, nel qual caso questo bit passerà automaticamente a 1 e noi lo dovremo rimettere a zero. Al termine di entrambi i cicli si esce dalla routine.

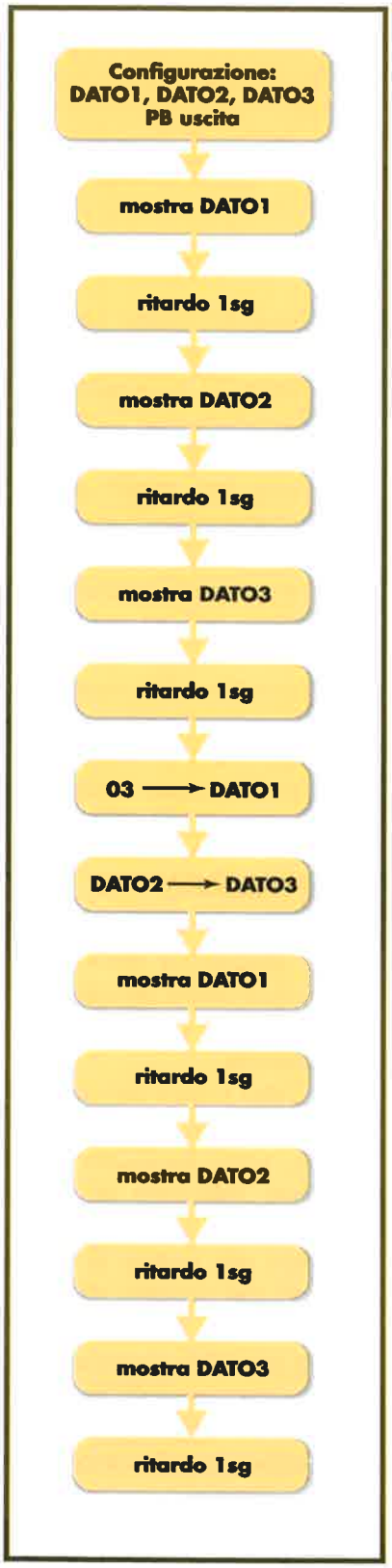
PROGRAMMA TOTALE

Una volta visto perché si fa così, come configurare il TMR0 e il registro OPTION, e come rimane la routine di temporizzazione, andiamo ad integrare tutto questo nel programma che vogliamo modificare.

Le linee in neretto sono quelle nuove aggiunte nel programma; come si può vedere tutti i registri che utilizzeremo in seguito devono essere definiti. Nella configurazione delle variabili abbiamo assegnato nuovi valori oltre a quelli che già esistevano. Il registro OPTION lo abbiamo definito come OPT perché la parola OPTION è una parola riservata dell'Assembler, e utilizzata in questo modo avrebbe generato un errore. Quelli che nell'organigramma erano rettangoli, di "ritardo di 1 s", ora sono stati convertiti in chiamate ad una routine di ritardo. Questa routine, di cui abbiamo già parlato, è stata posta al termine del programma, anche se poteva essere collocata da qualsiasi altra parte, tenendo sempre conto dell'ordine con cui si esegue il programma; notate che dopo l'ultima chiamata alla routine "ritardo" abbiamo un salto alla fine del programma, altrimenti tornerebbe ad eseguirsi questa routine. Il programma, come sempre, termina con la direttiva END.

```
RITARDO   movlw   d'15'     ;Ciclo esterno che si ripeterà 15 volte
          movwf   CONT
CICLO1    movlw   00        ;Inizializzazione del ciclo interno
          movwf   TMR0
CICLO2    btfss   INTCON,2  ;Ha finito di contare?
          goto    CICLO2    ;No. Continua nel ciclo interno
          bcf     INTCON,2  ;Si. Azzera il flag
          decfsz  CONT,1    ;Decrementa il contatore del ciclo esterno
          goto    CICLO1    ;Dato che non è zero torna al ciclo interno
          return            ;Il contatore esterno è arrivato a zero, esci dalla routine
```

Routine di temporizzazione del TMR0.



Organigramma dell'enunciato con l'inserimento dei ritardi.

dente all'interno del programma principale, ed è chiamato routine, o subroutine, e si utilizza quando un pezzo di programma deve essere ripetuto varie volte. In questo caso questo pezzo di programma si ripete 6 volte, sempre nello stesso modo. L'ultimo ritardo non sarebbe necessario, dato che come abbiamo potuto verificare nella simulazione, il dato finale permane, e lo si può osservare senza problemi.

IL TMR0

I metodi per ottenere un ritardo, "una perdita di tempo", sono molto vari. Il primo che ci viene in mente è porre una fila di istruzioni nop una dietro all'altra; però se vogliamo ottenere una temporizzazione di un secondo, e ogni istruzione impiega 1 µs ad essere eseguita, dovremmo utilizzare più o meno un milione di istruzioni di questo tipo, il che oltre che assurdo è impossibile, dato che il PIC16F84 ha solo una capacità di 1024 istruzioni. La seconda opzione è eseguire cicli all'interno di altri cicli, in modo che una ripetizione dietro l'altra fornisca il valore desiderato. Considerata la struttura di controllo dedicata alla programmazione del PIC, avremo bisogno di un contatore a cui far decrementare un valore, e dato che i registri sono a 8 bit, il valore massimo che si può ottenere è 255, quindi il valore totale da contare si divide in gruppi da 256 (da 0 a 255). I cicli proposti realizzano 256x256x15 = 983.040 istruzioni nop, che si avvicina al valore cercato.

Questo modo di lavorare è valido, anche se non è il migliore. Quando ci serve contare il tempo, conviene fare ricorso ad una risorsa del PIC che è il temporizzatore Timer0 (TMR0). Questo temporizzatore, una volta configurato, è capace di contare un valore determinato (256 x 256 come valore massimo), in modo indipendente da quello che si sta facendo con la CPU del PIC. Alla fine ci avviserà. Così, in questo caso dobbiamo fare un unico ciclo in cui il TMR0 conterà 15 volte il valore

```
FOR   256   DO
      FOR   256   DO
            FOR   15    DO
                  Nop
            ENDFOR
      ENDFOR
ENDFOR
```

Con un ciclo all'interno dell'altro si può arrivare alla temporizzazione desiderata.

massimo. Il modo di configurare il TMR0 e il suo registro associato per questa applicazione, il registro OPTION, è illustrato nella figura.

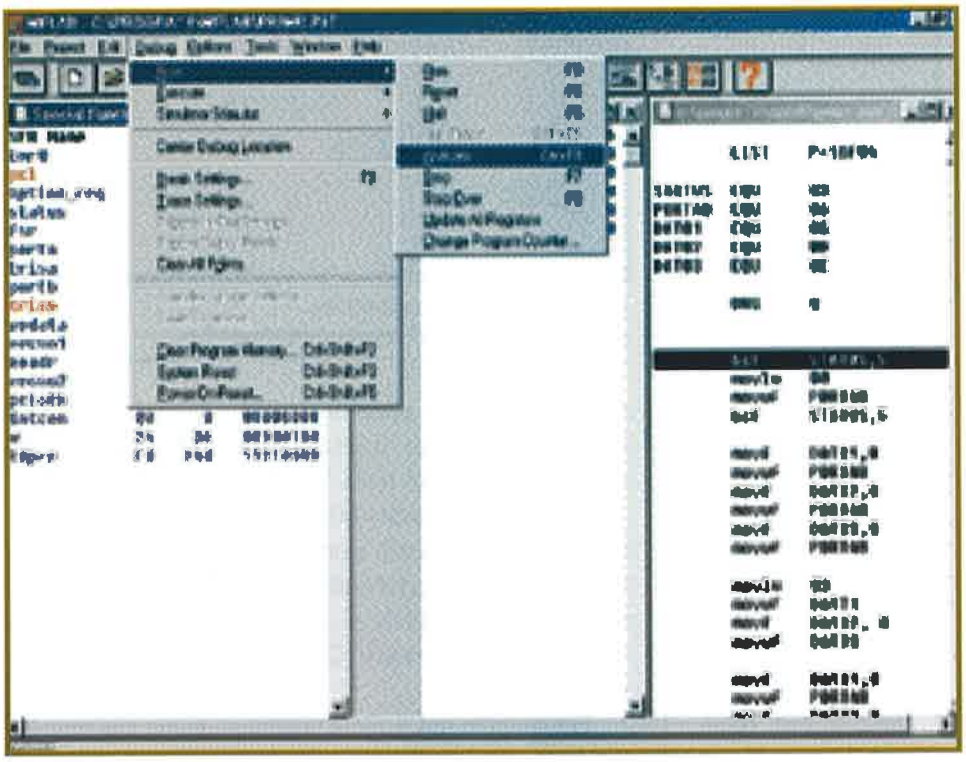
Metteremo il valore 00h nel TMR0, il quale si incrementerà ad ogni ciclo sino ad arrivare a FFh, da dove tornerà a 00h, producendo quello che viene chiamato overflow, con il quale si attiverà un flag per indicare che ha finito di contare. Questo flag è il bit 2 del registro INTCON.

Per quanto riguarda la configurazione del registro OPTION, questa deve essere fatta all'inizio del programma. Nel caso di questo esercizio, dobbiamo

Applicazione pratica: contando il tempo

Quando abbiamo modificato il primo programma per visualizzare i registri prima e dopo il movimento dei dati, proponemmo di fare una simulazione del tipo istruzione per istruzione, per vedere come veniva eseguita; avete provato a fare una simulazione del programma in modo completo, per vedere come cambiano i registri nel momento in cui avviene? Questo è proprio il momento. Apriamo il progetto e ci appariranno le finestre così come le abbiamo lasciate. Facciamo un reset del sistema e introduciamo il valore FA nella posizione 0D della memoria dei dati. Quando saremo pronti per fare questo tipo di simulazione, dovremo selezionare l'opzione Animate all'interno del menù Debug>Run. Fatelo ora.

Così come succedeva eseguendo il primo programma in questo modo, al termine delle nostre istruzioni il Program Counter (PC) continua ad incrementarsi, e dobbiamo fermarlo con Debug>Run>Halt. La velocità dipenderà dal computer su cui si sta eseguendo il programma, ma, quasi certamente, sarà troppo alta per poter vedere adeguatamente come variano i registri. La stessa cosa succederà quando il programma invece di essere simulato, sarà eseguito nel microcontroller con hardware reale: la visualizzazione dei dati che ci interessano sarà appena apprezzabile. L'unico dato che si potrà vedere chiaramente sarà l'ultimo che rimane nella Porta B come nella simulazione. Quindi sarà necessario introdurre alcuni tipi di ritardo, che ci permetteranno, quando saremo nei panni dell'utilizzatore, di interagire con il sistema.



Uno dei possibili modi di simulazione.

ORGANIGRAMMA

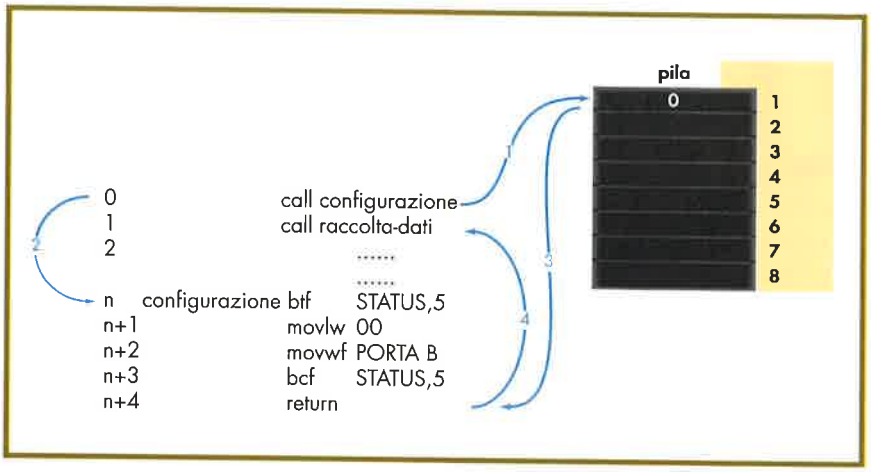
Qualsiasi cambio che faremo nel programma dovrà essere riportato anche all'inizio, cioè dovrà essere inserito nell'organigramma.

L'idea è che i dati che dobbiamo vedere mediante i diodi LED, siano visualizzati per il tempo sufficiente ad essere visti. Questo si ottiene introducendo una routine di ritardo dopo aver spostato ogni dato sulla Porta B. Per fare in modo che ogni dato sia visualizzato per circa 1 secondo sarà sufficiente per raggiungere il nostro obiettivo. Modifichiamo anche l'organigramma.

Quello che è stato introdotto nell'organigramma che già abbiamo, è una specie di piccolo programma indipen-

Scheda delle istruzioni: l'istruzione RETURN

È l'istruzione di complemento alla call. Mentre la call si utilizza per la chiamata a subroutine, l'istruzione return comanda il ritorno dalla subroutine, una volta terminata la sua esecuzione. Per il ritorno non è necessario nessun parametro, dato che l'indirizzo si trova già nella prima posizione dello Stack, memorizzato al momento dell'esecuzione dell'istruzione call. Per questo motivo bisogna fare attenzione a non utilizzare l'istruzione return senza aver preventivamente usato l'istruzione call, altrimenti non avremo nessun indirizzo di ritorno nello Stack, oppure finiremmo per utilizzare l'indirizzo di ritorno di un'altra istruzione.



Sequenza dei passi durante la chiamata a subroutine.

ESEMPI CON L'ISTRUZIONE RETURN

Nella figura possiamo osservare parte di un programma che utilizza delle subroutines, e come funziona il PIC internamente quando le esegue. A sinistra abbiamo una serie di chiamate a subroutine e lo sviluppo di una di esse. Possiamo supporre che la prima istruzione inizi nella posizione 0 della memoria di codice e le seguenti occupino posizioni successive a partire da questa. La scheda rappresenta la sequenza di esecuzione delle istruzioni del programma e quello che comportano, e i numeri sopra le frecce, l'ordine di questa sequenza. Quando si esegue la prima istruzione, si tratta di una call, si scrive nello Stack l'indirizzo dell'istruzione (la 0), per poi poter ritornare. Nel PC si carica l'indirizzo dell'etichetta che accompagna la call, in questo caso "configurazione", in modo che l'istruzione che si trova in questo indirizzo sarà la seguente da eseguire. Si segue in ordine la subroutine fino ad arrivare all'istruzione return, che farà sì che si torni a caricare nel PC l'indirizzo memorizzato in cima allo Stack, che permette di tornare ad eseguire l'istruzione successiva alla prima chiamata a subroutine.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
r e t u r n		

Operazione: ritorno da subroutine.
Cicli: 2
Codice OP: 00 0000 0000 1000
Flag: nessuno

Caratteristiche dell'istruzione return.

Scheda delle istruzioni: l'istruzione CALL

Quando abbiamo spiegato le strutture di controllo, abbiamo visto che un programma non può essere composto solamente da istruzioni disposte in forma sequenziale, dato che la normalità è introdurre strutture che rompano questa sequenza, realizzando un salto all'interno del programma per eseguire parti distinte, secondo la convenienza.

In questo caso rompiamo la sequenza con la quale si eseguono le istruzioni, non a causa del valore di una condizione, ma per rendere la struttura del programma più chiara. Quando abbiamo visto le parti di cui si componeva questa struttura, abbiamo già parlato di subroutines, anche se non le abbiamo trattate in profondità. Per realizzare questa subroutine si utilizza l'istruzione call. Questa istruzione fa sì che il puntatore salti ad un'altra parte del programma, etichettato col nome che accompagna l'istruzione come parametro. Questa chiamata, o salto è incondizionato, come l'istruzione goto, però non si comporta come questa. All'arrivo di un'istruzione call, prima di produrre il salto, si porta automaticamente allo Stack del PIC il valore del PC, in modo che, una volta conclusa la subroutine, si possa tornare alla parte di programma da cui è stata fatta la chiamata, e continuare con l'esecuzione normale. Poiché non si conosce il valore del PC sino a che non si esegue l'istruzione, non si è potuto calcolare in precedenza, per cui i cicli di istruzione sono due.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
call	k	

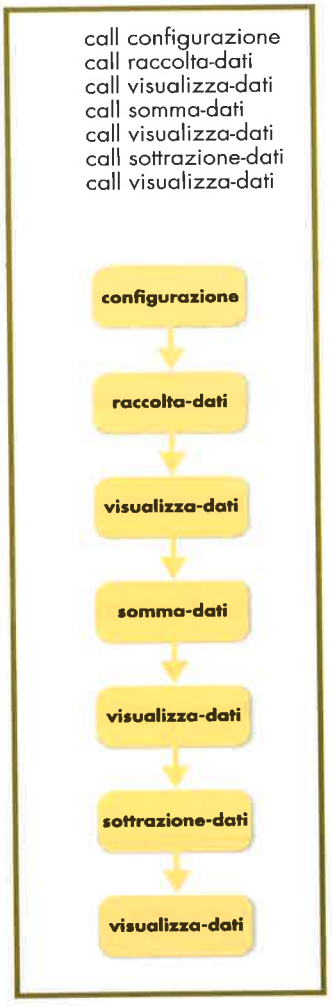
Operazione: chiamata ad una subroutine il cui nome appare come parametro dell'istruzione.
Cicli: 2
Codice OP: 10 0kkk kkkk kkkk
Flag: nessuno

Caratteristiche dell'istruzione call.

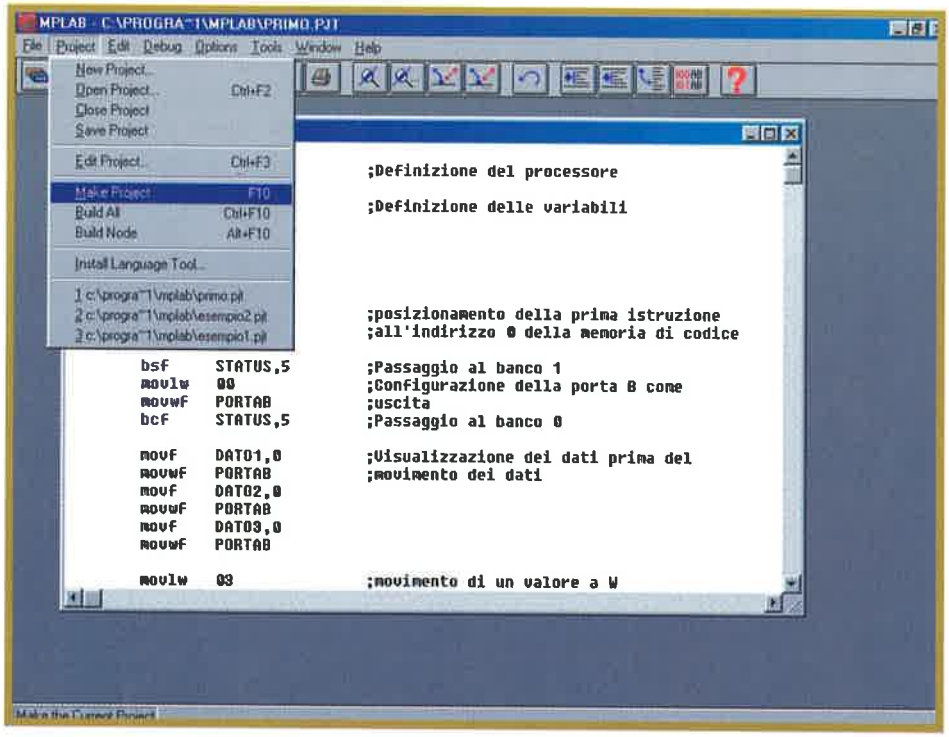
ESEMPI CON L'ISTRUZIONE CALL

La chiarezza in un programma è una qualità molto apprezzata, specialmente quando il linguaggio utilizzato è l'assembler, che necessita di molte istruzioni per realizzare operazioni semplici. La bravura consiste nel pensare ad una struttura del programma così come se si trattasse di una sequenza di moduli, senza entrare nei loro dettagli interni. Quanto più complesso sarà il modulo, e più frammenti ripetuti contiene, tanto più sarà necessario fare uso di subroutine.

La traduzione dell'organigramma, che rappresenta il flusso principale del programma, può essere immediata utilizzando le subroutines. Se si ritiene necessario si possono fare altri organigrammi indipendenti per ognuno dei moduli. Trasportando questo in codice assembler si converte in una sequenza di istruzioni call che si eseguono una dopo l'altra, e formano la parte principale del programma. Al termine di questa parte principale subentra lo sviluppo di ognuna delle subroutines.



Sequenza dei passi durante la chiamata a subroutine.



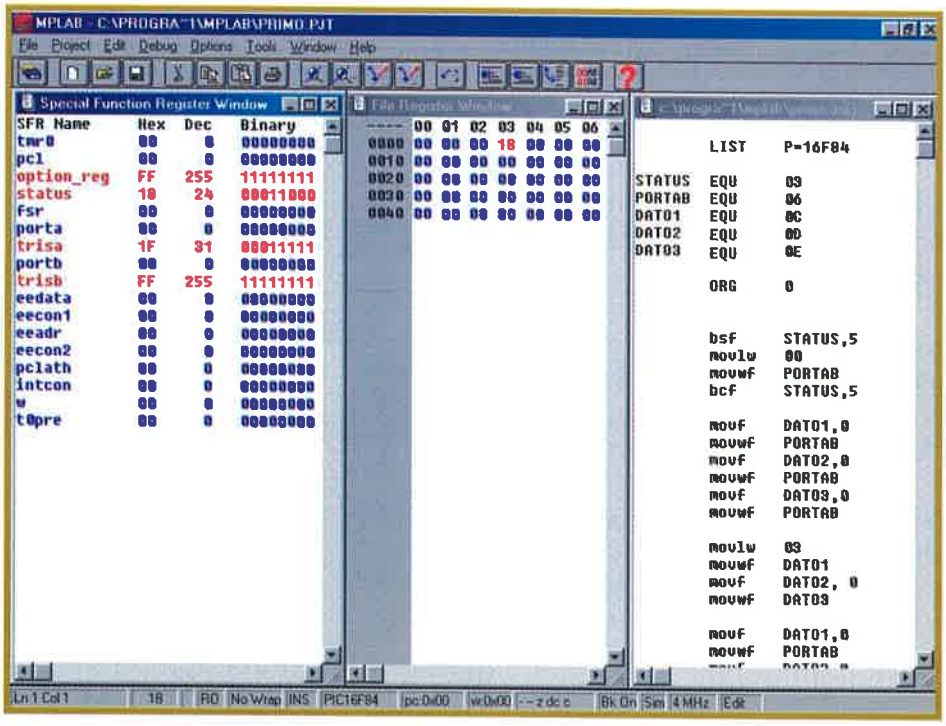
Compilazione del programma.

aprire l'ultimo progetto che è stato in uso, in modo che, se coincide con il progetto "primo" basterà rispondere affermativamente. Si apriranno in seguito i files associati a questo progetto e nel nostro caso il file con lo stesso nome.

Nel caso che non sia stato l'ultimo progetto utilizzato, possiamo rispondere no, e aprire il progetto tramite il menù principale di MPLAB. Allo stesso modo si apriranno i files associati. Nell'uno o nell'altro caso si può ampliare direttamente il programma che abbiamo già visto per adattarlo ai nuovi requisiti.
Una volta salvato il file e sommato al progetto, si passerà alla sua compilazione.

Dopo aver verificato che non ci siano errori di sintassi procederemo a realizzare la simulazione per vedere se il programma esegue quello che vogliamo. Nella finestra Option sceglieremo il modo simulazione e apriremo le finestre dei registri specifici (Special Function Register) e dei registri generali (File Register), oltre a quelle del programma che abbiamo già aperto. Per vedere tutte le finestre nello stesso tempo, le ordineremo verticalmente (Tile Vertical). Potremo in seguito regolare la loro dimensione a piacere.

Dopo aver fatto un Reset del sistema introdurremo un valore nel registro etichettato come DATO2, ad esempio il valore esadecimale FA, e tra le possibili azioni di simulazione sceglieremo quella di evidenziare. Cliccando successive volte F7 resteranno evidenziate le istruzioni che entreranno in esecuzione e i valori che prenderanno i registri. Alla fine verificheremo che il valore contenuto in essi sia corretto.



Finestre necessarie per la simulazione del programma.

Normalmente, prima di cominciare il programma si devono configurare i registri prima definiti. Il nostro programma non fa eccezione. I registri che dobbiamo configurare sono quelli specifici, e questa configurazione determinerà il comportamento successivo del PIC. Per utilizzarli correttamente dobbiamo conoscere bene le caratteristiche di ognuno di questi registri.
La configurazione che dobbiamo fare è quella di assegnare le otto linee della porta B come linee di uscita per mostrare i dati che vengono richiesti. Per questo prima dobbiamo passare al banco 1. L'assegnazione come linea di uscita si farà ponendo a 0 il registro TRISB o, come si è spiegato precedentemente, in PORTAB sapendo che siamo nel banco 1. Dopo questo si torna al banco 0.

La visualizzazione dei dati è semplice tanto come muovere il valore che contiene ognuno dei registri alla porta B. Dato che non abbiamo nessuna istruzione che permetta il movimento dei dati tra due registri direttamente, si utilizza come punto di appoggio il registro di lavoro W. Si vede così che la porta B (e lo stesso per la porta A), anche se è speciale perché permette l'ingresso e l'uscita dei dati da e per l'esterno del PIC, si gestisce in eguale modo di tutti gli altri registri.

Il blocco seguente di istruzioni è già stato spiegato nel primo programma, ed è quello che realizza il movimento dei dati.

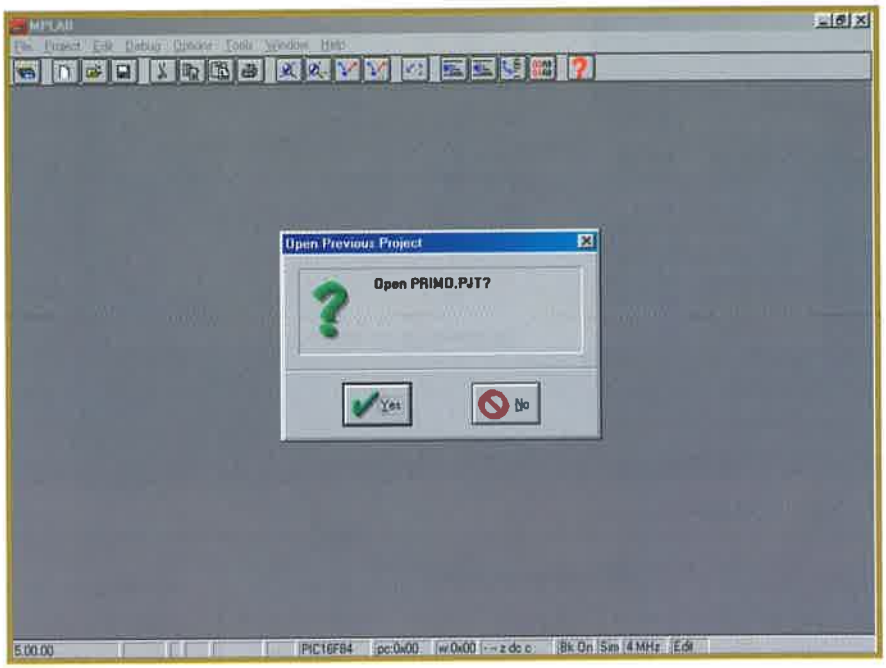
Per ultimo ci viene richiesto di visualizzare i dati dei registri con dei nuovi valori. La sequenza di istruzioni è una ripetizione di quelle già viste per la prima visualizzazione.

La fine del programma si ha con la direttiva AND. Ricordiamoci che abbiamo seguito le norme per scrivere un programma così come sono state spiegate per realizzare il primo. Il programma, inoltre, contiene tutte le parti che ogni programma di compilazione richiede, anche se non sono tutte utilizzate.

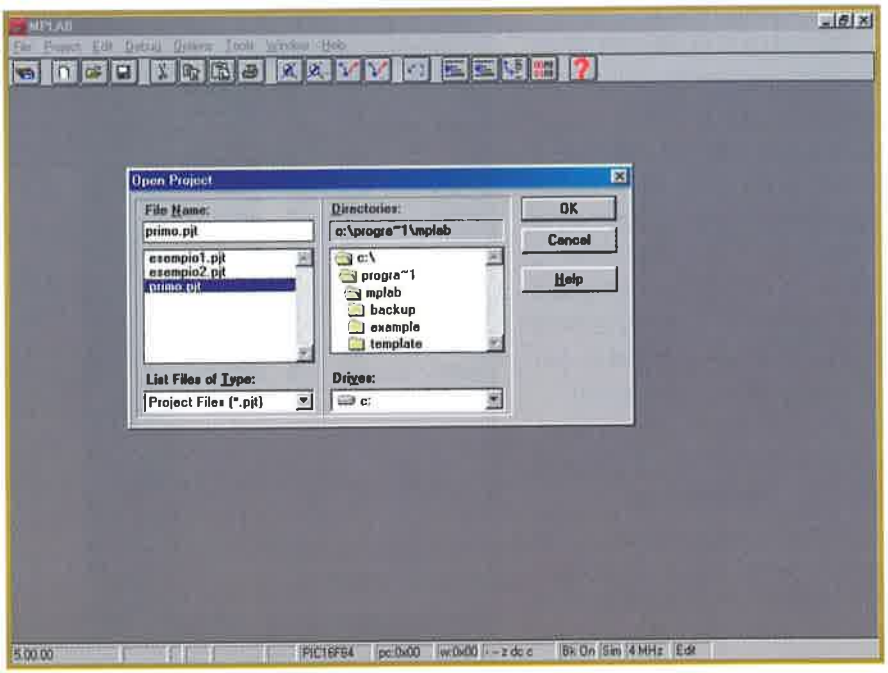
COMPILAZIONE DEL PROGRAMMA

Il programma che abbiamo appena visto dovremo editarlo in MPLAB per poterlo poi in seguito compilare. Si può creare un progetto e un file nuovo o aprire quello che avevamo già fatto con il nome "primo".

Entrando in MPLAB ci verrà chiesto se dobbiamo



Apertura di un progetto che abbiamo già e che è stato l'ultimo utilizzato.



Apertura di un progetto che abbiamo già, però non è stato l'ultimo ad essere utilizzato.

SCRITTURA DEL PROGRAMMA SORGENTE

Fatto l'organigramma, dobbiamo tradurlo in istruzioni per il PIC.

Dopo la direttiva LIST si definiscono i registri che vogliamo usare. Nell'organigramma apparivano già i registri dei dati e la porta B. Nel realizzare il programma vediamo che abbiamo bisogno anche del registro di STATO. Il bit 5 di questo registro ci permette di muoverci dal banco 0 al banco 1. Nel banco 0 avremo la porta B, che si trova all'indirizzo 06 di questo banco, e nel banco 1 il registro di configurazione di questa porta B (TRISB) che occupa la stessa posizione. Questi registri si potrebbero chiamare rispettivamente PORTAB e TRISB, oppure si possono chiamare entrambi nello stesso modo (PORTAB), così come abbiamo fatto noi, tenendo conto però che significano cose diverse, a seconda del banco in cui le stiamo utilizzando. DATO1, DATO2 e DATO3 sono registri di utilizzo generale che possono essere impiegati sia nel banco 0 che nel banco 1.

Dopo la definizione delle variabili, si deve posizionare una prima istruzione del programma alla posizione 0 della memoria di istruzioni. Questo si fa attraverso la direttiva ORG con il parametro 0.

```
                LIST    P=16F84         ;Definizione del processore

STATUS          EQU     03              ;Definizione delle variabili
PORTAB          EQU     06
DATO1           EQU     0C
DATO2           EQU     0D
DATO3           EQU     0E

                ORG     0               ;posizionamento della prima istruzione
                                        ;all'indirizzo 0 della memoria di codice

                bsf     STATUS,5        ;Passaggio al banco 1
                movlw   00              ;Configurazione della porta B come
                movwf   PORTAB          ;uscita
                bcf     STATUS,5        ;Passaggio al banco 0

                movf    DATO1,0         ;Visualizzazione dei dati prima del
                movwf   PORTAB          ;movimento dei dati
                movf    DATO2,0
                movwf   PORTAB
                movf    DATO3,0
                movwf   PORTAB

                movlw   03              ;Movimento di un valore a W
                movwf   DATO1           ;Movimento di W a DATO1
                movf    DATO2, 0        ;Movimento di DATO2 a W
                movwf   DATO3           ;Movimento di W a DATO3

                movf    DATO1,0         ;Visualizzazione dei dati dopo il
                movwf   PORTAB          ;movimento dei dati
                movf    DATO2,0
                movwf   PORTAB
                movf    DATO3,0
                movwf   PORTAB

                END                     ;Fine del programma
```

Programma in assembler dell'enunciato proposto.

Applicazione pratica: visualizzazione dei dati del primo programma

A questo punto abbiamo imparato a realizzare tutte le fasi che comporta un programma sino al momento della simulazione. Complicheremo ora un po' il primo programma proposto, e faremo tutte le fasi una dopo l'altra.

Modificheremo l'enunciato dell'esercizio nel seguente modo:

> **Abbiamo tre posizioni di memoria che chiameremo DATO1, DATO2 e DATO3. Nel DATO1 si vuole scrivere il valore esadecimale 03 e nel DATO3 il valore che si trovava nel DATO2. Si mostreranno tramite i diodi LED i valori dei registri prima e dopo le operazioni di movimento dei dati.**

Enunciato ampliato del primo programma proposto.

L'applicazione dell'esercizio suppone di vedere i dati, di modo che l'utilizzatore possa osservare cosa sta succedendo all'interno del microcontroller.

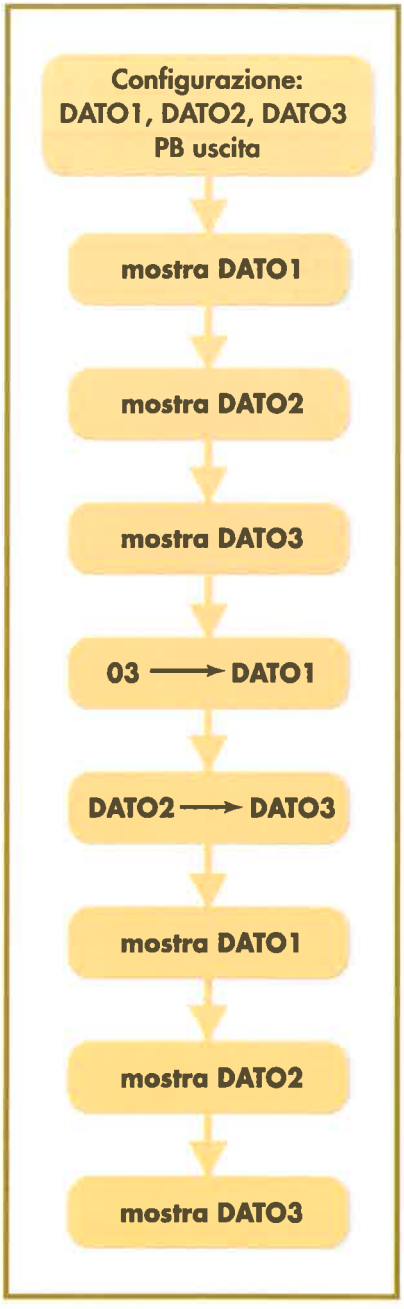
ORGANIGRAMMA

Quando parliamo dei passi per la realizzazione di un progetto, abbiamo già visto che la prima cosa da fare è plasmare l'enunciato dell'esercizio in un organigramma. Quanto più preciso è l'enunciato, tanto più facile sarà questa trasformazione, e meno ambiguità esisteranno al momento di interpretarlo. Se si vuole fare un organigramma dell'enunciato precedente dovrebbe risultare come quello della figura a fianco.

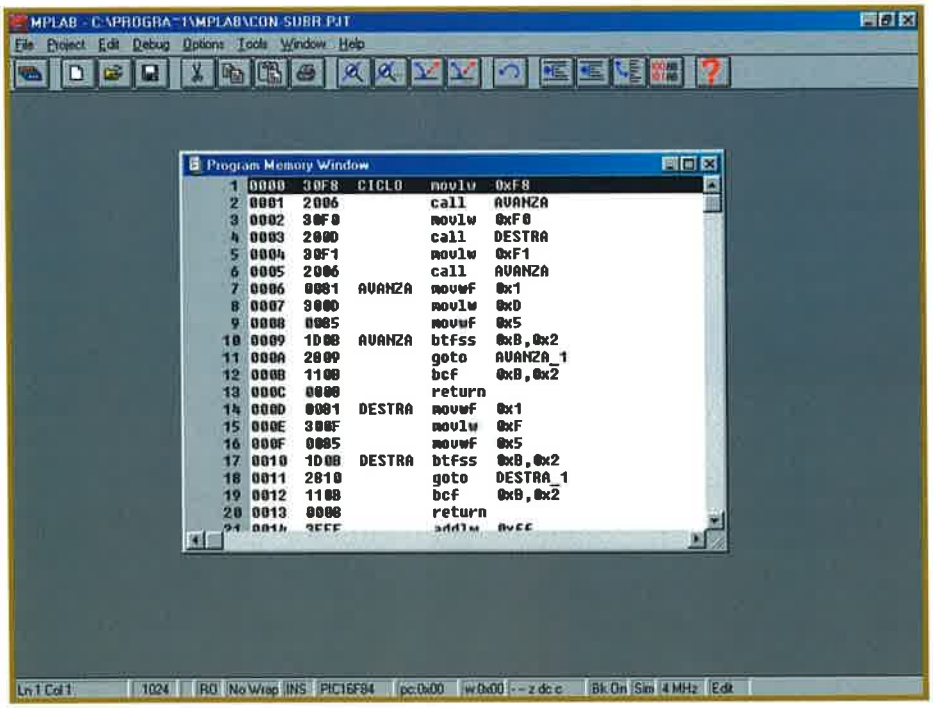
Nel fare un organigramma, il primo riquadro deve corrispondere alla definizione e alla configurazione dei registri. Anche se non è obbligatorio decidere in precedenza quali risorse si utilizzeranno, facilita il lavoro successivo di programmazione. Dal nostro enunciato possiamo dedurre che si utilizzeranno tre registri e una porta, che verrà configurata come uscita, per mostrare i dati. Dato che i registri hanno otto bit e delle due porte esistenti nel PIC16F84 solo la porta B è di otto linee, si è scelta questa.

Il programma consisterà in una serie di istruzioni disposte in forma sequenziale, e questo si può vedere nell'organigramma per l'assenza dei rombi che fanno prendere una decisione in base a un dato.

All'interno dell'organigramma, il modo di rappresentare la visualizzazione di un dato si può ridurre scrivendo la parola "DATO", senza preoccuparci ancora di quali istruzioni si utilizzeranno per questo.



Organigramma dell'enunciato proposto.



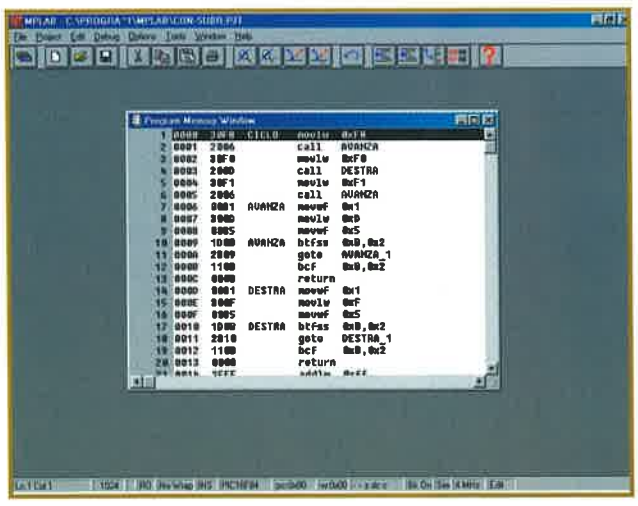
Generazione del codice nel programma scritto con subroutine.

ne, il numero di istruzioni generate, e che più tardi saranno scritte nella memoria del microcontroller, risulta lo stesso di quelle scritte nel programma al momento dell'editazione. Quando si utilizzano macro, invece, ogni apparizione di un nome di macro è sostituito dal compilatore, da tutte le istruzioni che formano la macro; questa operazione è chiamata espansione di una macro. In questo modo la quantità di istruzioni che saranno scritte nella memoria di programma sarà maggiore che nel caso di utilizzo di subroutine, però il tempo di esecuzione sarà minore, perché non ci saranno chiamate a subroutine durante l'esecuzione del programma. Quando il codice da includere in un modulo separato dal programma principale è molto grande, di solito si usano le subroutine, quando detto codice è molto piccolo, è meglio utilizzare le macro.

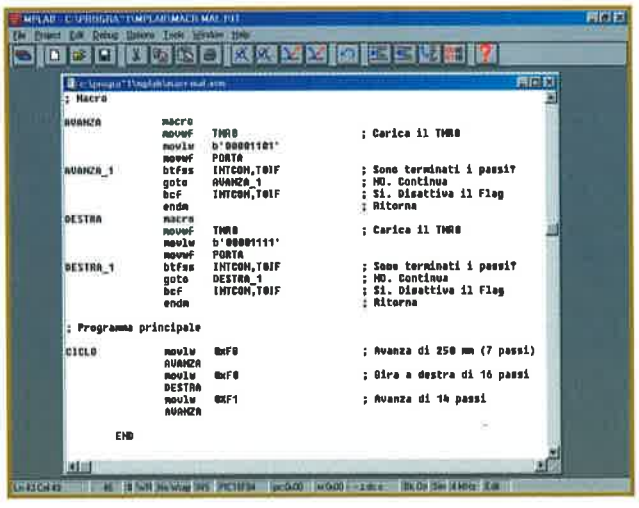
PROBLEMI NELL'USO DELLE MACRO

Non si può parlare di problemi propriamente detti, ma di precauzioni e di norme di buon uso, che bisogna conoscere e applicare, perché questi non si producano. A causa della differenza che esiste al momento di assemblare, fra macro e subroutine, l'inserimento dei cicli non si può realizzare nello stesso modo. Nel secondo programma di esempio, sono state sostituite le subroutine con le macro, definendole in modo errato.

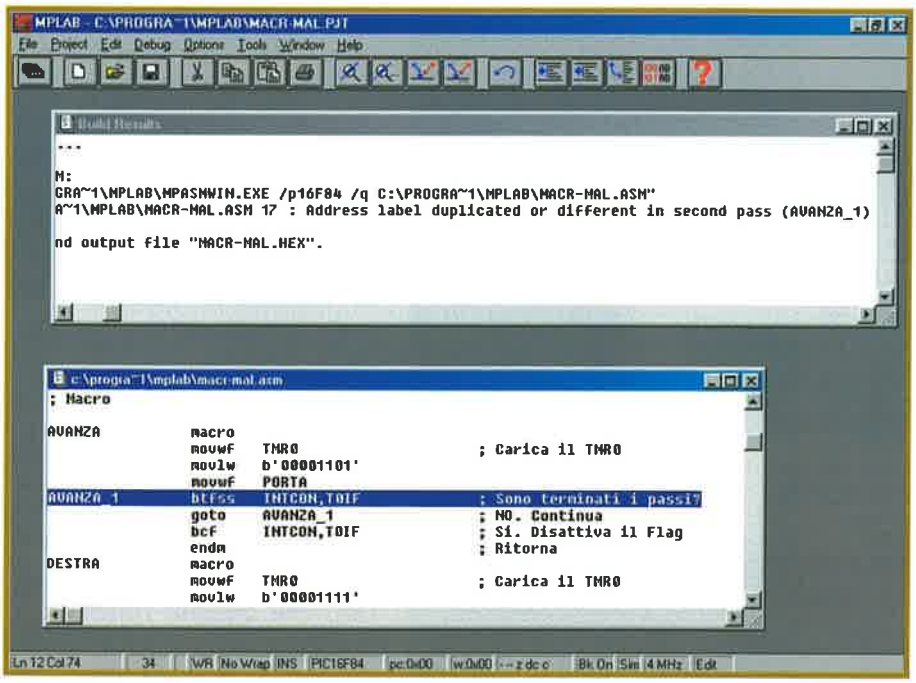
Dato che tutte le istruzioni delle macro si ripetono tante volte quante sono le chiamate, si verificherà anche la ripetizione delle etichette che formano parte di queste macro. Questo genera un problema, quando un'istruzione, ad esempio una goto, fa riferimento ad un'etichetta e questa etichetta appare duplicata nel



Generazione del codice nel programma scritto con macro.



Secondo programma esempio con le macro definite in modo errato.



Errore che genera MPASM cercando di assemblare il secondo programma.

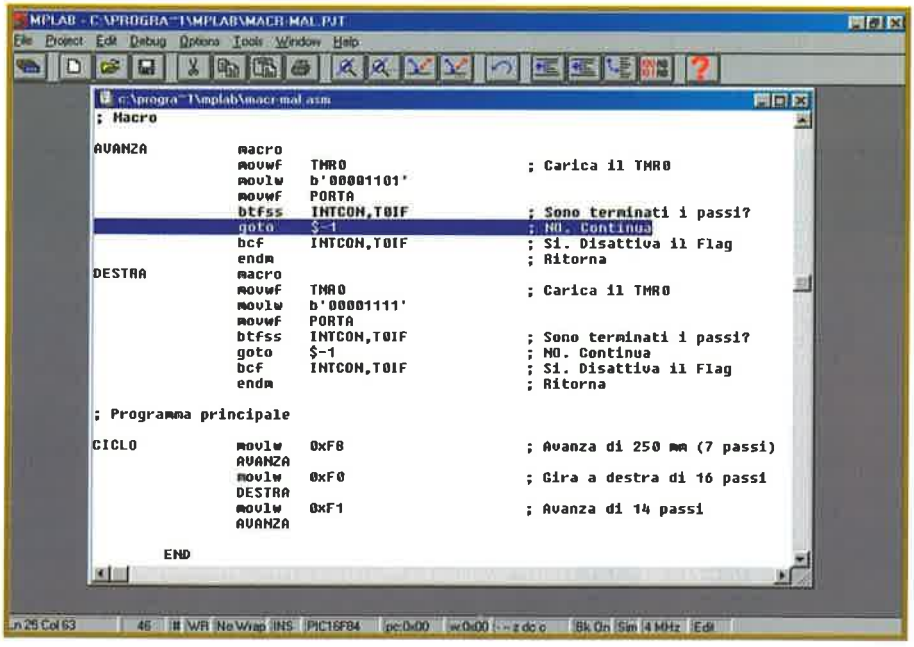
codice compilato. Una soluzione a questo inconveniente, la troviamo utilizzando gli indirizzi relativi. Invece di saltare verso l'indirizzo esatto di un'etichetta, si salta verso una posizione che sarà traslata in anticipo o in ritardo del valore che ha in quel momento il PC. Visto che ogni chiamata a macro verrà sostituita da tutte le istruzioni che la compongono, ogni istruzione verrà collocata in indirizzi diversi della memoria di programma, per cui il PC al momento di eseguirle avrà un valore diverso, e gli indirizzi che ne risulteranno per incremento o decremento anche. Se i salti sono realizzati "verso l'esterno" della macro, si possono utilizzare etichette, dato che queste non saranno considerate ripetute.

RECURSIVITÀ

La recursività è definita come la capacità di un subprogramma

Si no	X = 0
Allora	FACTOTIAL = 1
Se no	FACTORIAL = X * FACTORIAL (X - 1)

Esempio di programma scritto in forma recursiva



Forma corretta di fare i salti nelle macro.

di auto-chiamarsi. Al momento di dichiararla, bisogna identificare bene il caso base per cui il procedimento terminerà.

Non tutti i problemi possono essere definiti in base alla recursività, e generalmente è una delle cose più difficili da capire e applicare nella programmazione. In ogni caso con il PIC potremo utilizzarla poco, perché con la recursività si generano indirizzi di ritorno ogni volta che la subroutine si "auto-chiama", e nel PIC16F84 abbiamo solo 8 indirizzi di ritorno, tanti quanti ne può contenere lo Stack, numero che è molto basso per la maggior parte dei programmi recursivi.

Scheda delle istruzioni: L'istruzione SUBLW

L'istruzione "sublw" potrebbe essere descritta come il contrario dell'istruzione "addlw", dato che realizza la sottrazione di un valore letterale meno il valore del registro W. Il risultato rimane in quest'ultimo. I flag che vengono attivati sono gli stessi che nel caso della somma, però, dato che l'operazione si realizza mediante il metodo del complemento a due, il flag di carry, invece di attivarsi quando c'è riporto, lo fa quando non c'è.

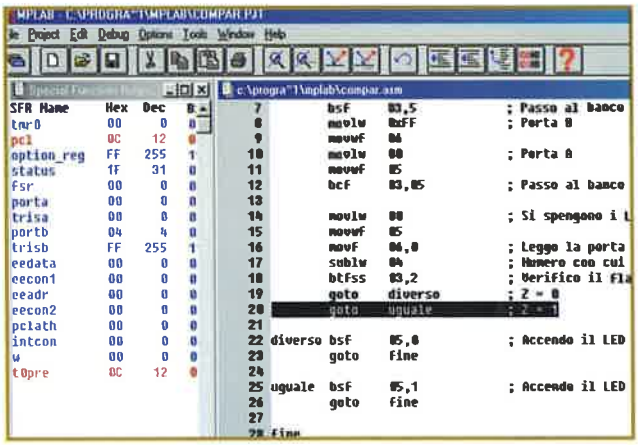
MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
s u b l w	k	

Operazione: Sottrazione del valore letterale **k** al valore del registro **W** (k-W)
Cicli: 1
Codice OP: 11 110x kkkk kkkk
Flag: C, DC, Z

Caratteristiche dell'istruzione sublw.

Valore di W prima dell'istruzione	00000011
Valore letterale da cui sottrarre	00000100
Complemento a 2 di W	11111101
Somma del letterale con il complemento a 2 di W	00000001
Dato che c'è riporto nella somma si attiva il flag C	
Valore di W prima dell'istruzione	00000011
Valore letterale da cui sottrarre	00000010
Complemento a 2 di W	11111101
Somma del letterale con il complemento a 2 di W	11111111
Dato che non c'è riporto nella somma si attiva il flag C	

Esecuzione dell'istruzione sublw.



Esempio tipico dell'uso dell'istruzione sublw.

Così come ci mostra l'esempio, vediamo che quando il risultato di un'operazione è un valore negativo, quello che si scrive in W non è il valore, ma il suo complemento a due, e il fatto che sia negativo viene segnalato da uno zero nel bit del carry.

ESEMPI CON L'ISTRUZIONE SUBLW

Oltre alla funzione classica di eseguire la sottrazione, l'istruzione sublw si utilizza per fare le comparazioni. Immaginiamo di aver formato un numero con gli interruttori, e che questi siano collegati, ad esempio, alla porta B. Questo numero può essere il numero di una chiave, ed è necessario sapere se esso possiede un valore determinato per convalidarla, e accendere un LED rosso per proibire l'accesso e uno verde per accettarlo. Sottraendo il valore che abbiamo in ingresso con il valore di riferimento (ma potremmo anche fare il contrario), possono succedere due cose: che il resto della sottrazione dia come risultato zero o che dia un numero diverso da zero, qualunque esso sia. Se il risultato è zero, significa che i due numeri sono uguali, e ottenendo questo risultato si attiverà il flag Z (di zero). Se dopo la sottrazione si verifica il valore di Z, si saprà se i valori comparati erano uguali oppure no, a seconda che il flag Z valga 1 o 0 rispettivamente.

Scheda delle istruzioni: L'istruzione SUBWF

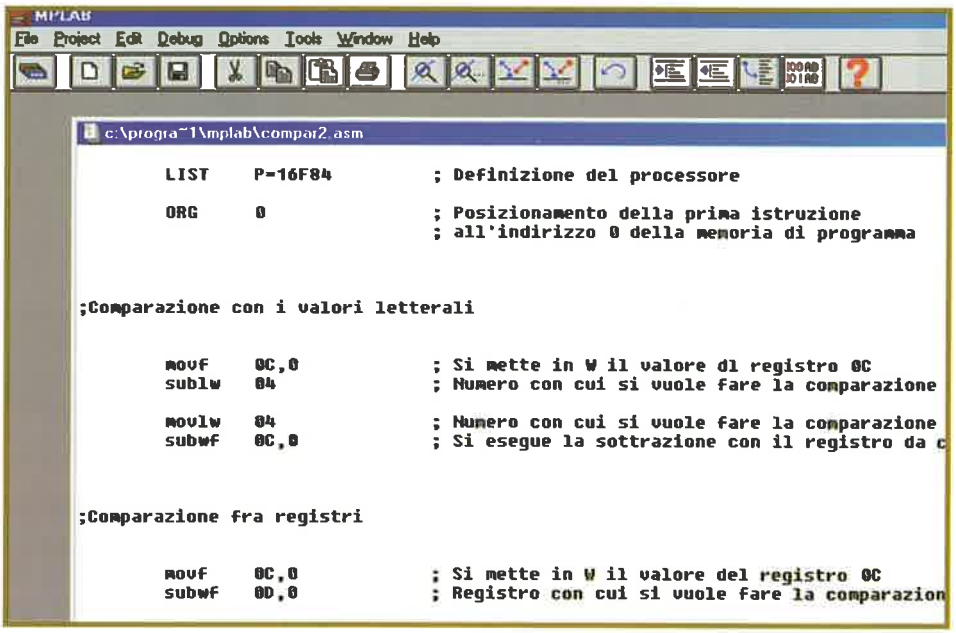
È la seconda istruzione di sottrazione e sottrae ad un registro chiamato "f" il valore che contiene il registro di lavoro W. Il risultato di questa operazione rimane in f o in W, a seconda del valore del parametro d.

Così come l'istruzione precedente, vengono coinvolti i flag C, DC e Z nella forma già descritta.

ESEMPI CON L'ISTRUZIONE SUBWF

L'uso di questa istruzione è simile a quello dell'istruzione sublw, che abbiamo appena visto. Entrambe servono per sottrarre valori e per fare comparazioni.

Comunque, a seconda del tipo di dato da comparare, non sempre si può utilizzare la sublw, a meno di casi semplici, dato che per eseguire delle comparazioni la subwf è più completa. Nel caso della comparazione fra il valore di un registro e un valore costante determinato, sarebbe a dire con un valore letterale, è indifferente l'uso di una o dell'altra istruzione, ma nel caso di comparazione fra registri è necessario l'utilizzo dell'istruzione subwf.



Esempio di comparazione con le istruzioni sublw e subwf.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
s u b w f	f	d

Operazione:	Sottrae al contenuto del registro **f** il contenuto del registro di lavoro **W**. Se **d=0** il risultato si porta nel registro di lavoro **W**, e se **d=1** si lascia nello stesso registro **f**.
Cicli:	1
Codice OP:	00 0010 dfff ffff
Flag:	C, DC, Z

Caratteristiche dell'istruzione subwf.

Applicazione pratica: un altro programma con il TMR0

A poco a poco stiamo conoscendo l'uso delle varie risorse del PIC. Abbiamo studiato i registri, le porte di ingresso/uscita e nell'ultimo esercizio proposto abbiamo parlato del TMR0. Però il TMR0 si può utilizzare in due modi: uno come temporizzatore, e l'altro come contatore di impulsi esterni. In questo esercizio lo utilizzeremo nel secondo modo.

Leggete con attenzione l'enunciato dell'esercizio proposto e pensate a come risolverlo.

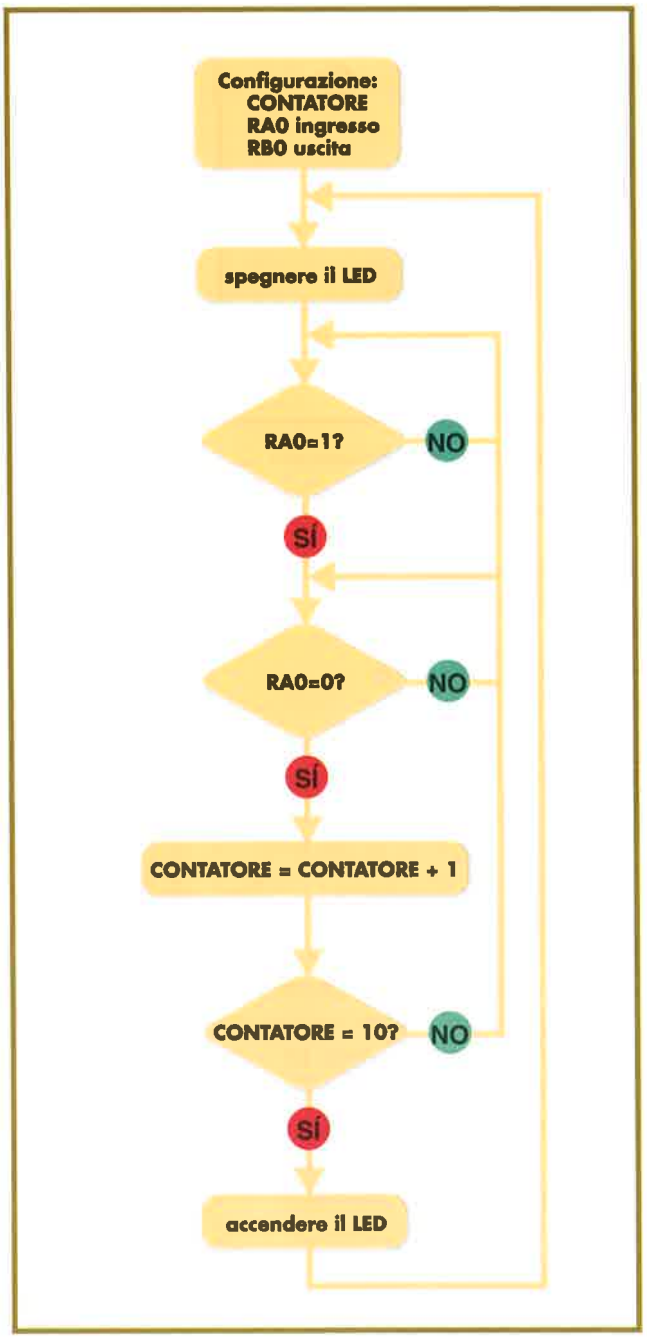
Abbiamo un nastro trasportatore sul quale stanno passando i pezzi che vogliamo contare. Per contare il numero dei pezzi posizioneremo un pulsante che sarà attivato dal passaggio di ogni singolo pezzo. Per ogni gruppo di 10 pezzi attiveremo un diodo LED che ci avviserà, ad esempio, che i pezzi possono essere impacchettati.

Enunciato dell'esercizio proposto.

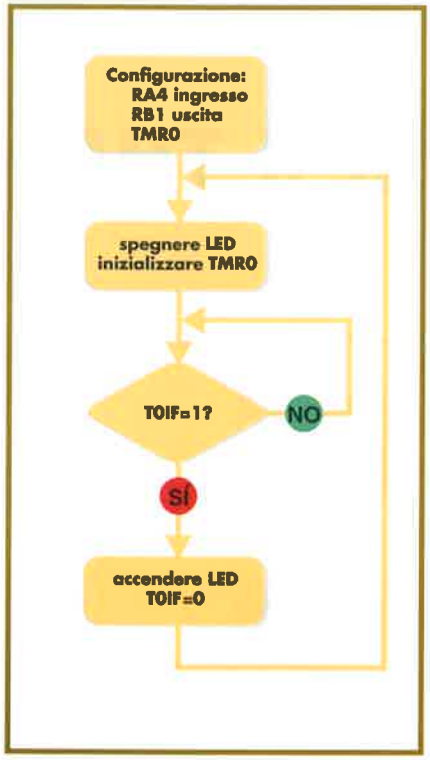
PRIMO ORGANIGRAMMA

La prima idea potrebbe essere l'utilizzo di un contatore da incrementare tramite il programma, ogni volta che rileveremo un impulso (passaggio da 0 a 1 e nuovamente a 0, oppure il contrario) sul piedino a cui abbiamo collegato il pulsante.

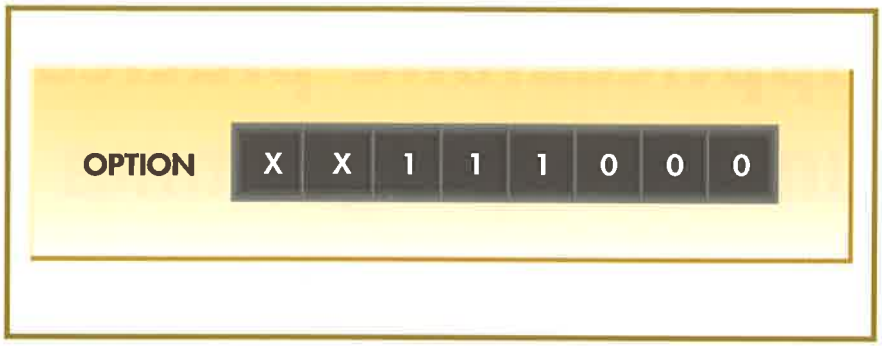
Ogni volta che aumenta il registro contatore, dovremo comparare al numero a cui vogliamo arrivare, in questo caso al 10. Il programma potrebbe essere risolto in modo soddisfacente già con il primo organigramma esposto. Dato che il nostro programma deve solo svolgere una funzione, non importa se impiega tutto il tempo nel testare un ingresso, incrementare un contatore, e compararlo con un valore dato; però se oltre a



Primo organigramma dell'esercizio proposto.



Secondo organigramma dell'esercizio proposto.



Configurazione del registro OPTION per questo esercizio.

questo dovesse controllare anche che i pezzi non siano difettosi, il movimento del nastro trasportatore su cui passano i pezzi, e il braccio del robot che li prende, questo sottrarrebbe tempo al processore, dato che ci sono modi migliori di fare la stessa cosa.

ORGANIGRAMMA DEFINITIVO

Il pulsante attivato dai pezzi, si può collegare al piedino RA4, il quale si può configurare per introdurre impulsi nel TMR0. Ogni volta che arriva un fronte di salita o di discesa, il TMR0 si autoincrementa senza la necessità di eseguire alcuna istruzione. In più, configurando il TMR0 in modo adeguato, potremo essere avvisati del termine del conteggio delle 10 unità. Anche se questo secondo modo di procedere è migliore del primo, non è ancora quello ottimale, però non potremo migliorare ulteriormente il programma senza l'utilizzo degli interrupt.

Come vedremo, abbiamo eliminato tutti i riferimenti al conteggio – che è controllato direttamente dall'hardware del TMR0 – ad eccezione del suo termine; questo controllo è realizzato verificando il valore del flag T0IF, che passerà automaticamente a 1 al termine del conteggio, e che dovrà essere rimesso a zero per un nuovo conteggio.

CONFIGURAZIONE DEL REGISTRO OPTION

In uno dei capitoli precedenti, abbiamo visto come configurare il registro OPTION per realizzare un determinato conteggio.

Dovremo impostare questo registro ogni volta che desideriamo lavorare con il TMR0. Per l'esercizio in corso, in cui il TMR0 dovrà contare gruppi di 10, non è

VALORE DEL TMR0	
00000011	Valore che si vuole contare
11111100	Complemento a 1 del valore da contare
11111101	Complemento a 2 del valore da contare, che è quello che si deve mettere nel TMR0
11111110	Valore che si incrementa nel TMR0, con il primo impulso o istruzione
11111111	Valore che si incrementa nel TMR0 con il secondo impulso o istruzione
00000000	Valore che si incrementa nel TMR0 con il terzo impulso o istruzione. A questo punto avviene l'overflow e si attiva un flag per avvisarci di questo

Esempio del valore che dobbiamo introdurre nel TMR0.

necessario il divisore di frequenza, e dobbiamo configurare il contatore in modo che gli incrementi siano prodotti con gli impulsi che arrivano dall'esterno e non con gli impulsi interni del clock.

I primi tre bit (0-2) rappresentano il valore del divisore. Come abbiamo già detto, per questo esercizio non è richiesto nessun divisore di frequenza, però assegnando il valore "000" al divisore per il TMR0, risulterebbe di 1:2. Pertanto dobbiamo combinare questo dato insieme al bit 3, che avendo valore 1 assegna il divisore al Watchdog in modo 1:1.

Con i valori dei bit 5 e 4 impostiamo rispettivamente l'incremento del TMR0 mediante impulsi, ed il fronte di discesa per questi impulsi. Le X riflettono valori che non hanno alcuna influenza sul risultato di questo esercizio.

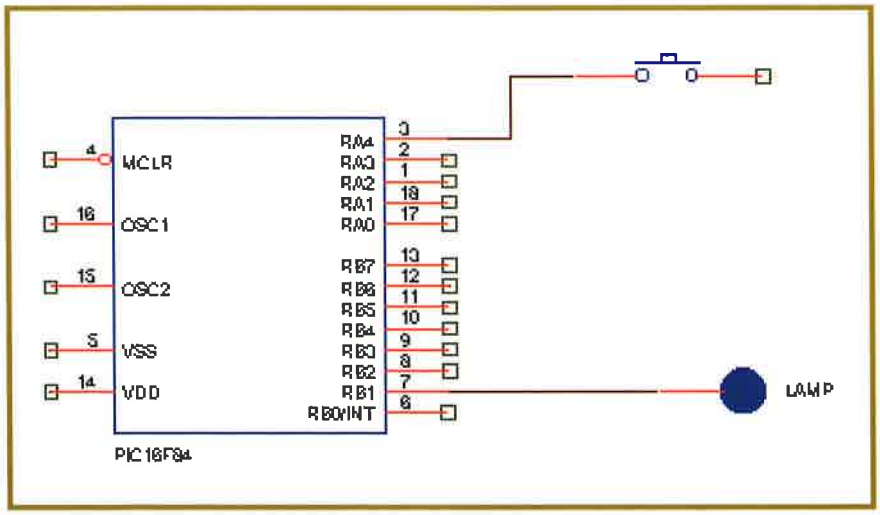
VALORE DEL TMR0

Un altro importante dato da considerare quando si lavora con il TMR0, è il valore con cui viene caricato. Per non creare equivoci è necessario conoscere come lavora il TMR0; questo contatore si autoincrementa ad ogni istruzione o impulso esterno, oppure ogni due o tre istruzioni o impulsi esterni, sino ad arrivare al suo valore massimo, che vale FF, dato che si tratta di un registro a 8 bit, per poi passare dal valore massimo al valore minimo, a 00.

Questo passaggio è conosciuto con il nome di overflow, e ci avvisa che il conteggio è terminato, anche se in realtà il conteggio non si ferma ma prosegue. Imparato questo, se vogliamo che il TMR0 conti fino a "3" non possiamo caricare questo valore direttamente, altrimenti otterremmo un conteggio di 3,4,5,6,7,8..., sino a FF, avverrebbe l'overflow e ne saremmo avvisati.

Ogni volta che vogliamo contare un valore dobbiamo introdurre il complemento a 2 di questo valore. Nella figura allegata, è riportato un esempio per il valore 3, in cui si vede in modo semplice e rapido come si compie questa norma.

La stessa tecnica si applicherà per numeri più grandi.



Schema elettrico dell'esercizio proposto.

```
TRISA       EQU     0x85
TRISB       EQU     0x86

            ORG     0                   ; Il programma inizia
            goto    INIZIO              ; all'indirizzo 0

INIZIO      bsf     STATUS, RP0         ; Si passa al banco 1
            movlw   B'00000000'         ; Porta B uscita
            movwf   TRISB
            movlw   B'00010000'         ; RA4 ingresso, il resto uscite
            movwf   TRISA
            movlw   B'00111000'         ; Configurazione del TMR0
            movwf   OPTION_REG
            bcf     STATUS,RP0          ; Si passa al banco 0

VALORI      bcf     PORTB,LED           ; Disattiva LED
            movlw   0xF6                ; Carica il TMR0 con il complemento
            movwf   TMR0                ; a 2 di 10

CICLO       btfss   INTCON,T0IF         ; Sono passati 10 pezzi?
            goto    CICLO               ; No, attendere
            goto    ACCENDERE           ; Si, avvisa con il LED

ACCENDERE   bsf     PORTB,LED           ; Accendere il LED
            bcf     INTCON,T0IF         ; Azzera il flag per la prossima temp
            goto    VALORI              ; Torna all'inizio

            END
```

Programma finale dell'enunciato proposto.

SCHEMA ELETTRICO

Nella figura è riportato lo schema elettrico semplificato. Montandolo dovremo collegare oltre all'alimentazione e la massa, il cristallo di quarzo e il pulsante di Reset per livello basso su MCLR.

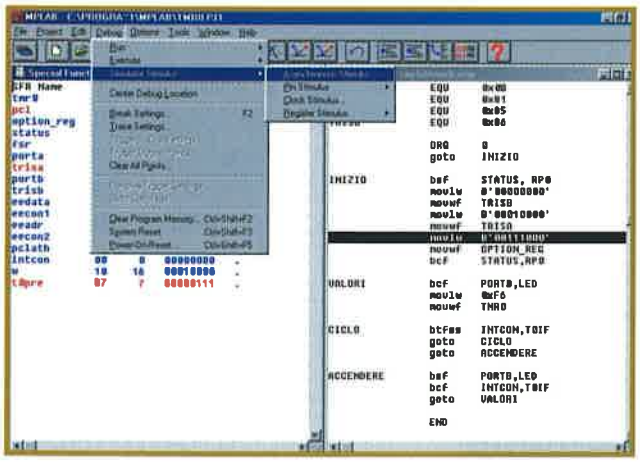
Dei due elementi montati il pulsante deve essere proprio su questa linea, perché è l'unica che è collega-

ta internamente con il TMR0, per fare in modo che si incrementi. Il LED è stato collegato al piedino RB1, ma potrebbe essere collegato a qualsiasi altro piedino di I/O.

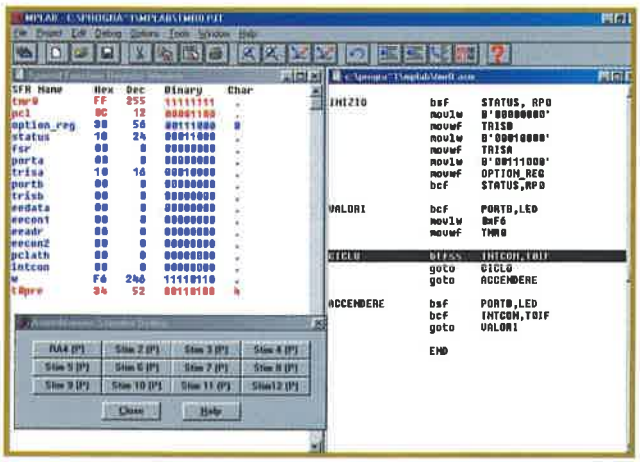
PROGRAMMA

Il programma, dopo aver considerato tutti i passi precedenti, apparirà come mostrato nella figura. Come novità, è possibile notare il nome assegnato ai registri e ai bit.

Anche se potremmo chiamarli come vogliamo, ferma restando la funzione di ognuno e il modo di utilizzarli, normalmente hanno nomi che si possono considerare universali, in modo che tutti quelli che utilizzano l'assembler del PIC abbiano un linguaggio comune.



Scelta dell'introduzione di stimoli esterni.



Stato della simulazione subito prima dell'overflow del TMR0.

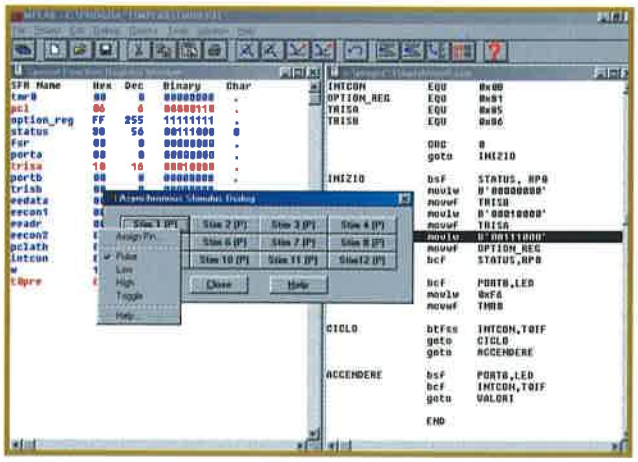
PROVA DEL PROGRAMMA

La simulazione del programma precedente ci pone una nuova sfida, dobbiamo introdurre impulsi tramite RA4 in modo che il TMR0 si incrementi.

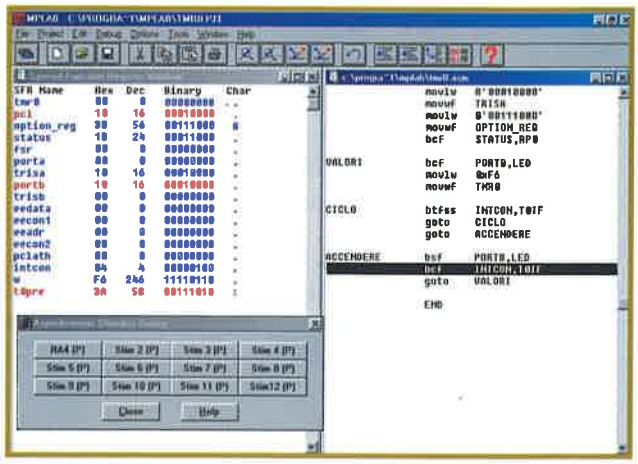
Se proviamo a farlo con l'opzione Window>Modify, vedremo che è impossibile.

Lo dobbiamo fare con l'opzione Debug>Simulator Stimulus>Asynchronous Stimulus. Da qui si aprirà una finestra in cui fare la scelta dei pin da utilizzare per introdurre tutti i dati necessari, e la forma di questi dati: impulso, livello basso, livello alto, o zero e uno alternativamente.

Nel nostro caso sceglieremo gli impulsi tramite RA4. In seguito mostreremo lo stato della simulazione, subito prima che il TMR0 vada in overflow, e poi dopo l'overflow, al momento dell'accensione del LED.



Scelta per l'inserzione di impulsi su RA4.



Stato della simulazione dopo l'overflow del TMR0 e accensione del LED.

Scheda delle istruzioni: l'istruzione CLRW

Questa è un'altra istruzione per la gestione dei registri, anche se questa volta si tratta del registro di lavoro W. L'operazione che compie non è altro che la cancellazione (clear) del contenuto del registro, ossia la sostituzione del valore del registro con 0. Come per molte altre istruzioni, producendo questo risultato si attiva il flag Z.

Dato che si agisce sul registro W non sono necessari ulteriori parametri addizionali.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
c l r w		

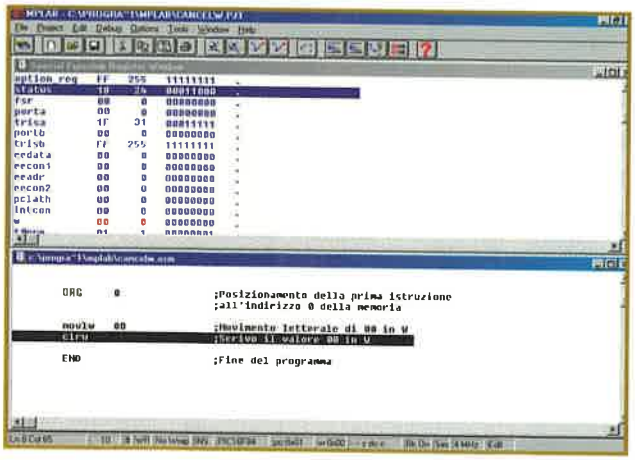
Operazione: il registro di lavoro **W** viene caricato con il valore 00.
Cicli: 1
Codice OP: 00 0001 0000 0011
Flag: Z

Caratteristiche dell'istruzione clrw.

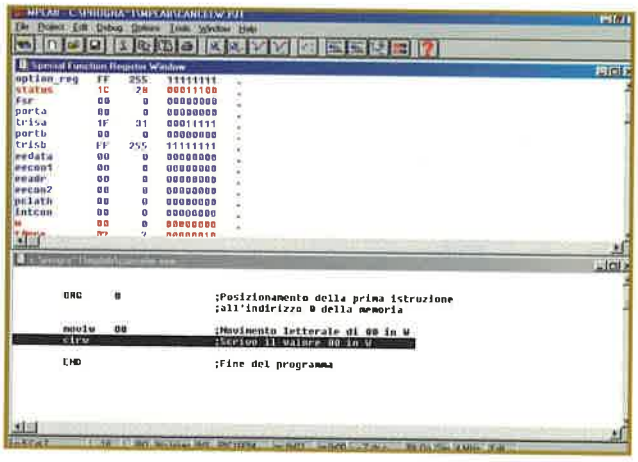
ESEMPI CON L'ISTRUZIONE CLRW

Come si può osservare nelle due figure seguenti, l'uso dell'istruzione clrw è simile a quello dell'istruzione movlw 00. L'unica differenza sta nel fatto che dopo clrw si attiva il flag Z, dato che il risultato presente W sarà 0, mentre dopo l'istruzione movlw 00 questo flag non viene attivato.

Per fare la prova con MPLAB e vedere bene quando variano i registri si raccomanda di cambiare i valori dei registri STATUS e W prima dell'esecuzione di ogni istruzione. Vi suggeriamo di introdurre nel registro STATUS il valore 00, anche se in pratica i bit 3 e 4 non si possono cambiare, e nel registro W il valore FF. Osserviamo che eseguendo l'ultima istruzione la riga evidenziata in nero non si modifica.



Valore dei registri STATUS e W dopo l'esecuzione dell'istruzione movlw 00.



Valore dei registri STATUS e W dopo l'esecuzione dell'istruzione clrw.

Scheda delle istruzioni: l'istruzione CLRF

Come l'istruzione precedente, anche questa realizza la cancellazione di un registro, in questo caso della memoria RAM dei dati. L'indirizzo del registro si specifica come parametro. Il suo flag associato è nuovamente Z, che verrà messo a 1 dato che il risultato sarà 0.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
c l r f		f

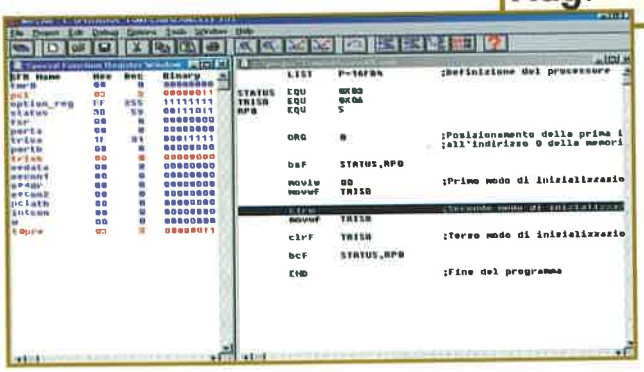
Operazione: il registro **f** viene caricato con il valore 00.
Cicli: 1
Codice OP: 00 0001 1fff ffff
Flag: Z

Caratteristiche dell'istruzione clrf.

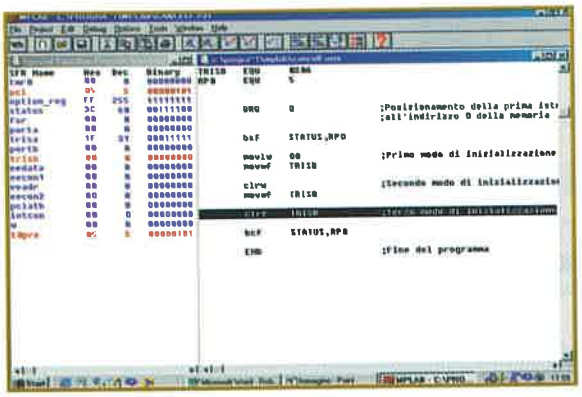
ESEMPI CON L'ISTRUZIONE CLRF

Lavorando con le porte di ingresso e uscita, sovente è stato necessario inizializzare tutte le linee di una porta come uscita, per cui dovevamo caricare il registro di configurazione con degli zero. Questo veniva fatto con la combinazione di due istruzioni: la movlw 00 e la movwf f, dove f era il registro da configurare.

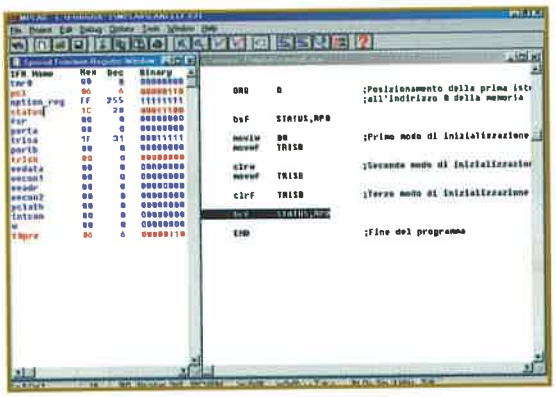
A partire da questo momento, con l'uso delle due nuove istruzioni clrw e clrf, avremo diverse opzioni. Per provare adeguatamente i differenti casi, dovremo inizializzare prima di ogni blocco i registri STATUS (00), W (FF) e TRISB (FF).



Valore dei registri STATUS e TRISB dopo l'esecuzione della prima opzione.



Valore dei registri STATUS e TRISB dopo l'esecuzione della seconda opzione.



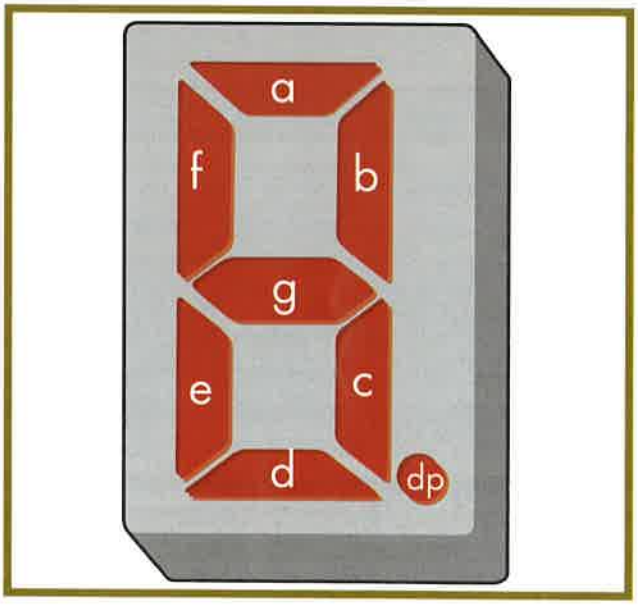
Valore dei registri STATUS e TRISB dopo l'esecuzione della terza opzione.

Nel primo caso non si attiva il flag Z per nessuna delle due istruzioni, nel secondo caso si modifica con la terza istruzione, così come nel terzo blocco.

Applicazione pratica: rappresentazione dei dati sul display a 7 segmenti

Tutti i sistemi, e il computer ne è un esempio, si possono differenziare in tre parti: l'ingresso dei dati, l'elaborazione dei medesimi e la rappresentazione del risultato. I dispositivi di ingresso e la rappresentazione dei risultati sono chiamati interfaccia utente, e stanno assumendo sempre maggiore importanza, rendendo le interfacce sempre più comode e facili da utilizzare. Un programma può essere molto buono e fare molte cose, però se la sua interfaccia è complicata da utilizzare o richiede molte conoscenze, azioni ripetitive, ecc. è condannato all'insuccesso.

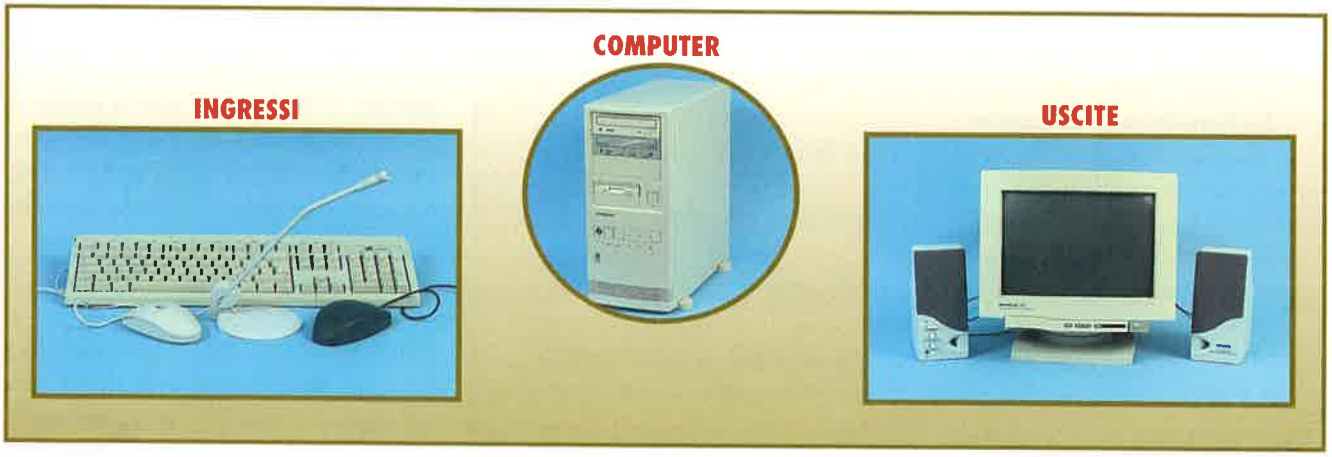
Non bisogna però confondere le interfacce di ingresso o di uscita più adatte, con le più moderne; il miglior modo per introdurre caratteri in un programma resta la tastiera, e per muoversi sul video, il mouse. Nel lavoro con i microcontroller succede qualcosa di simile. Sino ad ora abbiamo lavorato solo con interrupt per introdurre i dati, e con i diodi LED per visualizzare i risultati. Ora faremo un passo in più con l'utilizzo di un nuovo elemento, il display a 7 segmenti. Sicuramente la maggioranza di voi avrà utilizzato questo elemento come utente. È quello che indica il turno in pescheria o al supermercato, però ora saremo noi a programmarlo affinché gli altri lo utilizzino.



Rappresentazione del display a 7 segmenti.

IL DISPLAY A 7 SEGMENTI

Quando abbiamo bisogno di rappresentare dei numeri, la soluzione più semplice ed economica è il display a 7 segmenti. Come indica il suo nome è un dispositivo formato da 7 segmenti che non sono altro che diodi LED



L'interfaccia uomo macchina è sempre più importante.

Valore interruttori			Valore decimale	dp	g	f	e	d	c	b	a	Valore esadecimale
0	0	0	0	0	0	1	1	1	1	1	1	3F
0	0	1	1	0	0	0	0	0	1	1	0	06
0	1	0	2	0	1	0	1	1	0	1	1	5B
0	1	1	3	0	1	0	0	1	1	1	1	4F
1	0	0	4	0	1	1	0	0	1	1	0	66
1	0	1	5	0	1	1	0	1	1	0	1	6D
1	1	0	6	0	1	1	1	1	1	0	1	7D
1	1	1	7	0	0	0	0	0	1	1	1	07

Tabella di trasformazione dei valori binari al codice per i 7 segmenti a catodo comune.

di forma allungata, disposti in modo da formare il numero 8. Questi segmenti sono numerati, come mostra la figura, con lettere consecutive dalla 'a' alla 'g' iniziando dal segmento superiore e continuando in senso orario per terminare con il segmento centrale. Oltre ai 7 segmenti c'è un punto decimale nella parte inferiore destra, indicato come dp, che a volte si ripete anche nella parte sinistra.

ENUNCIATO DELL'ESERCIZIO

L'esercizio che ora vi proponiamo non ha grandi pretese, vuole solo aiutare a comprendere il funzionamento del display a 7 segmenti per il suo uso in progetti più ambiziosi. Leggiamo pertanto l'enunciato e mettiamoci al lavoro.

Quello che si richiede è di trasformare il valore binario che un utente introduce tramite gli interruttori in un codice nuovo, in modo che si illuminino i differenti display a seconda del numero che si vuole rappresentare. I numeri possibili sono dallo 0 al 7, dato che i valori degli interruttori potranno variare da 000 a 111. Prima di fare la tabella corrispondente dobbiamo conoscere un dato in più sul display. Tutti sappiamo che un diodo LED ha due terminali, conosciuti come anodo e catodo, e che se colleghiamo il catodo a terra e l'anodo a un PIC dobbiamo far uscire dal piedino del PIC un 1 logico per accenderlo, al contrario se l'anodo è collegato a 5 V e il catodo al PIC, dobbiamo fornire uno 0 logico per accenderlo. Con i display succede la stessa cosa, solo che tutti i segmenti (LED interni) sono uniti fra loro per l'anodo o per il catodo, dando quindi il nome al display che potrà essere di anodo comune o di catodo comune rispettivamente. In un display di anodo comune, si dovranno mandare degli zero per accendere i differenti segmenti, e in un display a catodo comune ogni segmento si accenderà con il suo 1 corrispondente. In questo esercizio simuliamo il display che si accende con uscita a 1, cioè un display a catodo comune. Vedi tabella in alto.

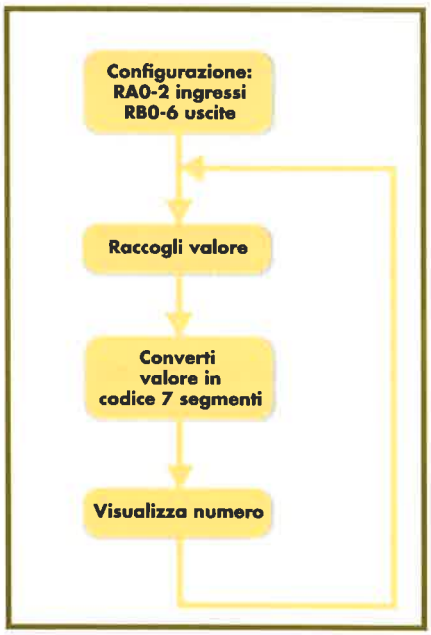
Alcuni digit possono essere rappresentati in più di una forma, per cui dovremo scegliere. Dato che in questo esempio non si usa il punto decimale, questo rimarrà spento. Il motivo per cui si passa ai valori esadecimali è che si possono maneggiare più efficacemente i dati, mentre potrebbero essere confusi facilmente con tanti 1 e 0.

Si vuole rappresentare per mezzo di un display a 7 segmenti il valore introdotto tramite tre interruttori, in modo che il valore binario di questi ultimi sia visualizzato in decimale. Questa può essere una piccola parte di un progetto più grande, in cui un fattore importante sia la rappresentazione dei numeri - dato che non tutti gli utenti potrebbero non essere a conoscenza del sistema binario, come succederebbe se si visualizzassero le informazioni tramite i diodi LED.

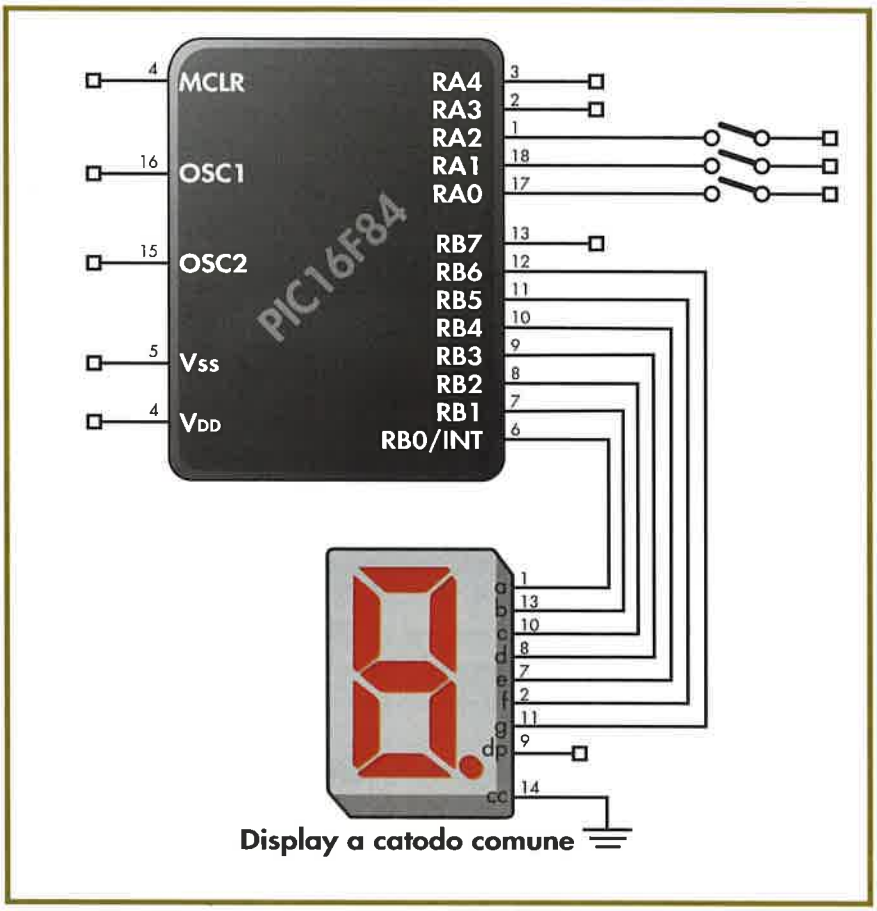
Enunciato dell'esercizio proposto.

ORGANIGRAMMA

L'organigramma per questo esercizio è molto semplice, visto il modo con cui utilizzeremo il display a 7 segmenti. Forse qualcuno starà pensando che bisognerà testare il valore degli inter-



Organigramma dell'esercizio proposto.



Schema elettrico dell'esercizio proposto.

ruttori, e a seconda di questo, mostrare i differenti valori sul display, però per questa simulazione non sarà necessario.

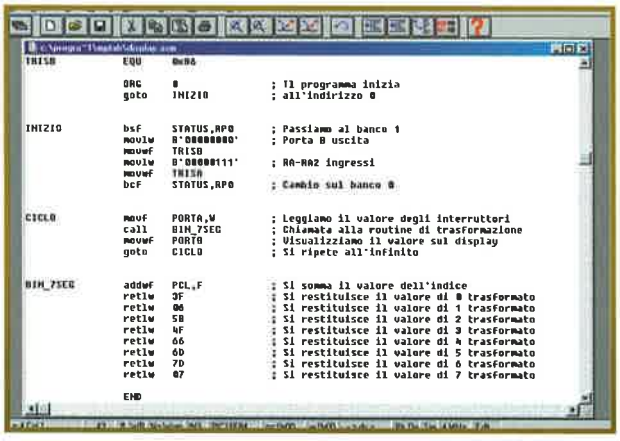
Configureremo la porta B come uscita, per collegare ad essa il display, e la porta A come ingresso per gli interruttori. Dopo aver raccolto il valore degli interruttori si realizzerà una conversione, secondo la tabella vista in precedenza, e si visualizzerà il risultato sul display a 7 segmenti.

SCHEMA ELETTRICO

Nella figura dello schema elettrico mostriamo la connessione degli elementi da utilizzare. Non sono riportati i collegamenti dell'alimentazione positiva e negativa, il quarzo né il pulsante di reset. Il punto decimale del display è stato lasciato in aria. Se si volesse collegarlo, si dovrebbe fare con la linea RB7. Gli interruttori sono collegati alle linee meno significative della porta A, per rendere più semplice la conversione.

PROGRAMMA

Il programma risultante è illustrato nella figura: si compone, oltre alle istruzioni tipiche di configurazione delle porte, di un ciclo principale che corrisponde con i passi visti nell'organigramma e che è ripetuto all'infinito. Per ultimo, la subroutine di conversione dei valori binari al codice per i 7 segmenti, che abbiamo già spiegato parlando dell'istruzione retlw.

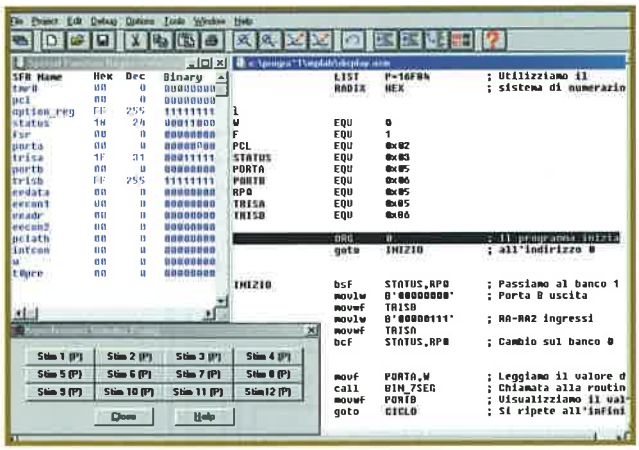


```
TRISB   EQU     0x86

        ORG     0               ; Il programma inizia
        goto    INIZIO          ; all'indirizzo 0

INIZIO  bsf     STATUS,RP0      ; Passiamo al banco 1
        movlw   B'00000000'     ; Porta B uscita
        movwf   TRISB
        movlw   B'00000111'     ; RA-RA2 ingressi
        movwf   TRISA
        bcf     STATUS,RP0      ; Cambio sul banco 0

CICLO   movf    PORTA,W         ; Leggiamo il valore degli interruttori
        call    BIN_7SEG        ; Chiamata alla routine di trasformazione
        movwf   PORTB           ; Visualizziamo il valore sul display
        goto    CICLO           ; Si ripete all'infinito

BIN_7SEG addwf  PCL,F           ; Si somma il valore dell'indice
        retlw   3F              ; Si restituisce il valore di 0 trasformato
        retlw   06              ; Si restituisce il valore di 1 trasformato
        retlw   5B              ; Si restituisce il valore di 2 trasformato
        retlw   4F              ; Si restituisce il valore di 3 trasformato
        retlw   66              ; Si restituisce il valore di 4 trasformato
        retlw   6D              ; Si restituisce il valore di 5 trasformato
        retlw   7D              ; Si restituisce il valore di 6 trasformato
        retlw   07              ; Si restituisce il valore di 7 trasformato

        END
```

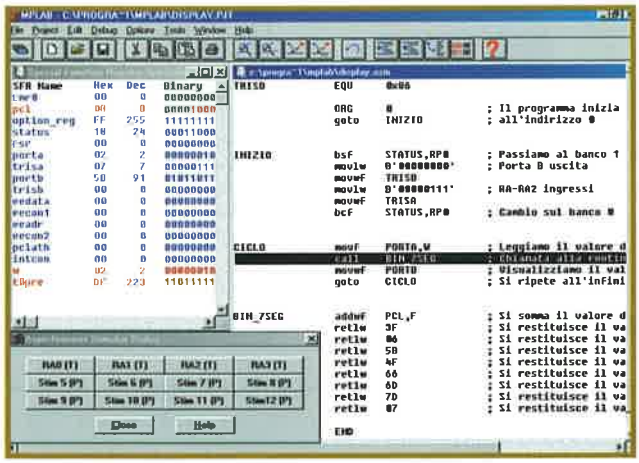
Programma dell'enunciato proposto.

PROVA DEL PROGRAMMA

La simulazione del programma non sarà semplice, dato che non abbiamo nel simulatore un display a 7 segmenti, su cui vedere i valori convertiti. Dovremo accontentarci di vedere i valori che otteniamo dalla tabella, sul



Configurazione dell'introduzione degli stimoli esterni.



Un momento della simulazione.

registro PORTB. Il valore mostrato dipenderà dal valore introdotto sulle linee RA0-RA2.

Ricordate che il modo di introdurre questi dati si configura dall'opzione Debug>Simulator Stimulus>Asynchronous Stimulus.

Abbiamo scelto l'opzione Toggle, che è quella in grado di simulare meglio il funzionamento degli interruttori.

Ogni impulso che introdurremo tramite una linea farà in modo che questa cambi il suo valore, cioè se è a 0 passera a 1 e se è a 1 passerà a 0.

La simulazione si realizzerà con l'opzione Debug>Run>Animate. Nella figura è riportato un momento di questa simulazione. Fate attenzione al registro PORTA, che contiene il valore degli interruttori, e al registro PORTB che contiene il valore convertito. I valori devono coincidere con la tabella che abbiamo visto.

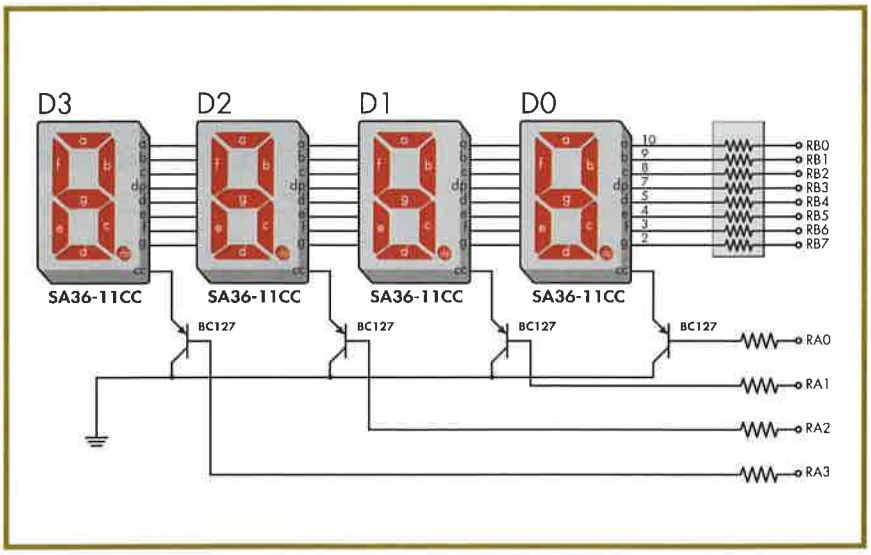
COMPLICHIAMO L'ENUNCIATO

Che cosa ne dite se vi proponiamo di utilizzare non un solo display, ma diversi, per rappresentare numeri con più di un digit? Di sicuro starete pensando: impossibile, il PIC 16F84 ha solo 13 linee di ingresso/uscita e ogni display ne utilizza 8!

Avete ragione, però si può sempre ricorrere a qualche piccolo trucco. In questo caso collegheremo diversi display in parallelo, in modo che l'informazione mandata a uno di essi sia ricevuta anche da tutti gli altri. Affinché non sia visualizzato lo stesso dato su tutti i display, che sarebbe la stessa cosa di averne uno solo, bisogna utilizzare le linee addizionali, una per ogni display, che li attiveranno in modo consecutivo tramite dei transistor; in modo che se ne attiva solo uno per volta, al momento di visualizzare il dato che gli compete.

Ogni attivazione sarà associata al dato da visualizzare per quel display. Questo processo è conosciuto con il nome di scansione, ed è simile a quello realizzato, ad esempio nei televisori, per mostrare l'immagine. Se si realizza l'attivazione di ogni display con sufficiente velocità, grazie alla modalità di funzionamento dell'occhio umano, si produrrà la sensazione ottica che tutti i display siano attivi allo stesso tempo, ognuno con il proprio dato.

Per realizzare questa simulazione però, servirebbe qualcosa in più dell'immaginazione.



Schema elettrico per l'utilizzo di diversi display.

Scheda delle istruzioni: l'istruzione ANDLW

Questa è la prima istruzione logica che troviamo. Il suo compito è realizzare la moltiplicazione logica dei due valori: un valore letterale e il contenuto del registro di lavoro W. Il risultato rimane sul registro di lavoro W. Se il risultato dell'operazione è zero, si attiva il flag Z.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
a n d l w	k	

Operazione: realizza l'operazione logica AND fra un valore letterale e un valore del registro di lavoro W.
Cicli: 1
Codice OP: 11 1001 kkkk kkkk
Flag: Z

Caratteristiche dell'istruzione andlw.

È considerato falso il valore 0, e vero tutto ciò che non è falso, cioè tutto ciò che ha valore diverso da 0. In alcuni linguaggi di programmazione, questa istruzione può avere più di due operandi. Se tutti i suoi operandi sono veri darà come risultato un valore vero, e se uno di essi è falso, anche il risultato sarà falso. Nella figura è mostrato come funzionerebbe questa istruzione se lavorasse sui registri, cosa che succede in alcuni linguaggi, e come lo fa nel PIC, dove lavora sui bit di un registro. La prima tabella mostra il funzionamento generale e la seconda un esempio particolare. Partendo da due valori per un registro e per un letterale, se si lavora con il registro al completo, il risultato finale sarà 0 (falso) solamente se uno dei due valori, o entrambi, sono 0. In tutti gli altri casi il risultato sarà vero, però non ha nessun senso dare un valore, infatti le istruzioni che operano in questo modo non lo danno: lasciano solo un valore V in una variabile di tipo logico o booleano. Se si lavora con i bit, avremo il risultato di ogni coppia di bit, e può succedere, come nell'esempio, che due valori che in principio non hanno valore 0, diano questo risultato combinandosi con una istruzione and logica.

ESEMPI CON L'ISTRUZIONE ANDLW

Mentre un'istruzione aritmetica utilizza come unità base un registro completo, un'istruzione logica opera con ognuno dei bit di questo registro in modo indipendente. Vediamo questa differenza con l'istruzione di cui ci stiamo occupando. L'istruzione and, chiamata anche moltiplicazione, suppone che i valori con cui si opera siano veri (V) o falsi (F).

Tabella generale

Valore 1	Valore 2	Risultato
F	F	F
F	V	F
V	F	F
V	V	V

Lavoro con registri

Registro 1	Letterale 2	Risultato
01010101	10101010	V

Lavoro con bit

Registro 1	Letterale 2	Risultato
01010101	10101010	00000000 (F)

Funzionamento dell'istruzione andlw.

Scheda di istruzioni: l'istruzione ANDWF

Fino a questo momento la maggior parte delle istruzioni viste hanno due modalità, per quanto riguarda gli operandi con cui lavorano; in questo caso, anche l'istruzione and ha questa seconda forma: la andwf. Come si può immaginare, per comparazione con le altre istruzioni, la differenza sta nel fatto che le operazioni si realizzano fra il valore del registro di lavoro W e il valore del registro f, lasciando il risultato in uno o nell'altro, a seconda del valore del secondo parametro. Inoltre verrà coinvolto il flag Z.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
a n d w f	f	d

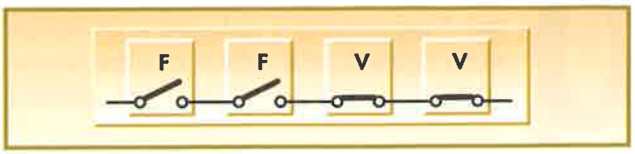
Operazione: realizza l'operazione logica AND fra il valore del registro f e il contenuto del registro di lavoro W. Se d = 0 il risultato si lascia nel registro di lavoro W e se d = 1 si lascia nel medesimo registro f.
Cicli: 1
Codice OP: 00 0101 dfff ffff
Flag: Z

Caratteristiche dell'istruzione andwf.

Nella figura corrispondente si realizza la rappresentazione grafica dell'istruzione and mediante un circuito formato da alcuni interruttori. È sufficiente che un solo interruttore sia aperto (valore falso) perché la lampadina non si accenda (valore falso).

ESEMPI DELL'ISTRUZIONE ANDWF

L'istruzione andwf opera nello stesso modo della andlw, cioè bit a bit, per cui l'esempio già esposto serve anche per questa istruzione. Così l'esempio che vedremo ora, sarà simile per entrambe le istruzioni. Queste, e tutte le logiche in generale, si utilizzano per fare quelle che vengono chiamate maschere. Una maschera permette di operare su diversi bit di un registro senza alterare il resto.



Circuito con interruttori per rappresentare l'istruzione and.

Questo è utile quando un registro si utilizza per diverse cose allo stesso tempo. Per esempio, nei programmi trattati fino a questo momento, le porte A e B si utilizzavano come ingressi o come uscite, però mai con le due funzioni allo stesso tempo; non perché non sia possibile, ma per maggior comodità. Immaginiamo ora che la porta B sia collegata a quattro interruttori e quattro diodi LED: si tratta di conoscere il valore di questa porta per realizzare questa o quella operazione, o di agire sui LED senza interferire con il valore degli ingressi. Per non falsare i dati, dovremo realizzare differenti maschere. Nella figura si esegue quest'operazione con l'istruzione andlw o con la andwf, essendo queste intercambiabili.

Annullamento dei bit più significativi della Porta B, rimanendo con i valori degli interruttori situati sui quattro bit meno significativi.

```
movf    PORTB,W
andlw   b'00001111'
```

Se PORTB = 01101100, dopo l'operazione avremo che W = 00001100, dato che gli zero del letterale fanno in modo che qualsiasi valore diventi zero, mentre il valore 1 lascia che il risultato dipenda dall'altro bit.

Azzeramento dei led della Porta B senza coinvolgere gli interruttori.

```
movlw   0F
andwf   PORTB,F
```

Anche se l'operazione sembra la stessa del caso precedente, nel primo caso avviene in W, per cui la Porta B non viene modificata, mentre nel secondo caso si realizza direttamente sulla porta, cambiandone il valore.

Esempi con le istruzioni andlw e andwf.

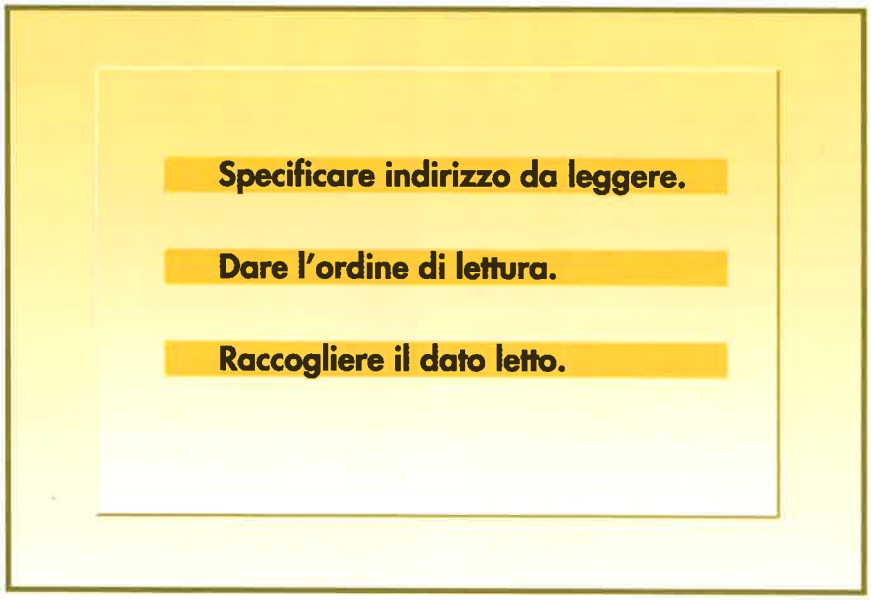
Applicazione pratica: scrivere chiavi nella EEPROM

In numerose applicazioni è necessario scrivere una serie di dati che dovranno poi essere recuperati. Se questi dati si memorizzano e si recuperano nella medesima esecuzione di programma, possono essere scritti nella memoria RAM che, anche se volatile, non perde le informazioni sino a che non si spegne il sistema.

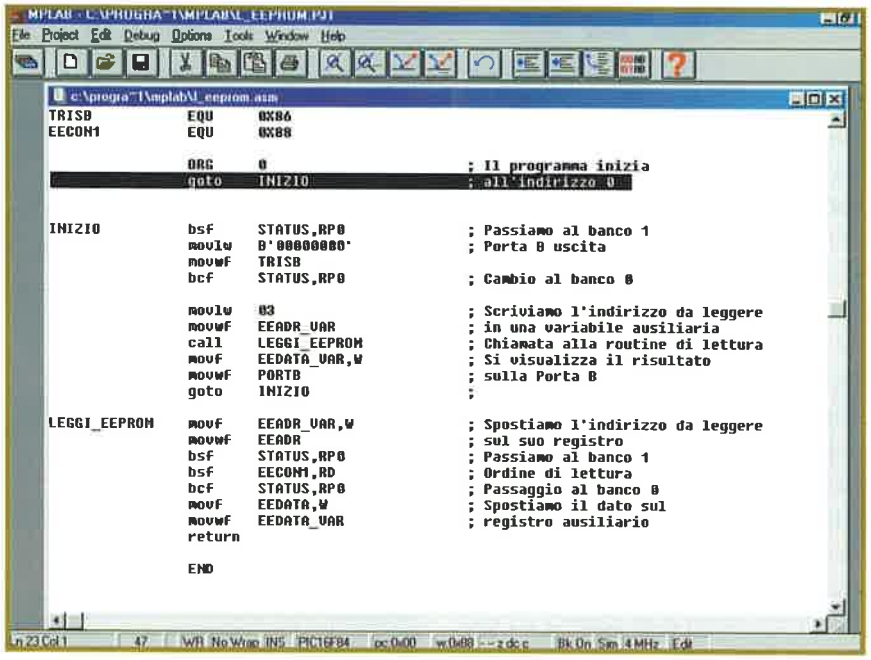
Pensate ad esempio a una sveglia: una volta configurata l'ora corrente e l'ora dell'allarme, essa ci avviserà ogni volta che entrambe le ore coincideranno, e non dovremo più riprogrammarla, a meno che non manchi la corrente. Immaginate ora una carta di credito, i cui dati sono scritti grazie a un'alimentazione esterna nel momento in cui viene introdotta nel lettore; in altre parole dato che non possiede alimentazione propria, quando è nel lettore è collegata, e quando non è inserita è scollegata.

In questo caso se si utilizzasse lo stesso tipo di memoria, la RAM, ogni volta si dovrebbero configurare alcuni dati dell'utente, ad esempio il codice personale, dato che qualsiasi modifica andrebbe persa togliendo la scheda dal lettore. Il tipo di memoria adeguato in questo caso è la EEPROM.

Questa memoria si scrive e si cancella elettricamente, e i suoi dati non vanno persi togliendo l'alimentazione. Ora impareremo a leggere e scrivere in questa memoria, e applicheremo tale metodo in un primo esempio molto semplice.



Passi per la lettura della EEPROM dei dati.



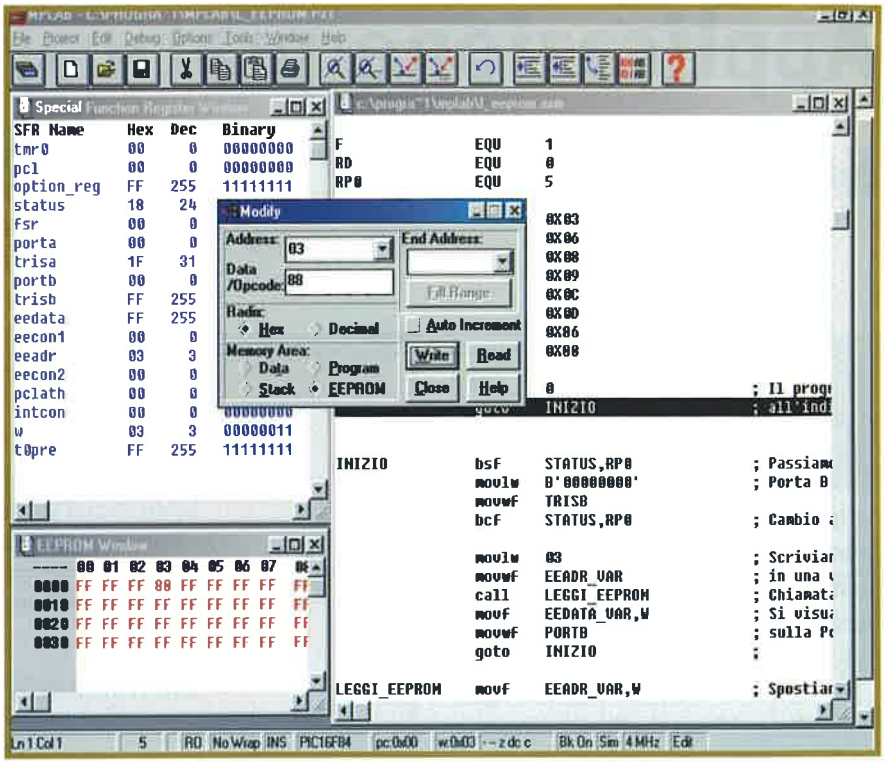
```
TRISB           EQU     0X86
EECON1          EQU     0X88

                ORG     0               ; Il programma inizia
                goto    INIZIO          ; all'indirizzo 0


INIZIO          bsf     STATUS,RP0      ; Passiamo al banco 1
                movlw   B'00000000'     ; Porta B uscita
                movwf   TRISB
                bcf     STATUS,RP0      ; Cambio al banco 0

                movlw   03              ; Scriviamo l'indirizzo da leggere
                movwf   EEADR_VAR       ; in una variabile ausiliaria
                call    LEGGI_EEPROM    ; Chiamata alla routine di lettura
                movf    EEDATA_VAR,W    ; Si visualizza il risultato
                movwf   PORTB           ; sulla Porta B
                goto    INIZIO          ;

LEGGI_EEPROM    movf    EEADR_VAR,W     ; Spostiamo l'indirizzo da leggere
                movwf   EEADR           ; sul suo registro
                bsf     STATUS,RP0      ; Passiamo al banco 1
                bsf     EECON1,RD       ; Ordine di lettura
                bcf     STATUS,RP0      ; Passaggio al banco 0
                movf    EEDATA,W        ; Spostiamo il dato sul
                movwf   EEDATA_VAR      ; registro ausiliario
                return

                END
```

Subroutine e programma per la lettura della EEPROM dei dati.

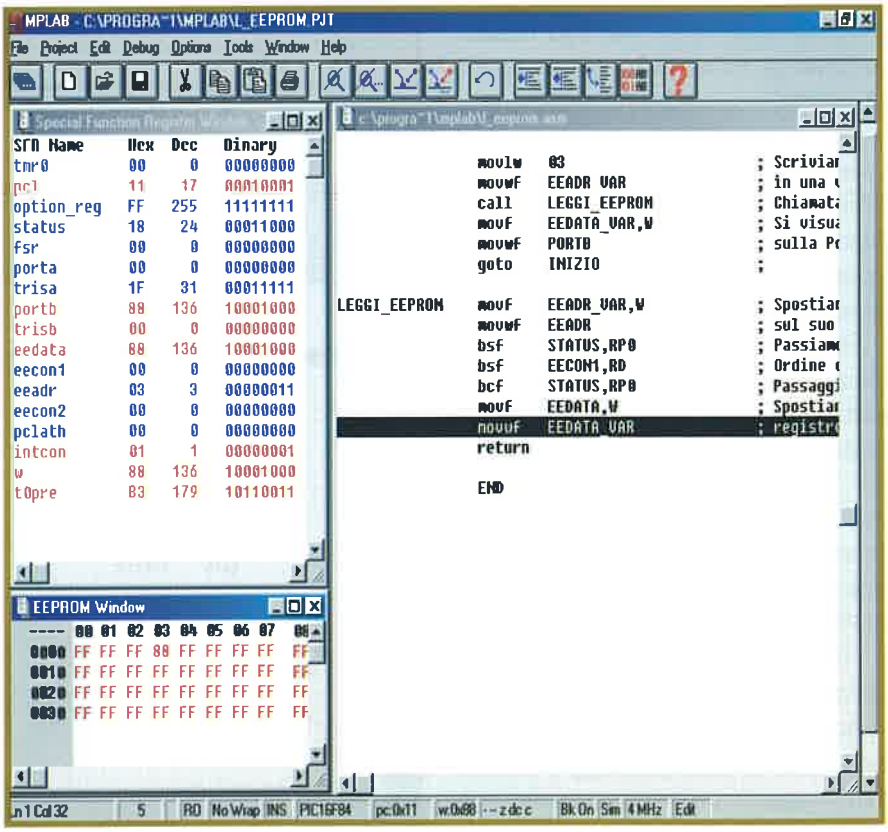
LETTURA DELLA MEMORIA EEPROM DEI DATI

Il processo di lettura della memoria EEPROM dei dati richiede solo di seguire i passi che sono elencati nella figura allegata, facendo uso della RAM dei dati.

Il primo passo è specificare l'indirizzo che si vuole leggere. Dato che la EEPROM dei dati nel PIC16F84 ha 64 byte, questo indirizzo deve essere compreso fra i valori 00h e 3Fh. Questo valore deve essere introdotto nel registro EEADR, ubicato nella posizione 09h del banco 0. L'ordine di lettura si dà ponendo a 1 il bit 0 (bit RD) del registro EECON1, che occupa l'indirizzo 08h del banco 1 nella memoria RAM. A quel punto il dato letto apparirà nel registro EEDATA, posto all'indirizzo 08h del banco 0. Questi passi sono stati scritti in una subroutine chiamata LEGGI_EEPROM, la quale viene utilizzata per leggere un dato a un determinato indirizzo e mostrarlo sulla porta B. Cambiando il valore del registro EEADR_VAR, si realizzerà la lettura di un altro indirizzo. Per fare la simulazione di questo primo programma dovremo prendere, oltre al file dei registri specifici, come abbiamo già fatto altre volte, anche i registri della porta EEPROM dei dati. Inoltre se vogliamo che nella porta B si veda il valore dell'indirizzo della EEPROM, prima dovremo mettere in quell'indirizzo un valore, dato che per default hanno tutti valore FF. Questo si ottiene con l'opzione Window>Modify. Nelle due figure sono riportate queste finestre, prima e dopo la realizzazione della simulazione. Questa simulazione si può fare con l'opzione Debug>Run>Animate, o Debug>Run>Run, infatti ci interessa solo il risultato finale, dato che questo programma è in pratica un ciclo che realizza continuamente la stessa funzione.



Finestre preparate per la simulazione.

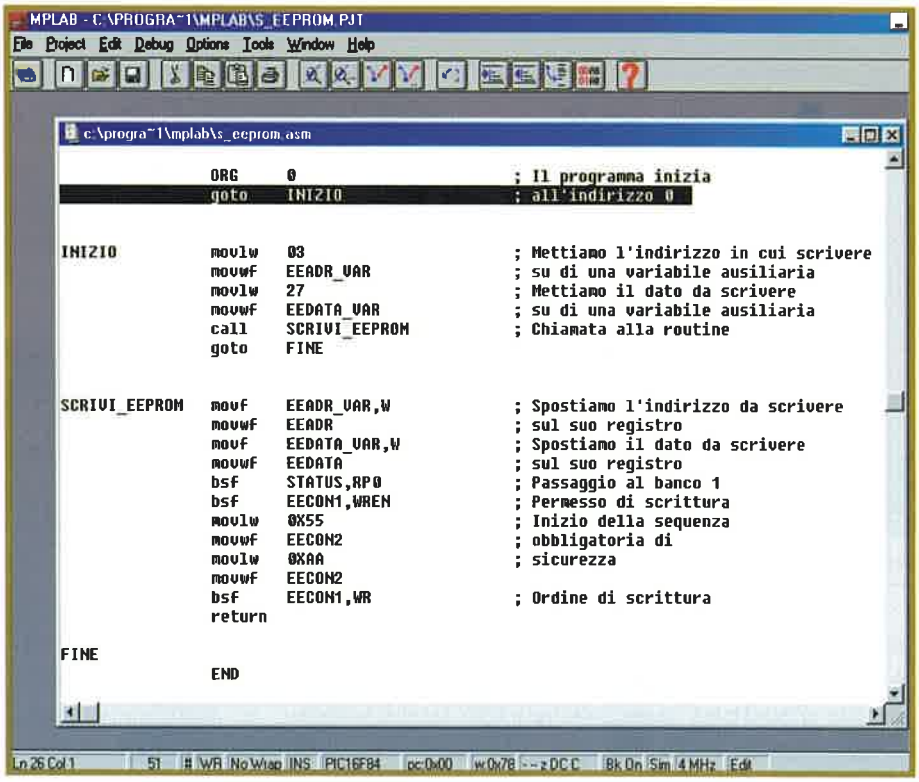


Finestre dopo aver fermato la simulazione.

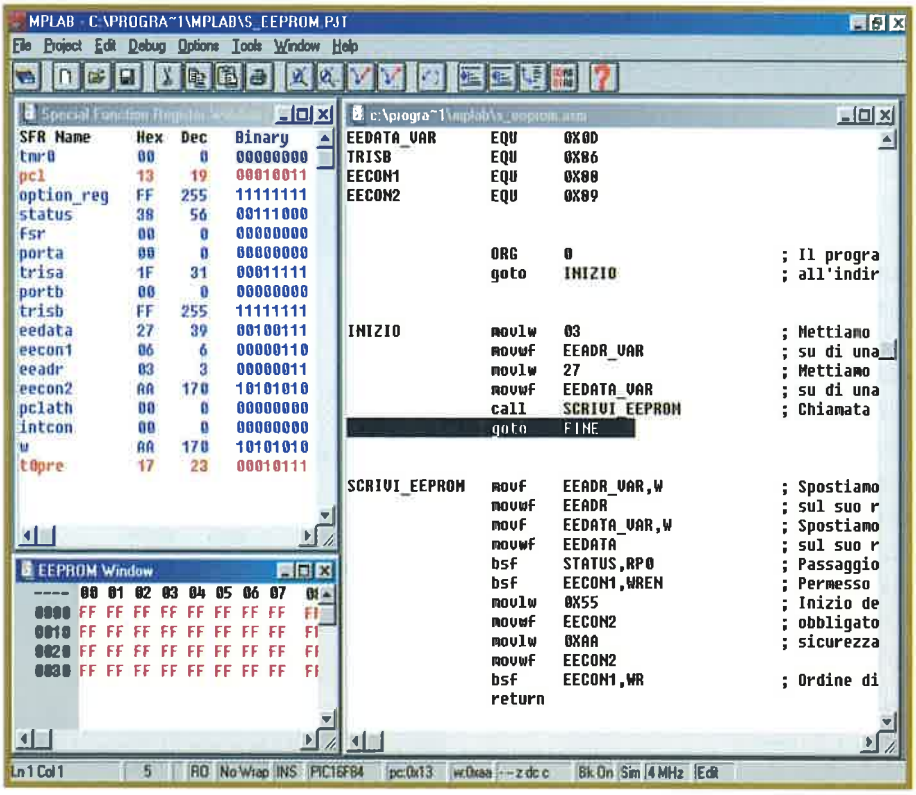
SCRITTURA DELLA MEMORIA EEPROM DEI DATI

Il processo di scrittura della EEPROM è un po' più complesso di quello della lettura. Questo perché mentre la lettura è immediata, la scrittura non lo è. La scrittura impiega un tempo minimo di alcuni millisecondi. Inoltre, affinché non si producano scritture errate, bisogna rispettare alcune procedure di sicurezza predisposte dal costruttore, Microchip. Così dopo aver specificato l'indirizzo di scrittura del dato e abilitata la scrittura, dobbiamo caricare nel registro EECON2 due valori reali: 55h e AAh. Questi registri non sono implementati fisicamente e si utilizzano solo in questo processo di scrittura.

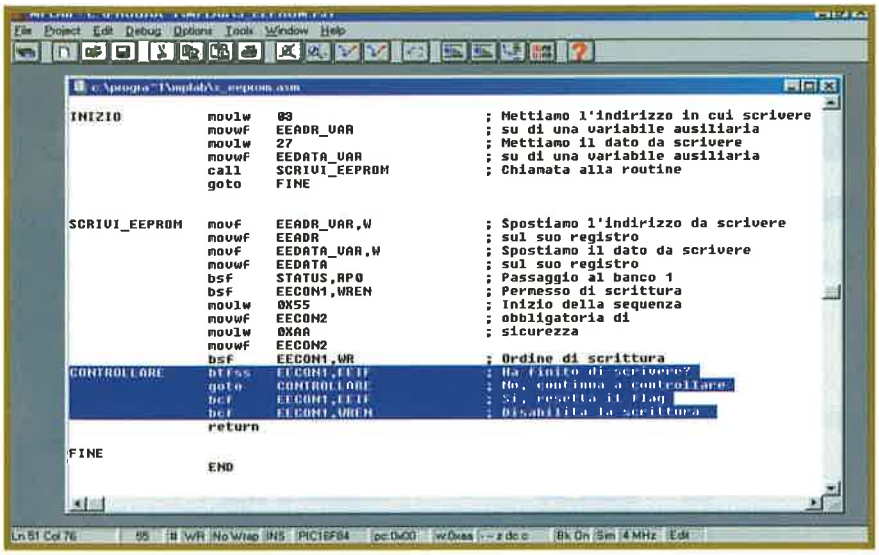
Oltre a questi, si utilizzano gli stessi registri che servono per la lettura; infatti sono stati inclusi in una subroutine chiamata SCRIVERE_EEPROM, che si usa nel programma della figura per scrivere il dato 27 all'indirizzo 03. Cambiando il valore dei registri EEDATA_VAR ed EEADR_VAR si realizzerà la scrittura di altri dati, in altri indirizzi. Per la simulazione di questo secondo programma, utilizzeremo gli stessi registri del caso precedente. Realizzate la simulazione con l'opzione Debug>Run>Step e fermate nell'ultima istruzione il programma, arrivando all'etichetta fine, così come è mostrato nella figura. Come si può osservare non è stato scritto nulla nella EEPROM dei dati. Questa scrittura non si produrrà fino a che non utilizzeremo ancora diverse volte l'opzione Step (F7). Questo è dovuto al



Subroutine e programma per la scrittura della EEPROM dei dati.



Finestre al momento dell'ultima istruzione.



```
INIZIO          movlw   03              ; Mettiamo l'indirizzo in cui scrivere
                movwf   EEADR_VAR       ; su di una variabile ausiliaria
                movlw   27              ; Mettiamo il dato da scrivere
                movwf   EEDATA_VAR      ; su di una variabile ausiliaria
                call    SCRIVI_EEPROM   ; Chiamata alla routine
                goto    FINE

SCRIVI_EEPROM   movf    EEADR_VAR,W     ; Spostiamo l'indirizzo da scrivere
                movwf   EEADR           ; sul suo registro
                movf    EEDATA_VAR,W    ; Spostiamo il dato da scrivere
                movwf   EEDATA          ; sul suo registro
                bsf     STATUS,RP0      ; Passaggio al banco 1
                bsf     EECON1,WREN     ; Permesso di scrittura
                movlw   0X55            ; Inizio della sequenza
                movwf   EECON2          ; obbligatoria di
                movlw   0XAA            ; sicurezza
                movwf   EECON2
                bsf     EECON1,WR       ; Ordine di scrittura
CONTROLLARE     btfss   EECON1,EEIF     ; Ha finito di scrivere?
                goto    CONTROLLARE     ; No, continua a controllare
                bcf     EECON1,EEIF     ; Si, resetta il flag
                bcf     EECON1,WREN     ; Disabilita la scrittura
                return

FINE
                END
```

Subroutine di scrittura della EEPROM dei dati modificata.

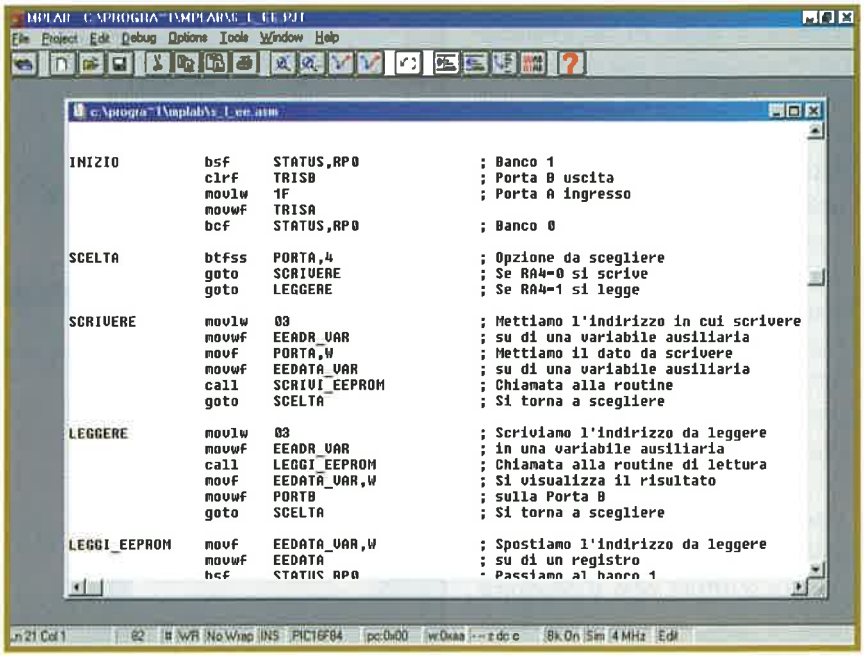
Vogliamo utilizzare la EEPROM per scrivere un codice di un solo digit che può essere letto, visualizzato o cambiato. L'opzione dipenderà dal valore di un segnale d'ingresso, e il valore da modificare si introdurrà tramite quattro piedini del PIC.

Enunciato dell'esercizio proposto.

tempo che impiega la EEPROM per scriversi, per cui, se si uscisse dalla subroutine immediatamente dopo l'ordine di scrittura, questo tempo non sarebbe ancora trascorso. Se dopo aver chiamato questa subroutine si proseguisse con il resto del programma, dopo un certo numero di cicli riusciremmo a realizzare la scrittura, però se subito dopo il ciclo di scrittura si va a leggere,o si pretende di scrivere consecutivamente diversi dati in differenti indirizzi, il programma non funzionerà, dato che non ha avuto tempo sufficiente per realizzare la scrittura. Dovremo quindi modificare la subroutine di scrittura. La modifica consiste nell'attendere che la scrittura del dato sia terminata, cioè verificando quando viene messo di nuovo a 0 il bit WR o quando il bit EEIF (bit 4 del registro EECON1) viene messo a 1. Dopo dovremo resettare via software questo flag. Una buona tecnica di programmazione consiste nel verificare che il dato scritto sia veramente il dato che volevamo scrivere, leggendolo e comparandolo con l'originale. Il bit 3 del registro WERR verrà messo a 1 in caso di errore.

LETTURA E SCRITTURA CONGIUNTE

Di seguito mostreremo una piccola applicazione dove si utilizza sia la scrittura che la lettura della EEPROM. Leggete l'enunciato che vi proponiamo. Utilizzeremo i quattro bit meno significativi della porta A per introdurre i valori da scrivere nella EEPROM e la porta B per mostrare questo valore. La quinta linea della porta A sarà quella che determina se si legge la EEPROM o se si scrive in essa un nuovo valore, a seconda che abbia valore 1 o 0 rispettivamente. Seguite le istruzioni del programma principale. Le chiamate alle subroutines le abbiamo già viste e commentate.



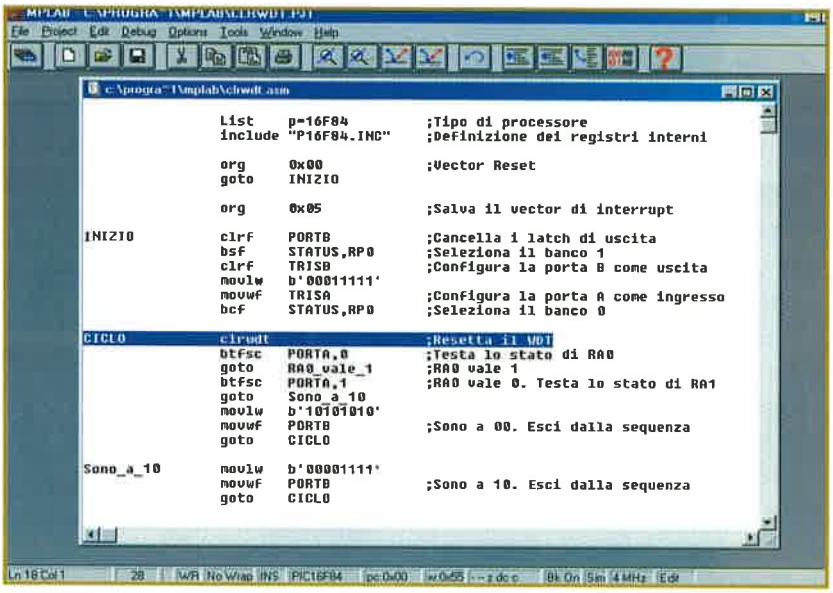
```
INIZIO          bsf     STATUS,RP0      ; Banco 1
                clrf    TRISB           ; Porta B uscita
                movlw   1F              ; Porta A ingresso
                movwf   TRISA
                bcf     STATUS,RP0      ; Banco 0

SCELTA          btfss   PORTA,4         ; Opzione da scegliere
                goto    SCRIVERE        ; Se RA4=0 si scrive
                goto    LEGGERE         ; Se RA4=1 si legge

SCRIVERE        movlw   03              ; Mettiamo l'indirizzo in cui scrivere
                movwf   EEADR_VAR       ; su di una variabile ausiliaria
                movf    PORTA,W         ; Mettiamo il dato da scrivere
                movwf   EEDATA_VAR      ; su di una variabile ausiliaria
                call    SCRIVI_EEPROM   ; Chiamata alla routine
                goto    SCELTA          ; Si torna a scegliere

LEGGERE         movlw   03              ; Scriviamo l'indirizzo da leggere
                movwf   EEADR_VAR       ; in una variabile ausiliaria
                call    LEGGI_EEPROM    ; Chiamata alla routine di lettura
                movf    EEDATA_VAR,W    ; Si visualizza il risultato
                movwf   PORTB           ; sulla Porta B
                goto    SCELTA          ; Si torna a scegliere

LEGGI_EEPROM    movf    EEDATA_VAR,W    ; Spostiamo l'indirizzo da leggere
                movwf   EEDATA          ; su di un registro
                bsf     STATUS,RP0      ; Passiamo al banco 1
```

Terminato il nostro programma, si prosegue eseguendo posizioni non valide.

Scheda delle istruzioni: l'istruzione CLRWDT

L'istruzione clrwdt (clear watchdog) cancella il temporizzatore chiamato Watchdog, evitando in questo modo che raggiunga il suo valore massimo (FF) e vada in overflow, cosa che farebbe resettare il sistema. Non ha nessun parametro, quindi il suo uso è semplice come scrivere il suo nome.

Nello stesso momento in cui viene messo a zero il Watchdog (WDT) si mette a zero anche il suo prescaler, o divisore di frequenza, che è una risorsa utilizzata dal Watchdog per ritardare ulteriormente il tempo di overflow, come avevamo già visto per il TMR0. I bit #TO e #PD, situati nel registro di STATO, prenderanno valore 1. Verificando questi bit all'inizio del programma; si potrà sapere quale sia stata la causa dell'ultimo reset.



Esempio di un programma che controlla il Watchdog.

ESEMPI CON L'ISTRUZIONE CLRWDT

Il Watchdog è una risorsa propria dei PIC per vigilare che le istruzioni di un programma siano eseguite in modo adeguato. Funziona con un clock interno di tipo RC, e si incrementa sino ad arrivare al valore FF, per poi andare in overflow e resettare il sistema. Questo non succede se fra le istruzioni del programma se ne trova qualcuna in grado di cancellare il Watchdog: la clrwdt, con cui il contatore ripartirà da zero. Può succedere che a causa del rumore esterno al sistema, il Program Counter vada a puntare su un indirizzo di memoria che non contiene nessuna istruzione. In questo caso il programma smette di funzionare, però il Watchdog continuerà ad incrementarsi sino ad arrivare a FF e debordare, resettando il sistema, e assolvendo così alla sua funzione. La figura mostra un esempio nel quale si controlla l'uso del Watchdog. Quando il programma è eseguito in modo corretto, il WDT viene messo a zero all'esecuzione dell'apposita istruzione, altrimenti si produrrà un reset.

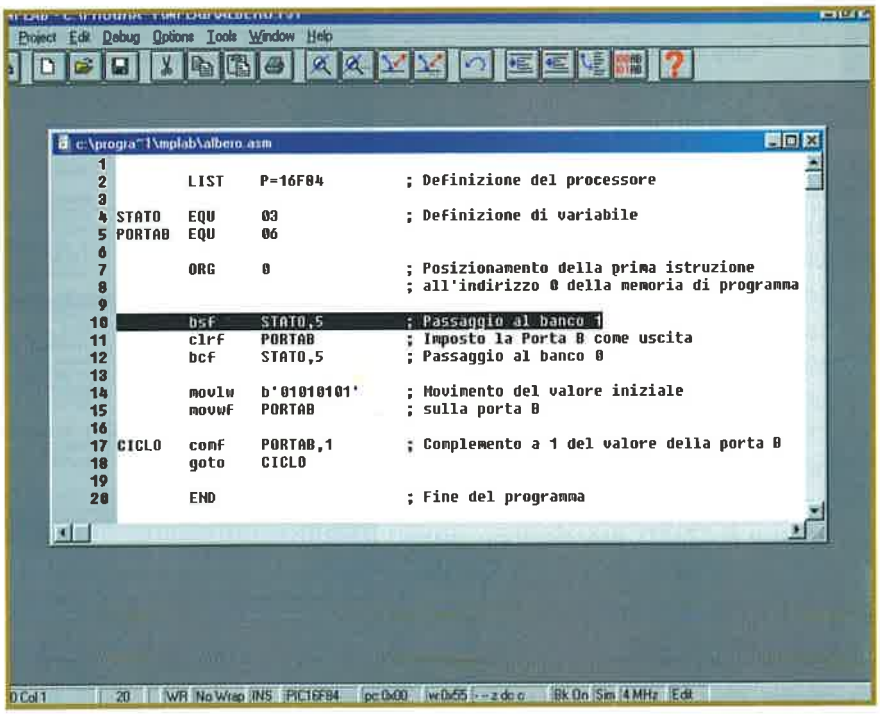
MNEMONICO	PARAMETRO 1	PARAMETRO 2
c l r w d t		

Operazione: cancellazione del Watchdog **(WDT).**
Cicli: 1
Codice OP: 00 0000 0110 0100
Flag: #TO, #PD

Caratteristiche dell'istruzione clrwdt.

Scheda delle istruzioni: l'istruzione COMF

Con questa istruzione si realizza il complemento a 1 del valore di un registro. Questo si fa bit a bit, cioè ogni bit di valore zero del registro viene cambiato con un uno e ogni bit che vale uno si cambia con uno zero. Il risultato di questa operazione si può memorizzare nello stesso registro da dove si prende il valore iniziale se il parametro d ha valore 1, o nel registro di lavoro W se il parametro d ha valore 0. Inoltre nel caso che l'operazione dia come risultato 0, il flag Z verrà messo a 1. Questo significa che il registro, prima dell'operazione, aveva valore FF.



Programma di simulazione delle luci di Natale.

ESEMPI CON L'ISTRUZIONE COMF

L'uso di questa istruzione è semplice, ma complicata potrebbe essere l'applicazione dove la si vuole utilizzare. Per il momento vogliamo limitarci ad usarla in un programma che la vede protagonista. Immaginiamo per esempio le luci di un albero di Natale. Possono seguire differenti sequenze, però la maggior parte delle volte si accendono e si spengono formando disegni in cui ognuna di esse si complementa con l'altra. Digitate sul PC il programma della figura e guardate nella porta B mentre viene eseguito con l'opzione "animate". Volete provare ad inserire un interruttore che, a seconda del suo valore, faccia eseguire una sequenza oppure un'altra inventata da voi? E perché non metterlo in pratica per il prossimo Natale?

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
c o m f	f	d

Operazione: complemento a 1 del contenuto del registro **f** bit a bit. Se **d=0** il risultato si lascia nel registro di lavoro **W**, se **d=1** si lascia nello stesso registro **f.**

Cicli: 1

Codice OP: 00 1001 dfff ffff

Flag: Z

Caratteristiche dell'istruzione comf.

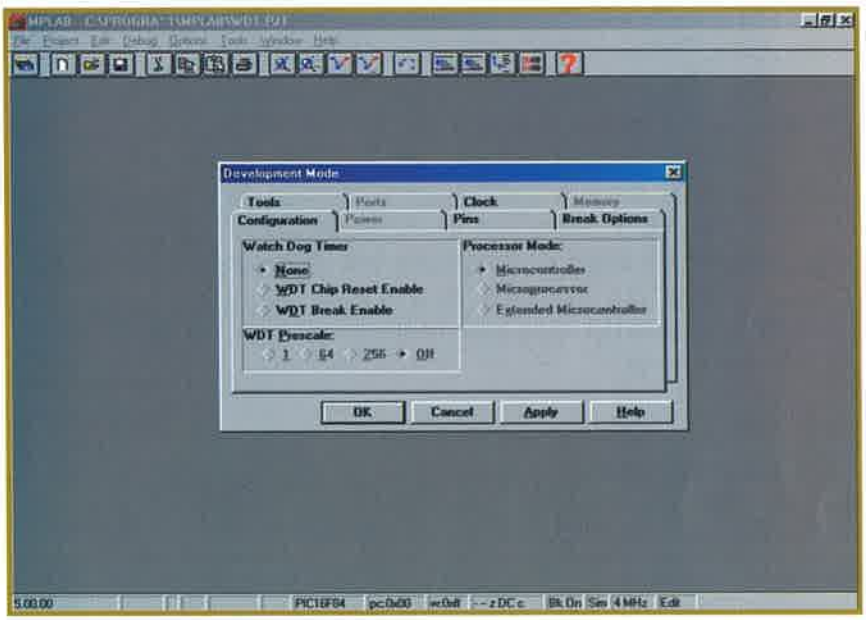
Applicazione pratica: attenzione al Watchdog!

Se ci venisse spiegato che il Watchdog è un temporizzatore a 8 bit che causa un Reset quando va in overflow, senza aggiungere altro, ci potrebbe sembrare una risorsa inutile. Senza dubbio non è così, tuttavia il suo uso risulta scomodo per alcuni utenti, che decidono di disattivarlo.

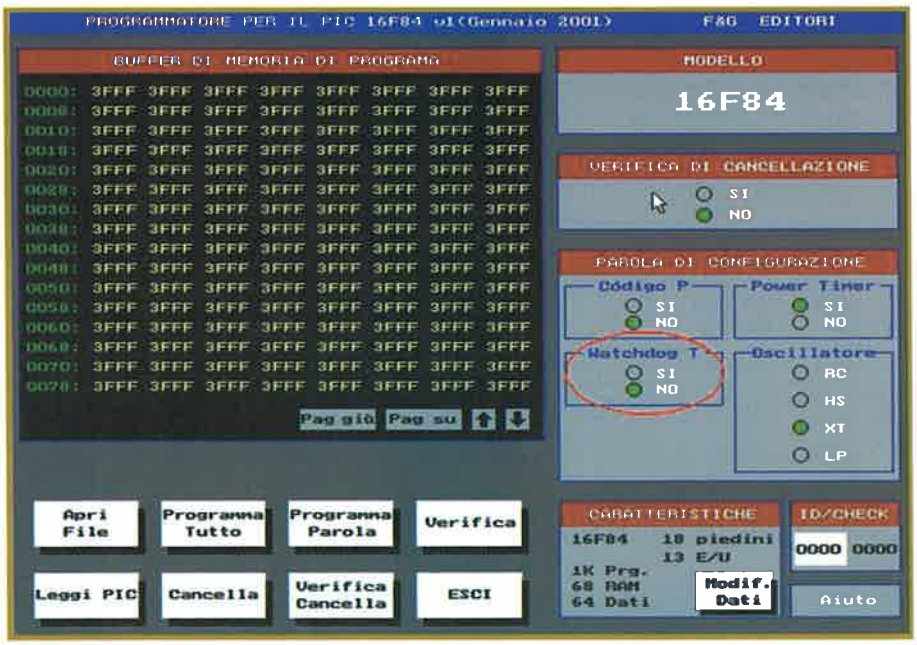
DISATTIVAZIONE DEL WATCHDOG

Il primo passo, parlando di Watchdog, sarà decidere se utilizzarlo o meno. Nel caso la risposta sia negativa, al momento di scrivere un programma sul microcontroller dovremo indicare al software di scrittura questa scelta.

La zona cerchiata in rosso è l'opzione che dovremo prendere in considerazione in questo caso: impostando NO il Watchdog rimarrà disabilitato, al contrario, impostando SI lo abiliteremo. Se in seguito desiderassimo utilizzarlo, dovremo riprogrammare il microcontroller con questa opzione cambiata.



Finestra di configurazione del Watchdog in MPLAB.



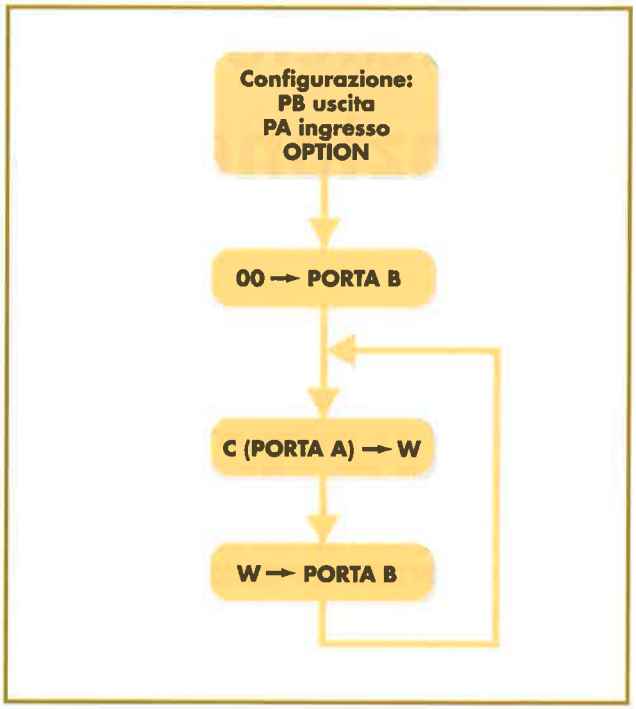
La disattivazione del Watchdog si fa tramite il software di scrittura.

Se invece di lavorare direttamente con il microcontroller utilizzeremo il simulatore, potremo attivare o disattivare ugualmente il Watchdog, lo faremo tramite Options>Development Mode, e una volta qui, sceglieremo la scheda che si chiama Configuration per cambiare le caratteristiche del Watchdog.

All'interno di questa scheda, se l'opzione scelta sarà "None" il Watchdog rimarrà disabilitato.

Si vuole visualizzare sulla porta B il valore complementato introdotto sulla porta A, in modo ciclico. Inizialmente la porta A avrà valore 0. Si controllerà l'uso del Watchdog.

Questa scelta equivale a mettere NO nella videata del software di scrittura. Per l'opzione SI abbiamo due possibilità: una equivale a quella del software di scrittura, vale a dire abilitare il Reset del PIC all'overflow del Watchdog (WDT Chip Reset Enable), e l'altra consiste nella possibilità di fermare il sistema quando si produce l'overflow (WDT Break Enable). L'opzione del Prescale del Watchdog (WDT Prescale) durante la memorizzazione, non serve quando si lavora con un PIC16F84, quindi appare disabilitata anche se si seleziona una delle due opzioni relative. La selezione del Prescale per il Watchdog, in modo che ritardi il momento dell'overflow, va fatta via software.

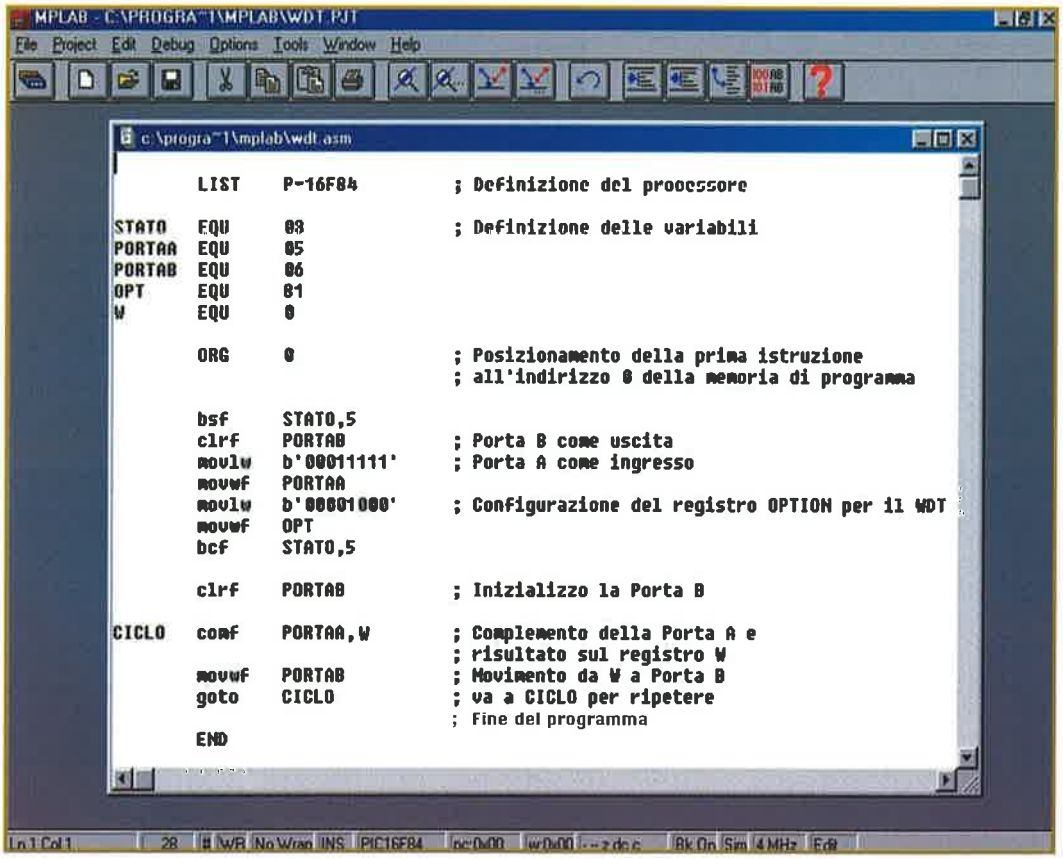


Organigramma dell'enunciato proposto.

IL LAVORO CON IL WATCHDOG

Se avete scelto di lavorare con il Watchdog, nelle prossime righe imparerete come fare. Scriveremo un primo programma d'esempio, molto semplice, dove si vede chiaramente come funziona il Watchdog. Come dichiarato nell'enunciato, realizzeremo un'azione in modo ripetitivo, ovvero un ciclo infinito, dopo l'inizializzazione delle variabili. L'azione da compiere consiste nel prendere il dato che si trova sulla porta A, che potrà essere variato dall'utente esterno, complementarlo, e mostrarlo sulla porta B.

Come sempre, per prima cosa fa-




```
        LIST    P-16F84         ; Definizione del processore

STATO   EQU     03              ; Definizione delle variabili
PORTAA  EQU     05
PORTAB  EQU     06
OPT     EQU     81
W       EQU     0

        ORG     0               ; Posizionamento della prima istruzione
                                ; all'indirizzo 0 della memoria di programma

        bsf     STATO,5
        clrf    PORTAB          ; Porta B come uscita
        movlw   b'00011111'     ; Porta A come ingresso
        movwf   PORTAA
        movlw   b'00001000'     ; Configurazione del registro OPTION per il WDT
        movwf   OPT
        bcf     STATO,5

        clrf    PORTAB          ; Inizializzo la Porta B

CICLO   comf    PORTAA,W        ; Complemento della Porta A e
                                ; risultato sul registro W
        movwf   PORTAB          ; Movimento da W a Porta B
        goto    CICLO           ; va a CICLO per ripetere
                                ; Fine del programma
        END
```

Organigramma tradotto in programma.

remo l'organigramma dell'enunciato proposto.

Come possiamo osservare si prevede l'uso del Watchdog, anche se non del suo controllo, dato che l'unico riferimento fatto al Watchdog è la sua configurazione nel registro OPTION. Inoltre possiamo vedere che il nostro organigramma si avvicina molto a quello che saranno le istruzioni del programma, dato che i tre riquadri avrebbero potuto contenere affermazioni del tipo: "inizializzare porta B a zero", "Complementare porta A" e "Passare complemento della porta A sulla porta B".

Nel listato del programma si possono vedere chiaramente ognuna delle parti riflesse in precedenza nell'organigramma. Nella configurazione del registro OPTION, gli unici valori che sono significativi sono i bit da 0 a 2, che pongono il prescale a 0, e il bit 3 che assegna questo predivisore al Watchdog. In conseguenza di questo, il Watchdog impiegherà circa 18 ms ad andare in overflow e a resettare il sistema. Dopo aver editato e assemblato il programma, faremo una prima simulazione, con il Watchdog disattivato (Options>Development Mode>Configuration>None) per vedere meglio la differenza.

SIMULAZIONE DEL PROGRAMMA

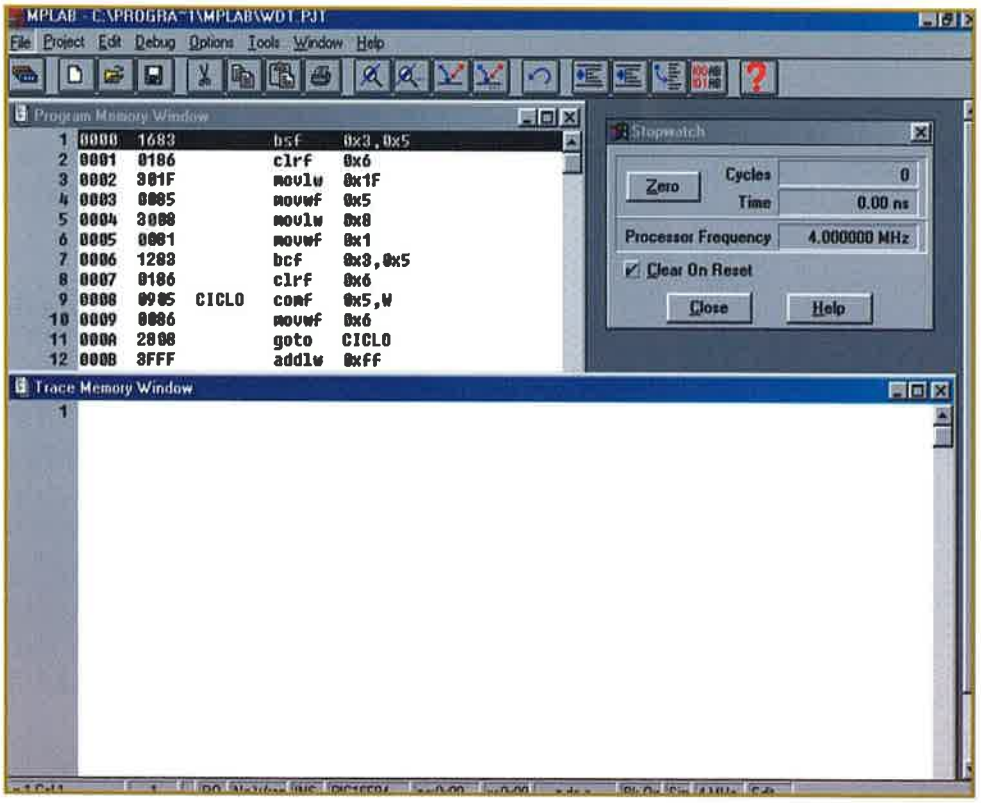
Utilizzare l'opzione Debug>Run>Animate per simularlo e, senza preoccuparci dei valori che assumeranno i registri, osservare come vengono eseguite le istruzioni: dopo che entra nel ciclo, il programma continua a girare lì all'infinito.

Configureremo ora il simulatore per lavorare con il Watchdog. Per il momento simuleremo solo la scelta che si può fare tramite il software di scrittura, cioè cliccheremo l'opzione Options>Development Mode>Configuration> WDT Chip Reset Enable. In questo modo il programma si resetterà quando andrà in overflow il Watchdog.

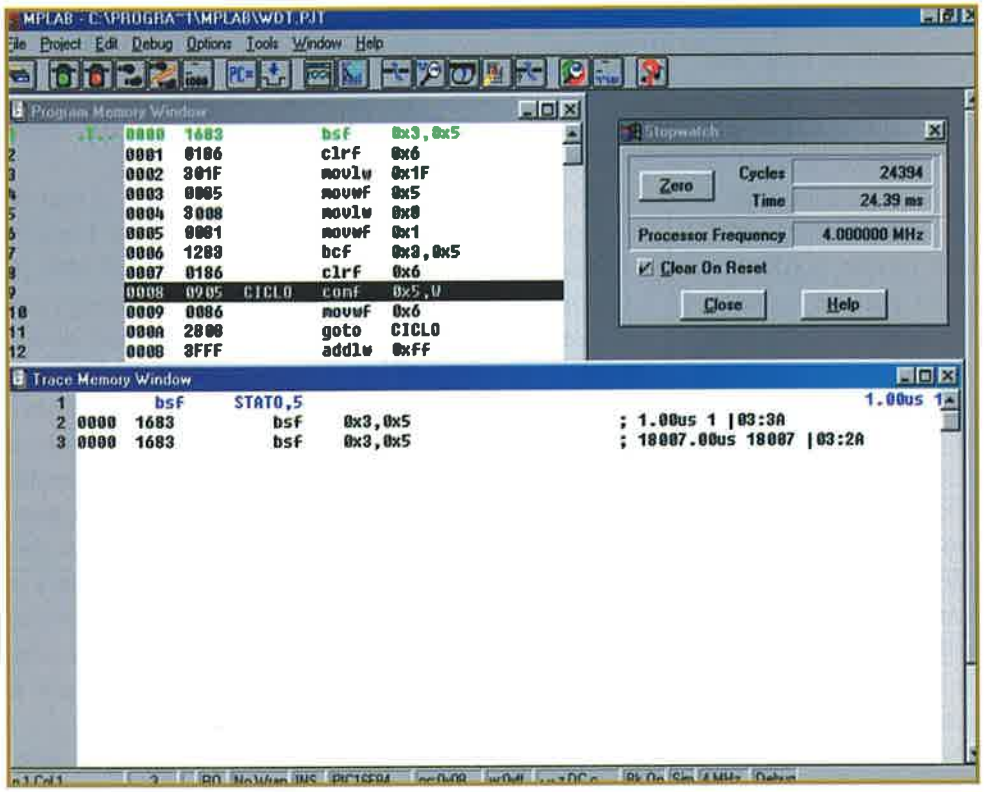
Dato che stiamo eseguendo una simulazione, se sceglieremo le l'opzioni Animate, potrebbe passare diverso tempo prima che si verifichi l'overflow , quindi apriremo alcune finestre per farlo in un altro modo, senza perdere dettagli.

Da un lato attiveremo una finestra che ci indichi in ogni momento il tempo trascorso: è la finestra Window>Stopwatch. Ad una frequenza di 4 MHz, il tempo che deve passare prima che il Watchdog vada in overflow sarà di 18 ms, però saremo capaci di fermare la simulazione in quel momento, per vedere in che modo il programma inizia nuovamente? Sicuramente no. Per questo utilizzeremo anche la finestra per tracciare il programma; in altre parole, questa finestra ci permette di vedere quando il programma passa per l'istruzione che ci interessa. Per fare questo utilizzeremo Window>Trace Memory e Window>Program Memory. Nella finestra di memoria del programma dovremo selezionare il codice che vogliamo mettere sotto controllo, ad esempio la prima istruzione del programma "bsf STATO,5".

Cliccheremo poi con il pulsante destro del muose, per fare apparire un menù contestuale. All'interno di questo menù sceglieremo la funzione di tracciatura



Finestre da utilizzare per la simulazione.



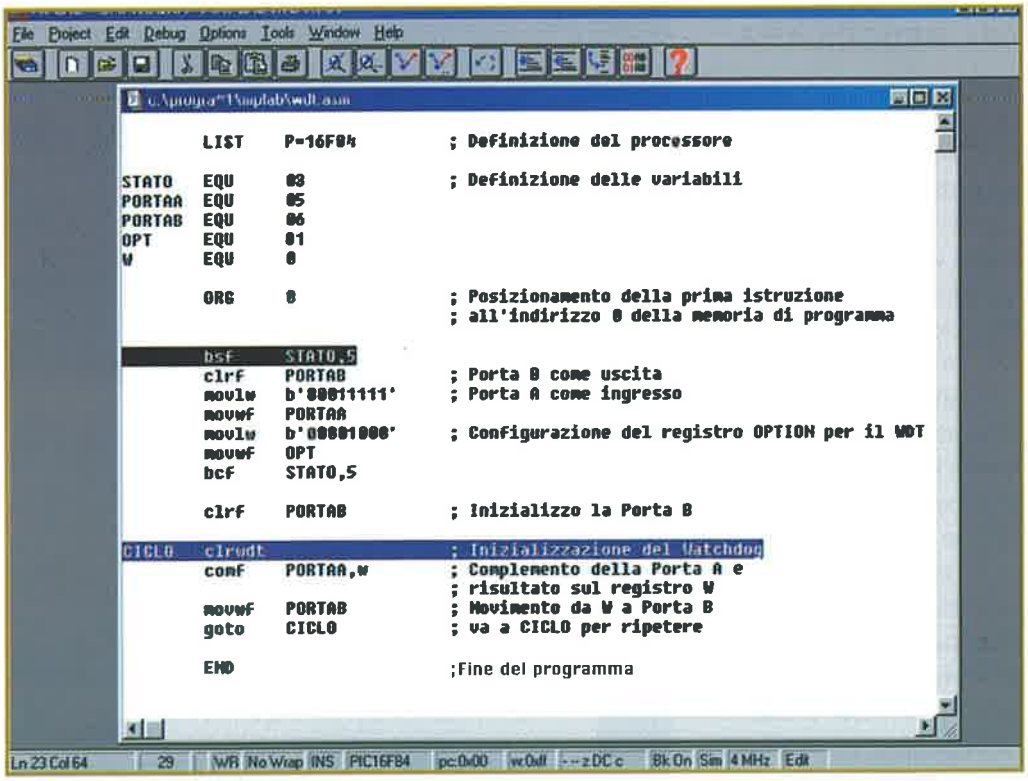
Stato delle finestre dopo aver fermato la simulazione.

CONTROLLO DEL WATCHDOG

Ora controlleremo il Watchdog, in modo che non provochi questi reset del programma. Per fare questo utilizzeremo l'istruzione "clrwdt": essa cancella il Watchdog ogni volta che viene eseguita, evitando che arrivi al limite e vada in overflow, a meno che il programma sfugga dal controllo, in questo caso il WDT resetterà davvero il sistema. Questa istruzione di solito si mette all'inizio dei cicli, e in sequenze di programma molto lunghe. Provate a inserire questa istruzione come nella figura, e ripetete la simulazione: ha funzionato il controllo sul Watchdog?

(Trace Point(s)); ora si può eseguire il programma

Quando la finestra Stopwatch segnerà almeno 18 ms, si potrà fermare la simulazione. L'opzione più rapida sarà Debug>Run>Run. Nella finestra Trace Memory apparirà nella parte sinistra l'istruzione messa sotto controllo, e nella parte destra il tempo trascorso. Il primo tempo è quello trascorso dall'inizio del programma, dato che si tratta della prima istruzione, ogni apparizione successiva invece indicherà il susseguirsi degli overflow del Watchdog.



Inserimento dell'istruzione per il controllo del Watchdog.

Scheda delle istruzioni: l'istruzione SLEEP

Questa istruzione pone il microcontroller in stato di riposo o di basso consumo. È sufficiente scrivere il mnemonico dell'istruzione, dato che si utilizza senza nessun parametro, per fermare il sistema. Il flag #PD (Power Down) si pone a 0 e il flag #TO (Timer Out) si pone a 1.

MNEMONICO	PARAMETRO 1	PARAMETRO 2
sleep		

Operazione: Porta il microcontroller nello stato di riposo
Cicli: 1
Codice OP: 00 0000 0110 0011
Flags: #TO, #PD

Caratteristiche dell'istruzione sleep.

Questo tipo di funzionamento è consigliato per applicazioni in cui ci siano lunghi periodi di inattività, nei quali il microcontroller debba comunque sorvegliare il valore degli ingressi.

ESEMPI CON L'ISTRUZIONE SLEEP

Quando si esegue l'istruzione, l'oscillatore del sistema si ferma.

Dato che il ritmo, o velocità delle istruzioni, è scandito dall'oscillatore, e questo è fermo, le istruzioni si fermano anch'esse, cioè non sono più eseguite.

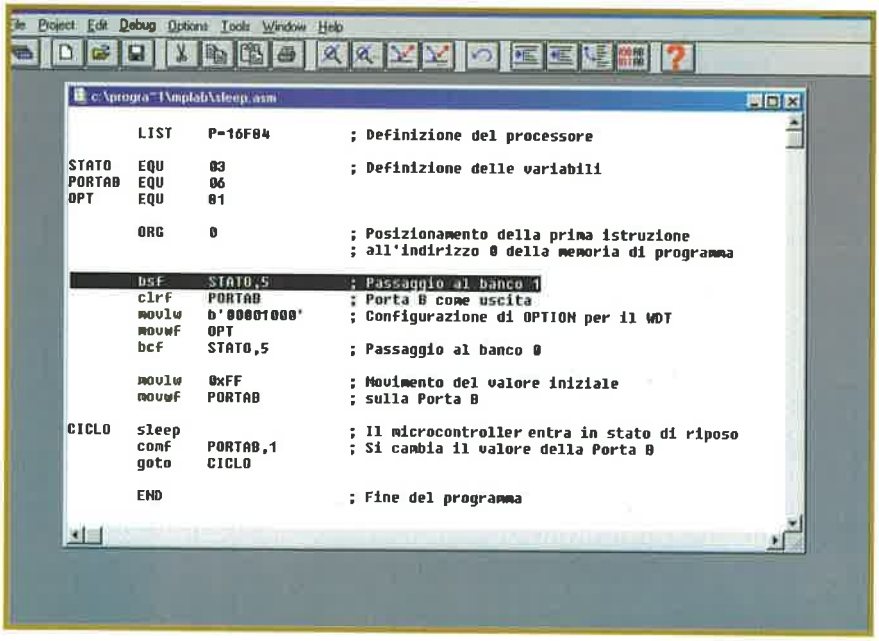
Dato che le uscite del sistema variano con le istruzioni, e queste non vengono eseguite, le linee configurate come uscite non cambieranno più valore; se erano a 1 resteranno a 1 e se erano a 0 anche. Nemmeno il TMR0 funziona, quindi non arriva all'overflow.

Questo è lo stato tipico di un telecomando; rimane in stato di riposo sino a che qualcuno decide di utilizzarlo e preme un pulsante.

Questo pulsante provoca un interrupt che farà uscire dal letargo il processore. Inoltre si può "svegliare" il processore provocando un reset tramite il piedino MCLR#, oppure mediante il Watchdog.

In quest'ultimo caso, entrando in sleep, il Watchdog si azzera, però dato che funziona con un oscillatore indipendente, può continuare a incrementarsi e arrivare all'overflow, provocando un reset. Si tratta comunque di un reset speciale, dato che non si riparte dall'inizio del programma, ma dall'istruzione successiva allo sleep.

Copiate il programma della figura e simulatelo con il WDT attivato e poi senza attivarlo. Attendete almeno 18 ms per vedere cosa succede.



Programma che permane in stato di riposo.

Scheda delle istruzioni: l'istruzione SWAPF

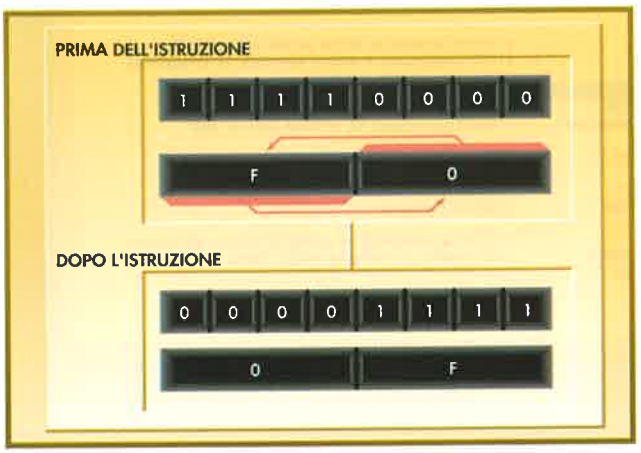
Con questa istruzione si cambiano i quattro bit più significativi di un registro con i quattro bit meno significativi dello stesso registro. Il risultato si memorizza nello stesso registro o nel registro di lavoro W, come sempre a seconda del valore del parametro d. Nella figura è rappresentato il modo in cui è eseguito questo cambio. Dato che il registro si divide in due si cambierà una parte con l'altra.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
s w a p f	f	d

Operazione: scambio dei quattro bit più significativi del registro **f** con i quattro meno significativi dello stesso registro. Se **d=0** il risultato resta nel registro di lavoro **W** e se **d=1** il risultato si lascia nello stesso registro **f**

Cicli: 1
Codice OP: 00 1110 dfff ffff
Flag: nessuno

Caratteristiche dell'istruzione swapf.



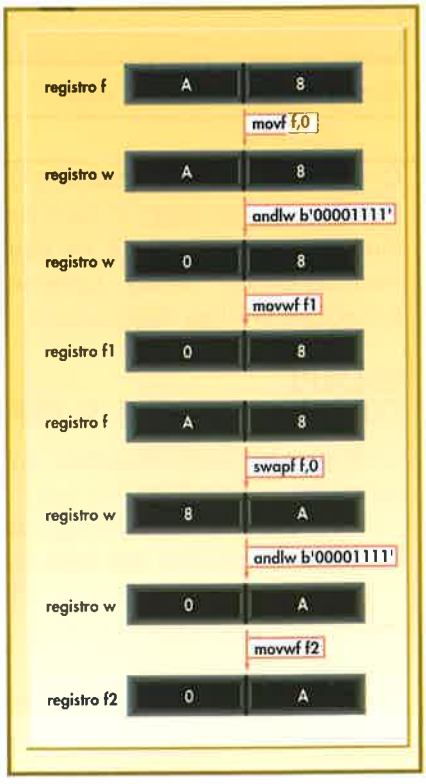
Rappresentazione del funzionamento dell'istruzione swapf.

ESEMPI CON L'ISTRUZIONE SWAPF

Questa istruzione è più importante di quanto sembri; come avrete già notato, i dati normalmente sono raggruppati in byte, cioè in gruppi da 8 bit, per poi rappresentare questi bit in binario o in esadecimale. Così 8 bit si dividono in due digit esadecimali, o se vogliamo rappresentare il dato in decimale, in due digit decimali. Quindi in un solo registro possiamo scrivere due digit, e data la scarsità dei registri di cui dispone il PIC16F84, questo è un grande vantaggio. Però come possiamo fare, se, ad esempio, vogliamo sommare i due numeri che sono scritti nello stesso registro? Ecco che entra in gioco la nostra istruzione. L'idea è schematizzata nella figura; dobbiamo annullare i bit più significativi e memorizzare il dato su un altro registro; si ottiene così il primo valore.

Si utilizza in seguito l'istruzione swapf per scambiare i bit più significativi con quelli meno significativi, e si torna ad annullare i bit più significativi (quelli che erano i meno significativi), e si memorizza il risultato su un nuovo registro.

Dopo aver ottenuto i due dati si può eseguire la somma.



Schema esplicativo di un esempio che utilizza l'istruzione swapf.

Applicazione pratica: l'interrupt arriva dall'esterno

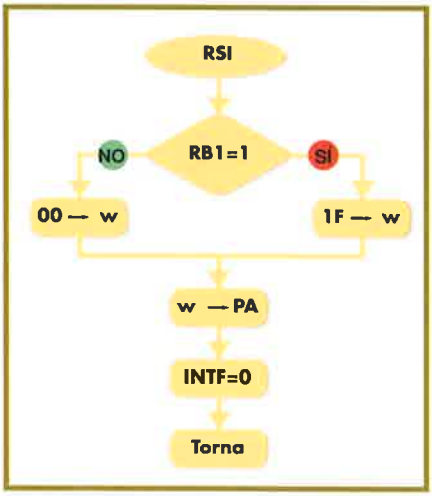
Quando si impara a programmare bene con i microcontroller, è normale utilizzare una struttura in cui il programma principale compia solo l'inizializzazione delle risorse, mentre nella routine di servizio all'interrupt (RSI) si realizzi solo ciò che si desidera dal programma. Il difficile non è utilizzare gli interrupt, ma sapere cosa fare quando sono generati. Però è meglio procedere a poco a poco, per il momento è sufficiente vedere un primo esempio di routine applicato ad un programma.

INTERRUPT DA RB0/INT

L'utilizzo dell'interrupt evita di dover testare continuamente se si è verificato un determinato evento. Così il microcontroller può restare in stato di riposo e risvegliarsi in caso si produca un interrupt, ad esempio uno esterno tramite RB0/INT.

Se si analizza l'enunciato proposto si arriva alla conclusione che nel programma principale non si fa niente; tutto parte dall'interrupt. Nella subroutine dedicata, dopo aver verificato il valore di un bit si esegue qualsiasi operazione.

L'organigramma rappresenta le operazioni che si faranno nella Routine di Servizio all'Interrupt. L'organigramma principale non è stato disegnato, perché è solamente di configurazione. Prima di trasformare l'organigramma in programma vediamo il significato dei bit di due registri molto importanti in questo caso: INTCON e OPTION.



Organigramma della routine di interrupt.

Nel registro INTCON si eseguono le abilitazioni ai diversi interrupt. Noi lavoreremo solo con RB0/INT, quindi dovremo proibire tutti gli altri. Il bit 4 è quello che dovremo mettere a 1 in questo caso, inoltre il bit 7 è l'abilitazione generale degli interrupt, e se non è a 1 non sarà possibile utilizzare nessun tipo di interrupt. Per ogni tipo di interrupt quindi, deve essere messo a 1 il bit che gli corrisponde, più il bit di abilitazione generale. Dopo aver prodotto l'interrupt, si attiverà un flag, per RB0/INT sarà il bit 1 di questo stesso registro. Per quanto riguarda il registro OPTION solo un bit è significativo per il nostro esempio: il 6. Con esso specifichiamo se il fronte attivo sarà quello di salita o di discesa, a seconda che il suo valore sia 1 o 0 rispettivamente.

**Abbiamo un dispositivo che permane in stato di riposo sino a che un utente non lo risveglia premendo RB0/INT.
Al risveglio, a seconda del valore di RB1, si pone la porta A a 0 oppure a 1.**

Enunciato per l'uso di un interrupt.

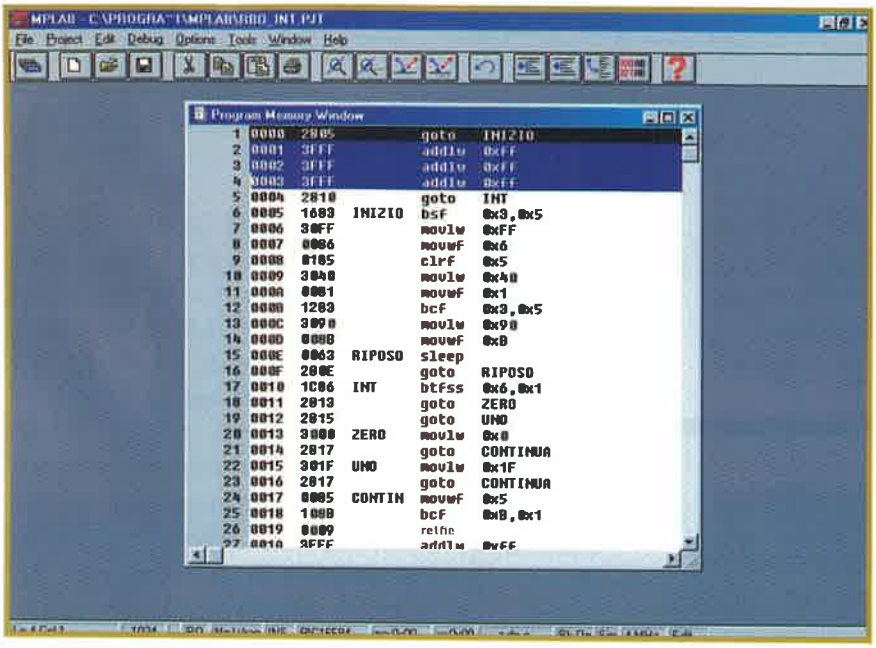
INTCON	1	0	0	1	0	0	0	0
OPTION	X	1	X	X	X	X	X	X

Registri coinvolti nell'interrupt RB0/INT.

IL PROGRAMMA COMPLETO

```
                LIST        P=16F84         ; Definizione del processore

STATO           EQU         03              ; Definizione di variabili
PORTAA          EQU         05
PORTAB          EQU         06
INTCON          EQU         0B
OPT             EQU         81
INTF            EQU         1

                ORG         0
                goto        INIZIO
                ORG         4               ; Indirizzo del Vector di interrupt
                goto        INT
                ORG         5

                                            ; Programma principale

INIZIO          bsf         STATO,5
                movlw       0xFF            ; Porta B come ingresso
                movwf       PORTAB
                clrf        PORTAA          ; Porta A come uscita
                movlw       b'01000000'     ; Configurazione del fronte attivo per l'interrupt
                movwf       OPT
                bcf         STATO,5

                movlw       b'10010000'     ; Attivazione dell'interrupt
                movwf       INTCON          ; tramite RB0/INT

RIPOSO          sleep                       ; Si pone in stato di riposo
                goto        RIPOSO          ; Ritornando dalla subroutine di
                                            ; interrupt si passa in stato di riposo
                                            ; sino al prossimo interrupt

INT             btfss       PORTAB,1        ; Verifica il valore del bit RB1
                goto        ZERO            ; È zero
                goto        UNO             ; È uno

ZERO            movlw       00              ; Si sposta lo zero
                goto        CONTINUA

UNO             movlw       b'00011111'     ; Si sposta uno
                goto        CONTINUA

CONTINUA        movwf       PORTAA          ; sulla Porta A
                bcf         INTCON,INTF     ; Si resetta il flag di RB0/INT
                retfie

                END
```

Programma proposto.

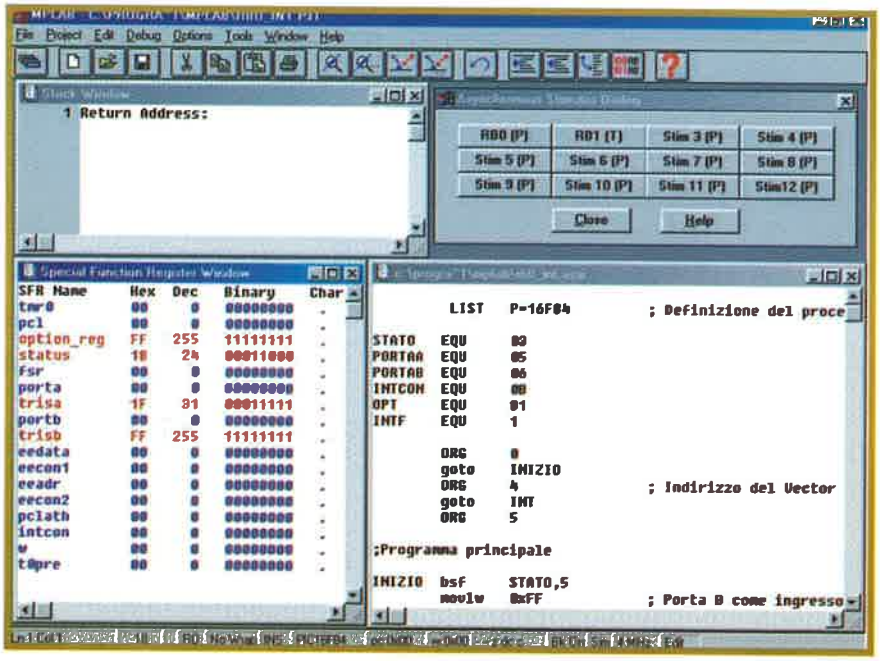


Indirizzi delle istruzioni nella memoria di programma.

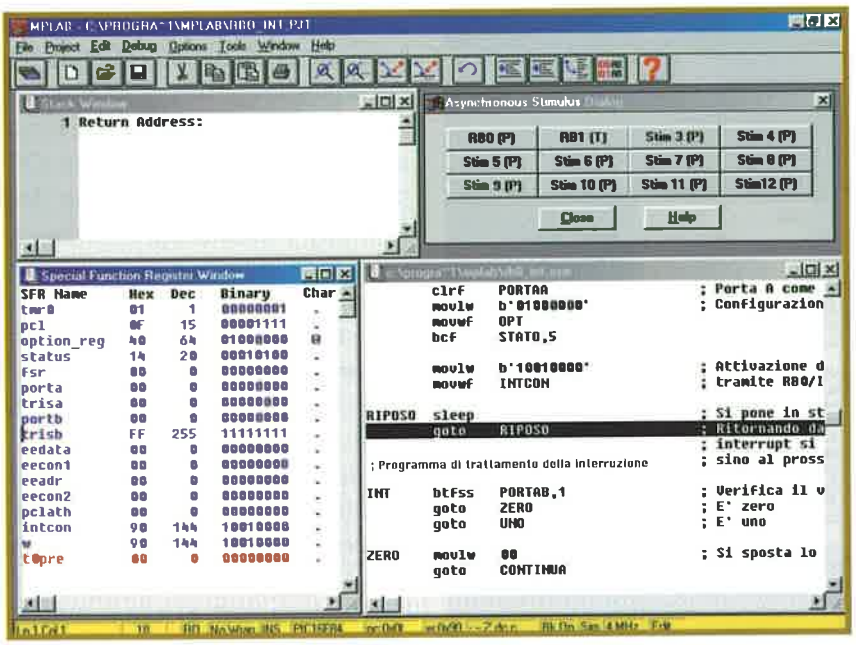
La novità è nel modo in cui il programma è organizzato, anche se di questo abbiamo già parlato nella parte dedicata alla struttura generale di un programma. Per prima cosa, come sempre, bisogna decidere che microcontroller vogliamo usare, e definirne le variabili. La prima istruzione del programma si posiziona all'indirizzo 0 della memoria di programma grazie alla direttiva ORG. Come elemento nuovo però, abbiamo introdotto gli interrupt e il vector di interrupt si trova all'indirizzo 4 di questa memoria, questo significa che quando si produce un interrupt, il PC punta a questo indirizzo sperando di trovare un'istruzione della Routine di Servizio all'Interrupt (RSI). Per questo motivo dobbiamo mettere la prima istruzione della routine all'indirizzo 4, e lo facciamo utilizzando nuovamente la direttiva ORG. Nella figura in alto possiamo vedere come restano localizzate le istruzioni nella memoria del programma, dopo averlo assemblato. Notate che negli indirizzi da 1 a 3 si trova il valore 3FFF, ciò significa che non ci sono istruzioni.

Il programma principale realizza solo le configurazioni: imposta la porta B come ingressi, dato che abbiamo bisogno di RB0 e RB1, la porta A come uscite per visualizzare il risultato delle operazioni da realizzare, e i registri INTCON e OPTION (OPT) per configurare l'abilitazione degli interrupt e selezionare il fronte con cui si produrranno. Il microcontroller viene poi messo in stato di riposo dall'istruzione sleep, e rimane lì, senza eseguire l'istruzione successiva, sino a che si sveglia, al verificarsi di un interrupt.

Un interrupt è un avvenimento asincrono, quindi non si conosce il momento in cui avverrà. Nel nostro caso l'interrupt si produce grazie all'attivazione del piedino RB0/INT, quindi sarà provocato dall'utente esterno premendo un pulsante. Il PC punterà all'indirizzo 4 e inizierà la routine dedicata all'interrupt. In essa si verifica il valore del bit 1 della porta B, configurata come ingresso, e a seconda del suo valore, carica nel registro di lavoro W il dato 00 oppure 1F, per poi passarlo alla porta A e visualizzarlo. Prima di uscire dalla routine con l'istruzione retfie bisogna resettare il flag di questo interrupt a 0. Questo bit è stato messo a 1 nel momento in cui si è



Finestre necessarie per la simulazione.



Il microcontrollore permane in stato di riposo in attesa di un interrupt.

verificato l'interrupt, così nel caso fossero previsti anche interrupt di diverso tipo, consultando i flag si saprà quale di questi si è verificato, dato che solo un flag sarà a 1. Inoltre quando si verifica un interrupt, l'abilitazione generale (bit 7 di INTCON) viene messa a 0 automaticamente, e sino a che stiamo eseguendo la subroutine del primo, non se ne potranno generare altri. Eseguendo l'istruzione di ritorno dall'interrupt questo bit viene nuovamente messo a 1, e l'istruzione "goto NIENTE" fa sì che il microcontroller torni in stato di riposo.

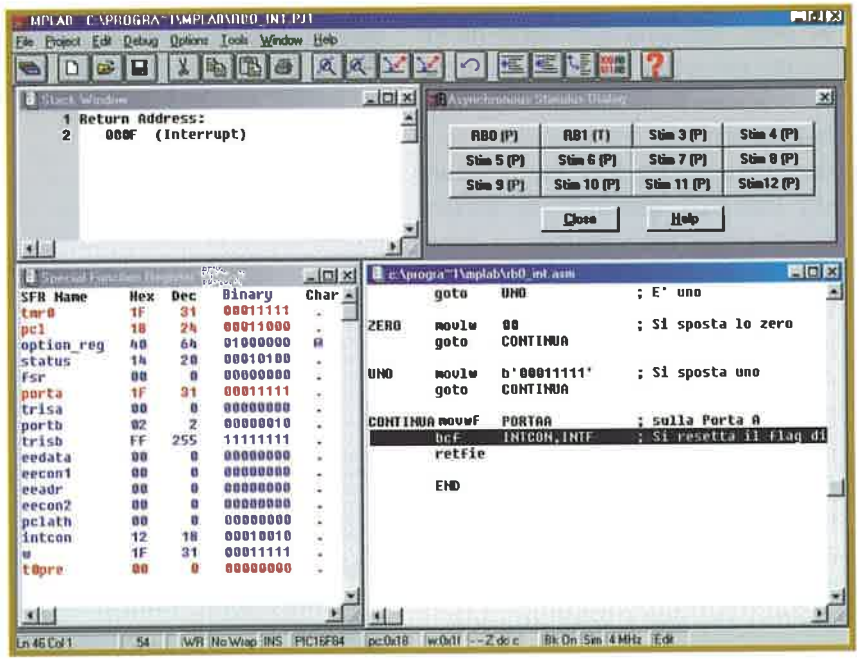
SIMULAZIONE DEL PROGRAMMA

Per simulare il programma avremo bisogno di una serie di finestre che ci permettano di osservare quello che avviene all'interno del microcontroller. Queste finestre sono: quella dei registri specifici (Window>Special Function Register), per vedere se le funzioni si compiono, lo stack (Window>Stack) per vedere gli indirizzi di ritorno che vengono memorizzati, e il programma vero e proprio, per vedere la sua evoluzione. Inoltre avremo bisogno della finestra che simula l'introduzione dei dati al sistema: Debug>Simulator Stimulus>Asynchronous Stimulus. In quest'ultima programmeremo due linee: la RB0 che introdurrà impulsi (Pulse) e la RB1, che simulerà un interruttore (Toggle).

Simuleremo ora il programma in forma di animazione. Il programma avanzerà sino ad arrivare all'istruzione sleep, la eseguirà e si fermerà aspettando che si produca l'interrupt, mentre punta all'istruzione successiva.

Prima di premere qualsiasi pulsante guardate il valore della linea RB1, e della porta A, entrambi nella finestra dei registri specifici. Premete il pulsante RB1 della finestra Stimulus e verificate che il valore cambi. Se ha valore 1, premendo RB0, la porta A assumerà valore 1 e se ha valore 0, premendo RB0, assumerà valore zero.

Premete ancora RB0. Vedrete che nella finestra di programma verranno eseguite rapidamente le altre istruzioni, però non potremo verificare quali, comunque vedremo che nella porta A si caricheranno valori diversi. Per verificare i salti all'interrupt fermate la simulazione ed eseguitela passo a passo (Debug>Run>Step).



Un momento della simulazione eseguita passo a passo.

L'istruzione IORLW

L'istruzione "ior" o "or inclusiva" realizza la somma logica degli operandi: il letterale che si accompagna al codice dell'istruzione e il contenuto del registro di lavoro W. Come nelle altre istruzioni dello stesso formato, il risultato si lascia nel registro di lavoro W.

Se il risultato dell'operazione è 0, si attiva il flag Z.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
i o r l w	k	

Operazione:	realizza l'operazione logica **OR** fra un valore letterale e il valore del registro di lavoro **W**
Cicli:	1
Codice OP:	11 1000 kkkk kkkk
Flags:	Z

Caratteristiche dell'istruzione iorlw.

ESEMPI CON L'ISTRUZIONE IORLW

Così come l'istruzione andlw, anche la iorlw, è un'istruzione logica, quindi l'operazione che esegue si realizza bit a bit e non prendendo come unità il registro completo. In una somma logica il risultato è sempre vero, cioè 1, quando almeno uno degli operandi lo è. Nel PIC questa istruzione prende due unici operandi, però nel caso ne prenda di più, come succede in altri linguaggi, il risultato sarà vero se almeno uno di essi lo sarà. Nella figura si rappresenta la tabella della verità relativa all'istruzione, e il suo modo di operare. L'esecuzione bit a bit è quello che realmente fa questa istruzione all'interno del PIC, mentre l'esecuzione a livello di registri è quello che succederebbe se fosse possibile con questa istruzione. Come si può osservare, a parte il fatto che un'istruzione con queste caratteristiche sui registri non fornisce un valore concreto ma solo vero (V) o falso (F), se valutiamo il risultato globale non potremo trovare differenze: indipendentemente dai registri iniziali, il risultato finale (V o F) sarà lo stesso in entrambi i casi.

TABELLA GENERALE		
Valore 1	Valore 2	RISULTATO
F	F	F
F	V	V
V	F	V
V	V	V

LAVORO CON REGISTRI			LAVORO CON BIT		
Registro 1	Letterale	Risultato	Registro 1	Letterale	Risultato
10000000	00000000	V	10000000	00000000	10000000(V)

Funzionamento dell'istruzione iorlw.

L'ISTRUZIONE IORWF

L'istruzione iorwf è il secondo modo di realizzare istruzioni di somma logica. Come mostra la figura, e secondo il formato visto fino ad ora, l'operazione si fa con i valori contenuti nei registri della memoria RAM e quello del registro di lavoro W. A seconda

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
i o r w f	f	d

Operazione: realizza l'operazione logica **OR** fra il valore del registro **f** e il valore del registro di lavoro **W**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro **f**.

Cicli: 1

Codice OP: 00 0100 dfff ffff

Flags: Z

Caratteristiche dell'istruzione iorwf.

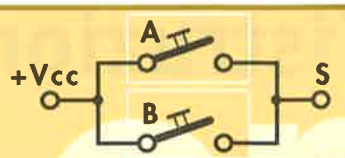


TABELLA DELLA VERITÀ		
A	B	S
0	0	0
1	0	1
0	1	1
1	1	1

Circuito con interruttori per rappresentare le istruzioni ior.

del valore del risultato può essere influenzato il flag Z. Se rappresentiamo graficamente la tabella della verità delle istruzioni ior vedremo che è sufficiente che uno solo degli interruttori sia chiuso (valore vero) per fare in modo che la corrente passi da un lato all'altro del circuito.

ESEMPI CON L'ISTRUZIONE IORWF

Parlando delle istruzioni andwf e andlw, abbiamo già introdotto il concetto di maschere, e abbiamo visto due esempi. Però, a seconda di quello che desideriamo ottenere, dobbiamo applicare una maschera o un'altra, quindi utilizzare diverse istruzioni. Pensate a questo nuovo esempio: vogliamo porre a 1 una serie di bit di un registro qualsiasi, però senza modificarne nessuno degli altri. Questo è normale con registri specifici come quelli che controllano le istruzioni, dove ogni bit deve essere trattato in modo indipendente, e non si può cambiare valore a tutto il registro. Se si tratta solo di modificare il valore di un bit si può fare con l'istruzione "bsf", però quando il numero dei bit aumenta, come nell'esempio dove vogliamo modificare 7 degli 8 bit, è più conveniente l'utilizzo delle maschere. In questo esempio modifichiamo uno dei bit a 1 e il resto si lascia uguale.

Nel caso si desideri porre dei bit a 1 e altri a 0 dovremo fare delle maschere successive con le istruzioni "ior" e "and".

Impostazione a 1 dei 7 bit più significativi di un registro senza influenzare quelli meno significativi.

Senza utilizzare la funzione ior	Utilizzando la funzione ior
bsf REG,1	movlw b'11111110'
bsf REG,2	iorwf REG,F
bsf REG,3	
bsf REG,4	
bsf REG,5	
bsf REG,6	
bsf REG,7	

Esempi dell'istruzione iorwf.

Applicazione pratica: quattro interrupt come uno solo

Abbiamo già visto il primo dei quattro interrupt che si possono verificare in un PIC16F84, quindi ne rimangono altri tre. Questa volta ne utilizzeremo uno molto particolare, dato che può essere generato da diverse cause. Stiamo parlando dell'interrupt per cambio di stato dei piedini più significativi della Porta B, vale a dire RB4-RB7.

INTERRUPT PER CAMBIO DI STATO DI RB4-RB7

Quando uno di questi piedini cambia stato, cioè passa da 0 a 1 o viceversa, si attiva il flag RBIF, e il PC punta il Vector di Interrupt. Nella Routine di Servizio dell'Interrupt, se l'applicazione lo richiede, si dovrà verificare quale delle quattro linee è quella che ha cambiato di stato, e se il nuovo valore è 1 oppure 0. Questo sarà ancora più complicato se le periferiche collegate alla porta B sono pulsanti anziché interruttori, dato che il loro valore torna allo stato iniziale dopo che sono stati rilasciati. Comunque nell'esempio che stiamo per fare non ci preoccuperemo di questa verifica, ma tratteremo in egual modo l'insieme delle quattro linee, senza verificare quale ha provocato l'interrupt.

Anche questa volta il programma principale è utilizzato solo per le inizializzazioni dei registri, infatti realizzeremo tutte le altre operazioni nella routine di interrupt.

Come potete osservare, questa volta nell'organigramma non siamo scesi a basso livello, sono state lasciate le azioni da realizzare in linguaggio naturale. In seguito, al momento di metterlo in pratica, ci preoccuperemo di vedere quali istruzioni o che tipo di elaborazione saranno necessarie in questo caso. Per lavorare con gli interrupt, avremo nuovamente bisogno del registro INTCON, i bit coinvolti in questo caso specifico sono riportati nella figura.

Organigramma della routine dell'interrupt.

Come nel caso dell'interrupt per RB0/INT, il bit 7 del registro INTCON dovrà essere messo a 1 per l'abilitazione generale degli interrupt, inoltre il suo bit specifico, il bit 3, dovrà anch'esso valere 1. I rimanenti interrupt dovranno essere disabilitati, dato che non sono contemplati. Il flag RBIF (bit 0 del registro INTCON) passerà a 1 quando si verificherà l'interrupt.

Si vuole proteggere una casa dall'ingresso di intrusi, quindi si montano quattro sensori digitali: uno sulla porta e tre sulle finestre. Quando uno qualunque di questi si attiva, si visualizza lo stato dei sensori in modo da conoscere la causa dell'allarme e si attiva un cicalino.

Enunciato per l'utilizzo dell'interrupt per cambio di RB4-RB7.

INTCON	1	0	0	0	1	0	0	0

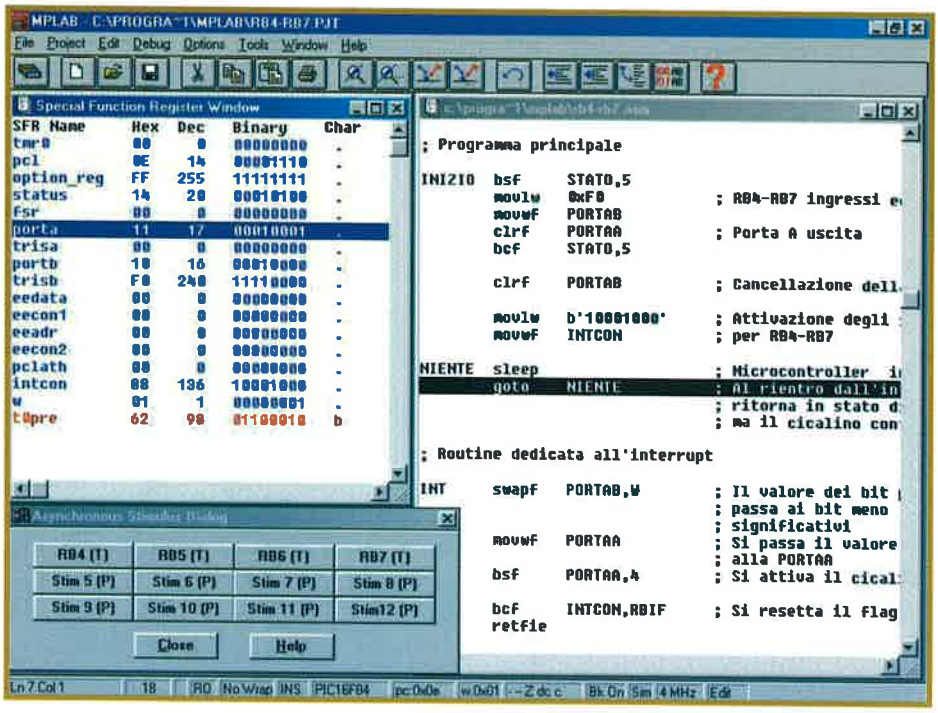
Registro importante nell'interrupt per cambio di stato di RB4-RB7.

```
                LIST        P=16F84         ; Definizione del processore

STATO           EQU         03              ; Definizione di variabile
PORTAA          EQU         05
PORTAB          EQU         06
INTCON          EQU         0B
RBIF            EQU         0
W               EQU         0

                ORG         0
                goto        INIZIO
                ORG         4               ; Posizione del vector di interrupt
                goto        INT
                ORG         5               ; Programma principale

INIZIO          bsf         STATO,5
                movlw       0xF0            ; RB4-RB7 ingressi e il resto uscite
                movwf       PORTAB
                clrf        PORTAA          ; Porta A uscita
                bcf         STATO,5

                clrf        PORTAB          ; Cancellazione della PORTAB

                movlw       b'10001000'     ; Attivazione degli interrupt
                movwf       INTCON          ; per RB4-RB7

NIENTE          sleep                       ; Microcontroller in stato di riposo
                goto        NIENTE          ; Al rientro dall'interrupt
                                            ; ritorna in stato di riposo
                                            ; ma il cicalino continua a suonare

                                            ; Routine dedicata all'interrupt

INT             swapf       PORTAB,W        ; Il valore dei bit più significativi
                                            ; passa ai bit meno
                                            ; significativi
                movwf       PORTAA          ; Si passa il valore dell'allarme
                                            ; alla PORTAA
                bsf         PORTAA,4        ; Si attiva il cicalino, collegato a RA4

                bcf         INTCON,RBIF     ; Si resetta il flag di RB4-RB7
                retfie

                END
```

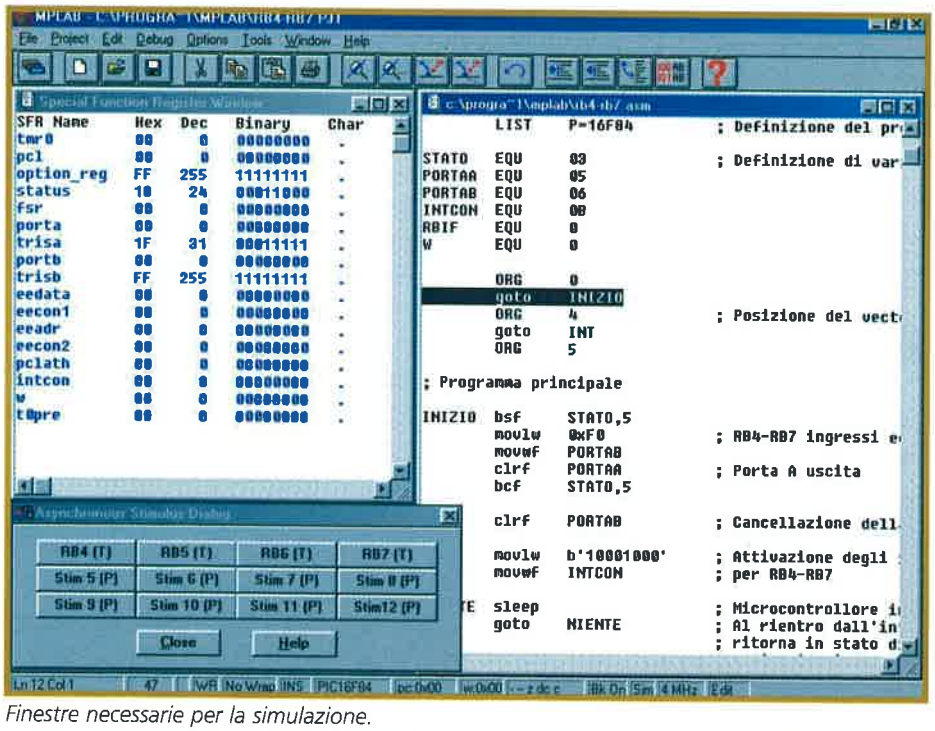
Programma proposto.

IL PROGRAMMA COMPLETO

Come potrete verificare, il programma è simile a quello già visto per l'interrupt RB0/INT. Nella parte principale, dopo aver realizzato le configurazioni opportune per questo caso, il programma entra in stato di riposo, e attende che si generi un interrupt. Parte della porta B si configura come ingresso, perché sarà quella a cui saranno collegati i sensori. Sulla porta A visualizzeremo l'informazione relativa all'allarme attivato, e sulla linea RA4 dovrà essere collegato un cicalino che si attiverà con l'interrupt. Per realizzare quanto detto, utilizzeremo l'istruzione swapf per scambiare i bit più significativi – dove sono collegati i sensori – con quelli meno significativi, dato che la porta A dispone solo di 5 linee e non di 8 come la porta B. Al rientro dalla routine di interrupt, il microcontroller tornerà in stato di riposo, però grazie alle sue caratteristiche, questo cicalino rimarrà attivato anche se non si esegue nessuna nuova istruzione, in questo modo l'allarme non si potrà scollegare, anche se i "supposti ladri" richiuderanno la porta o la finestra da cui sono entrati, a meno che non si resetti il sistema. Prima di uscire dalla routine di interrupt bisogna resettare il flag dell'interrupt prodotto.



Situazione della simulazione dopo aver attivato RB4.



Finestre necessarie per la simulazione.

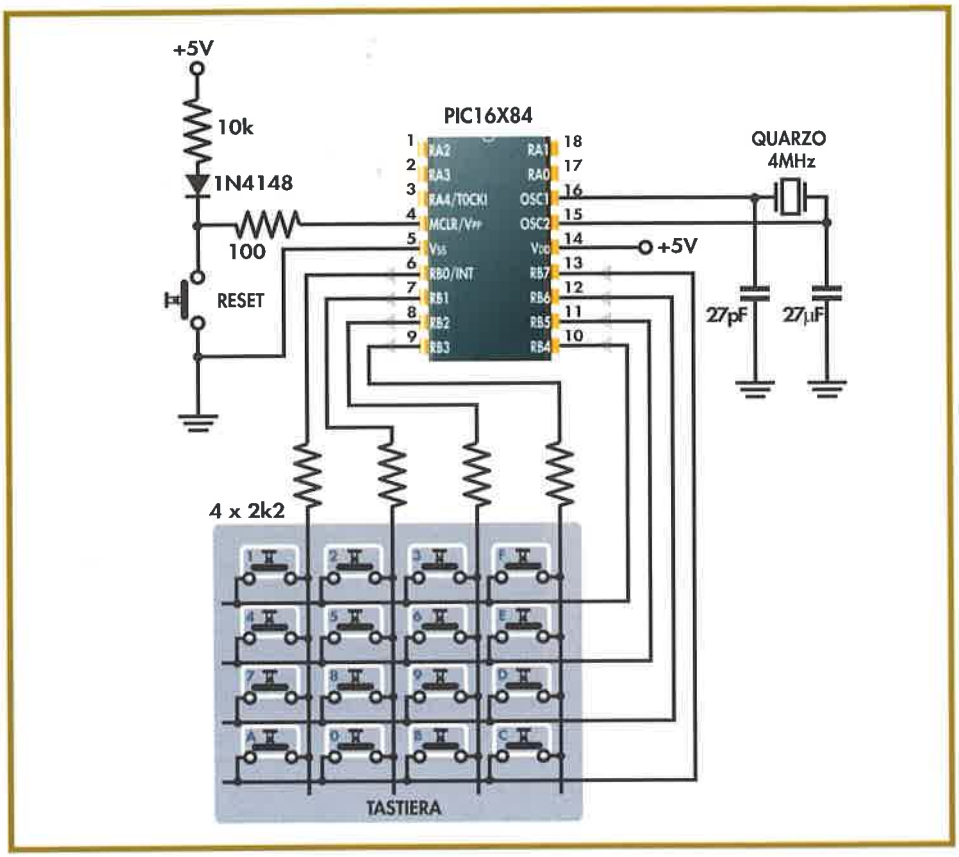
SIMULAZIONE DEL PROGRAMMA

Simuleremo il programma in modo continuo (Debug>Run>Animate) e lo seguiremo tramite i registri specifici (Window>Special Function Register) poi introdurremo dei valori tramite le linee RB4-RB7. Queste linee le dovremo configurare con la finestra Debug>Simulator Stimulus>Asynchronous Stimulus, per simulare un interruttore (Toggle) come nella figura, o un pulsante (Pulse).

Dopo aver attivato la simulazione, potremo, ad esempio,

attivare la linea RB4. Vedremo che si produrrà un rapido cambiamento nelle linee di esecuzione del programma, e che la linea che evidenzia le istruzioni in corso tornerà nella sua posizione di stato di riposo dopo aver eseguito la Routine di Servizio all'Interrupt. Qualcosa comunque è cambiato: nella porta A sono ora indicati i valori della porta B, e si è attivata la linea RA4, il cicalino.

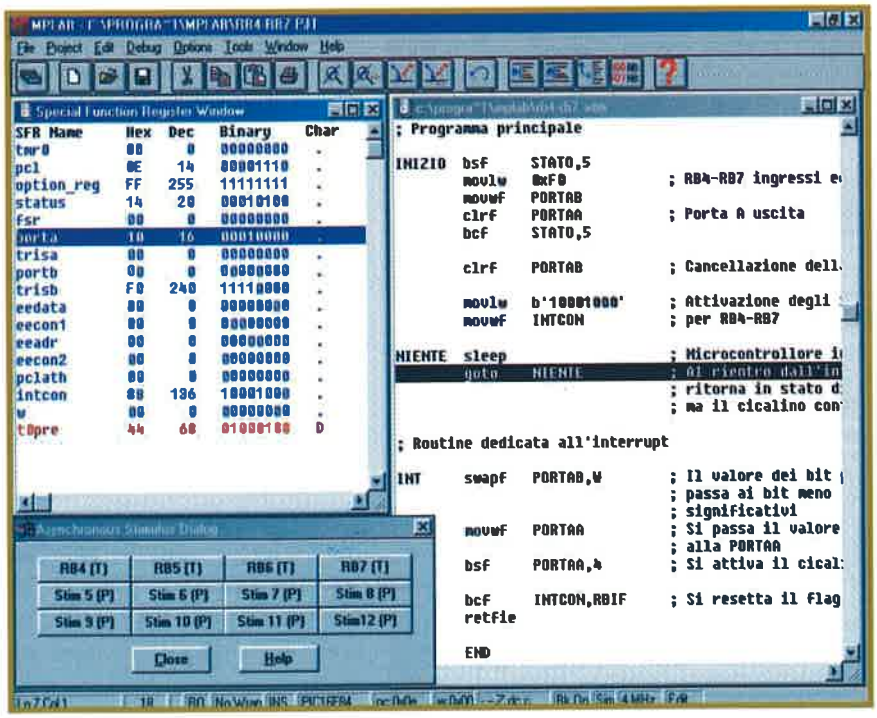
Anche se torneremo a cliccare su RB4 per riportarla alla sua situazione iniziale, il cicalino non si disattiverà, anche se il nuovo valore della porta B si potrà vedere sulla porta A.



Collegamenti di una tastiera esadecimale per l'utilizzo con interrupt.

UTILIZZO DELL'INTERRUPT PER RB4-RB7

In questo caso abbiamo utilizzato l'interrupt per RB4-RB7 in un esempio semplice ed efficiente. Un utilizzo tipico di questo interrupt è la rilevazione dell'attivazione di un pulsante di una tastiera matriciale. Queste tastiere possono contenere quantità diverse di pulsanti, quelle più comuni sono le esadecimali. Una tastiera occupa la porta B al completo: i bit più significativi configurati come ingressi sono quelli che utilizzano l'interrupt per RB4-RB7, e i bit meno significativi sono configurati come uscite. Premendo un pulsante si produrrà un interrupt, all'interno della routine bisognerà verificare di quale pulsante si tratta, per realizzare le operazioni opportune. Questo uso apre le porte ad una moltitudine di applicazioni, che tramite l'utilizzo delle tastiere, forniscono la possibilità di introdurre dati in modo versatile, e in modo familiare per l'utente.



Situazione della simulazione dopo aver riportato la linea RB4 come all'inizio.

L'istruzione XORLW

Esistono due istruzioni di somma logica: la inclusiva "ior", di cui abbiamo già parlato e la esclusiva "xor", a cui dedichiamo queste pagine.

Una somma logica esclusiva fra due operandi significa che uno e solo uno di essi può essere a 1 per fare in modo che lo sia anche il risultato. Nel caso della xorlw, la somma è realizzata fra i bit che compongono il registro di lavoro W e quelli del letterale che si accompagna al codice dell'istruzione. Il risultato si carica nel registro di lavoro W, e si attiva il flag Z nel caso valga zero.

TABELLA GENERALE

Valore 1	Valore 2	RISULTATO
F	F	F
F	V	V
V	F	V
V	V	F

LAVORO CON REGISTRI			LAVORO CON BIT		
Registro 1	Letterale	Risultato	Registro 1	Letterale	Risultato
10101010	01010101	F	10101010	01010101	11111111(V)

TABELLA PER TRE OPERANDI

Valore 1	Valore 2	Valore 2	RISULTATO
F	F	F	F
F	F	V	V
F	V	F	V
F	V	V	F
V	F	F	V
V	F	V	F
V	V	F	F
V	V	V	V

Funzionamento dell'istruzione xorlw.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
x o r l w	k	

Operazione: realizza l'operazione logica **OR esclusiva** fra un letterale e il valore del registro di lavoro **W**.
Cicli: 1
Codice OP: 11 1010 kkkk kkkk
Flag: Z

Caratteristiche dell'istruzione xorlw.

ESEMPI CON L'ISTRUZIONE XORLW

L'operazione di somma logica esclusiva segue la tabella della verità della figura.

Nel caso di linguaggi che permettano più di due operandi, il risultato sarà vero se esiste un numero dispari di valori veri e sarà falso per un numero uguale a zero oppure pari di valori veri.

È come se si realizzassero operazioni concatenate di istruzioni xor fra due operandi.

L'istruzione XORWF

La xorwf realizza la stessa funzione della xorlw però prendendo i valori da locazioni differenti, cioè dal registro di lavoro W e da un registro della memoria RAM. Ancora una volta il risultato si può scrivere nello stesso registro oppure in W, attivando il flag Z in caso che sia zero.

ESEMPI CON L'ISTRUZIONE XORWF

In questo caso utilizzeremo l'istruzione xor per fare delle comparazioni. Approfitteremo del fatto che il flag Z si attiva in presenza di un risultato uguale a zero, inoltre utilizzeremo nuovamente le maschere.

```
Compara le uguaglianze fra due registri
movf        REG1,W
xorwf       REG2,W
btfss       STATO,Z
goto        NO_UGUALE
goto        UGUALE
Se REG1 = REG2 il risultato sarà 0 e il flag Z passerà a 1

Compara l'uguaglianza di un registro con un valore determinato
movf        REG1,W
xorlw       b'01010000'
btfss       STATO,Z
goto        NO_UGUALE
goto        UGUALE
Se REG1 = 01010000 il risultato sarà 0 e il flag Z passerà a 1
```

Esempi con l'istruzione xor.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
x o r w f	f	d

Operazione: realizza l'operazione logica **OR esclusiva** fra un valore del registro **f** e il valore del registro di lavoro **W**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro **f**.

Cicli: 1
Codice OP: 00 0110 dfff ffff
Flag: Z

Caratteristiche dell'istruzione xorwf.

ESEMPI CON L'ISTRUZIONE XOR

Nell'esempio si mostrano due tipi di comparazioni, una fra registri e l'altra fra un registro e un valore letterale.

Per entrambi si utilizza il registro di lavoro W, a cui si porta il valore del primo registro, e in seguito dopo averlo comparato con il valore letterale o con un altro registro, si salta nel caso che il flag Z si sia attivato, cioè nel caso in cui il risultato valga zero, il che significa che entrambi i valori sono uguali.

Applicazione pratica: la EEPROM avvisa al termine della scrittura

Come abbiamo visto nelle applicazioni mostrate sino ad ora, gli interrupt utilizzati bene evitano del lavoro al processore, lasciandolo in stato di riposo o libero di dedicarsi ad altre cose. Questo è utile con risorse che richiedono un determinato tempo ma che possono funzionare in modo autonomo, come nel caso della memoria EEPROM dei dati. Come ricorderete questa memoria si legge e si scrive elettricamente e i suoi dati non vanno persi quando si toglie alimentazione. Se il processo di lettura è immediato, quello di scrittura richiede diversi millisecondi per essere realizzato, una volta dato l'ordine.

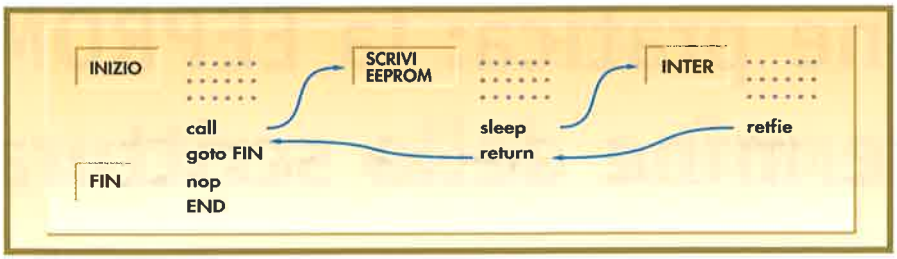
INTERRUPT PER FINE DEL CICLO DI SCRITTURA DELLA EEPROM

Quando parlammo per la prima volta della EEPROM costruimmo due subroutines, una di lettura e una di scrittura. Ora modificheremo quest'ultima per fare in modo che si produca un interrupt ogni volta che termina la scrittura di un dato nella EEPROM. La subroutine di lettura non necessita di modifiche, però il programma principale dove si trova dovrà essere cambiato per adattarsi agli interrupt.

Nel programma della figura non è stata inclusa la subroutine di lettura per non distrarre la nostra attenzione. La chiamata all'etichetta della Routine di Servizio dell'Interrupt è stata collocata nel Vector di Interrupt che è all'indirizzo 4 della memoria di programma. Sceglieremo nuovamente i valori 03 e 27 come indirizzo della memoria e dato da scrivere rispettivamente. Nel programma principale dob-

```
                LIST    P=16F84     ; Utilizziamo il PIC16F84
                RADIX   HEX         ; Sistema di numerazione esadecimale

WR              EQU     1
WREN            EQU     2
EEIF            EQU     4
RP0             EQU     5
GIE             EQU     7

STATUS          EQU     0X03
EEDATA          EQU     0X08
EEADR           EQU     0X09
INTCON          EQU     0X0B
EEADR_VAR       EQU     0X0C
EEDATA_VAR      EQU     0X0D
EECON1          EQU     0X88
EECON2          EQU     0X89

                ORG     0           ; Il programma inizia all'indirizzo 0
                goto    INIZIO
                ORG     4           ; Vector di interrupt
                goto    INTER
                ORG     5

INIZIO          movlw   03          ; Metto l'indirizzo da scrivere
                movwf   EEADR_VAR   ; su una variabile ausiliaria
                movlw   27          ; Metto il dato da scrivere
                movwf   EEDATA_VAR  ; su una variabile ausiliaria
                movlw   b'11000000' ; Abilito gli interrupt
                movwf   INTCON
                call    SCRIVI_EEPROM ; Chiamo la routine
                goto    FINE

SCRIVI_EEPROM
                movf    EEADR_VAR,W ; Porto l'indirizzo sul registro
                movwf   EEADR
                movf    EEDATA_VAR,W ; Porto il dato sul registro
                movwf   EEDATA
                bsf     STATUS,RP0  ; Passo al banco 1
                bcf     INTCON,GIE  ; Disabilito gli interrupt
                bsf     EECON1,WREN ; Abilito la scrittura
                movlw   0X55        ; Inizio la sequenza obbligatoria
                movwf   EECON2      ; di scrittura
                movlw   0XAA
                movwf   EECON2
                bsf     EECON1,WR   ; Ordine di scrittura
                bsf     INTCON,GIE  ; Abilito gli interrupt
                sleep               ; Entro in stato di riposo
                                    ; da cui uscirò con un interrupt
                return

INTER           bcf     EECON1,EEIF ; Resetto il flag
                bcf     EECON1,WREN ; Disabilito la scrittura
                bcf     STATUS,RP0  ; Passo al banco 0
                retfie              ; Rientro dall'interrupt

FINE            nop
                END
```

Subroutine e programma per la scrittura della EEPROM dei dati con interrupt.



Schema del processo che si esegue nel programma.

biamo solo configurare gli interrupt che si possono produrre e chiamare la routine di scrittura prima di andare alla fine del programma.

Da parte sua la sua la subroutine di scrittura ha una parte critica, la scrittura del registro EECON2 secondo le specifiche del costruttore, durante la quale si deve inibire l'accettazione di interrupt, cosa che si ottiene mettendo a 0 il bit GIE di INTCON. Terminata questa prima scrittura si può tornare ad accettare gli interrupt riportando a 1 questo stesso bit, e dare l'ordine di scrittura. Quando termina la scrittura della EEPROM si uscirà dallo stato di riposo grazie ad un interrupt, e si passerà ad eseguire la corrispondente Routine di Servizio dell'interrupt. In essa eseguiremo solamente il reset del flag di fine scrittura e del bit che permette questa scrittura, prima di rientrare. Usciti dalla routine di interrupt si esegue l'istruzione successiva alla "sleep", la "return" con cui si esce anche dalla subroutine e si va alla fine del programma. Nello schema sono riportati i salti che si producono lungo il programma. Al momento della simulazione dobbiamo aprire le finestre dei registri speciali, quella della EEPROM e il programma in sé. Inoltre saranno molto utili la finestra dello stack e il clock di sistema.

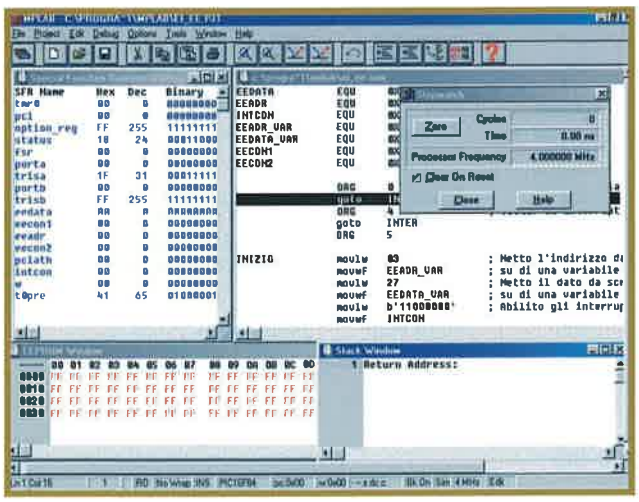
Facendo la simulazione mediante animazione (Debug>Run>Animate) si vedono chiaramente i salti che si producono lungo l'esecuzione, sia per il movimento della linea-cursore lungo le istruzioni del programma, sia per gli indirizzi di ritorno che si memorizzano nello stack. Comunque, dato che si impiegano almeno 10 ms per uscire dallo stato di riposo, questo tipo di simulazione può risultare tedioso, anche se è raccomandabile quando si sta iniziando.

Inoltre si può eseguire la simulazione con l'opzione più rapida (Debug>Run>Run) e fermarla quando il clock di sistema segna un po' più di 10 ms, in quel momento all'indirizzo 03 della EEPROM dei dati deve apparire il valore 27.

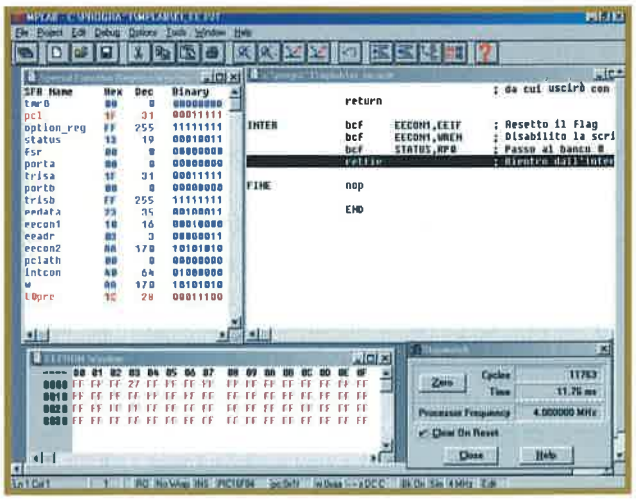
APPLICAZIONE PRATICA

Una volta visto come funziona l'interrupt per fine scrittura della EEPROM, integriamolo in un programma riferito ad un'applicazione concreta. Come si può vedere tutto si complica poco a poco. Guardate l'organigramma nella pagina successiva.

Dopo la configurazione della porta B come uscita per mostrare il valore della EEPROM, della porta A come ingresso per inizializzare la EEPROM, quindi con la somma di nuovi valori, e con l'abilitazione dell'interrupt per fine della scrittura, si arriva al blocco principale del programma. Il passo seguente sarà mostrare il valore



Preparazione delle finestre per la simulazione del programma.



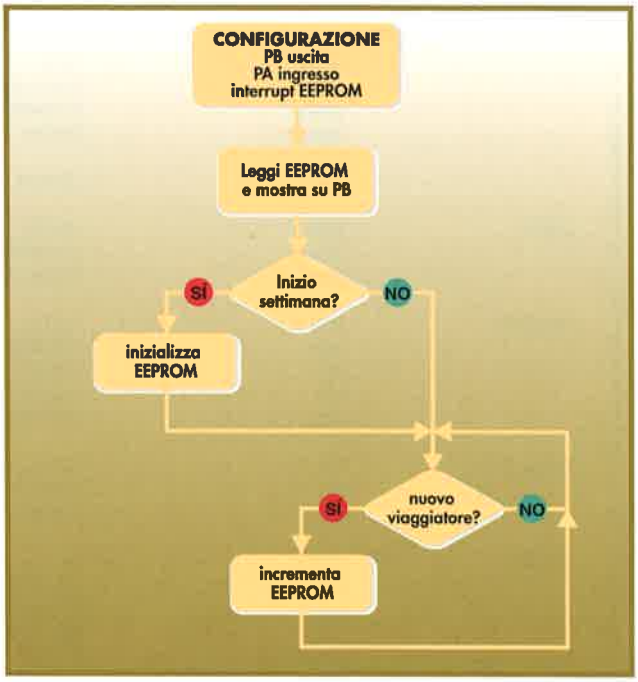
Finestre dopo la simulazione del programma.

**Si vuole utilizzare la memoria EEPROM dei dati per memorizzare la raccolta settimanale di un autobus.
All'inizio della settimana l'utente comanda con un interruttore l'inizializzazione della memoria. A partire da questo momento ogni impulso introdotto da un pulsante indicherà il pagamento di un nuovo biglietto che si memorizzerà nella EEPROM. Ad ogni inizio giornata all'accensione del sistema verrà mostrato per un periodo di tempo il valore memorizzato all'interno della EEPROM, tramite dei diodi LED.**

Enunciato del programma proposto.

della EEPROM. A seconda se si è all'inizio della settimana o no, si inizializzerà il valore della EEPROM, il che significa scrivere in essa un valore zero. In seguito si entrerà in un ciclo infinito che incrementerà il valore della EEPROM ad ogni nuovo passeggero, sarebbe a dire ad ogni impulso su una linea della porta A. Si uscirà da questo ciclo con un reset di sistema, o quando si toglie alimentazione al medesimo. Dato che ogni nuovo valore è scritto nella EEPROM, i dati non andranno persi togliendo alimentazione.

Passando al programma in assembler bisogna fare attenzione a non perdersi, seguendo passo passo l'organizzazione pensata nell'organigramma. Ricordate che i rombi di solito si trasformano in domande che utilizzano informazioni di tipo salto: "btfss", "btfsc",..., e i relativi rami sono istruzioni "goto". I riquadri sono di solito gruppi di istruzioni, che potrebbero anche includere subroutine.



Organigramma del programma proposto.

```
                LIST    P=16F84         ; Utilizzo il PIC16F84
                RADIX   HEX             ; Sistema di numerazione esadecimale

W               EQU     0
F               EQU     1
RD              EQU     0
WR              EQU     1
WREN            EQU     2
EEIF            EQU     4
RP0             EQU     5
EEIE            EQU     6
GIE             EQU     7
STATUS          EQU     0X03
PORTA           EQU     0X05
PORTB           EQU     0X06
EEDATA          EQU     0X08
EEADR           EQU     0X09
INTCON          EQU     0X0B
EEADR_VAR       EQU     0X0C
EEDATA_VAR      EQU     0X0D
TRISA           EQU     0X85
TRISB           EQU     0X86
EECON1          EQU     0X88
EECON2          EQU     0X89
                ORG     0               ; Il programma inizia all'indirizzo 0
                goto    INIZIO
                ORG     4               ; Vector di interrupt
                goto    INTER
                ORG     5
INIZIO          bsf     STATUS,RP0      ; Banco 1
                clrf    TRISB           ; Porta B uscita
```

Soluzione del programma proposto.

```
                movlw   1F          ; Porta A ingresso
                movwf   TRISA
                bcf     STATUS,RP0  ; Banco 0
                movlw   b'11000000' ; Abilito gli interrupt
                movwf   INTCON

                call    LEGGERE     ; Legge la EEPROM e la porta su PB

SCELTA          btfss   PORTA,0     ; Scelta da fare
                goto    SOMMARE     ; Se RA4=0 non è inizio settimana
                goto    INIZIALIZZA ; Se RA4=1 è l'inizio settimana

SOMMARE         btfss   PORTA,1     ; Verifica sul pulsante
                goto    SOMMARE     ; Non è stato premuto
                incf    EEDATA_VAR,F ; Premuto, aumenta l'incasso e scrivi la EEPROM
                call    SCRIVERE    ;
                goto    SOMMARE     ; Continua l'incasso

INIZIALIZZARE   clrf    EEDATA_VAR  ; Inizializza l'incasso
                call    SCRIVERE
                goto    SOMMARE     ; Inizia la raccolta

LEGGERE         movlw   00          ; Scriviamo l'indirizzo da leggere
                movwf   EEADR_VAR   ; in una variabile ausiliaria
                call    LEGGI_EEPROM ; Chiamata alla routine di lettura
                movf    EEDATA_VAR,W ; Si visualizza il risultato
                movwf   PORTB       ; sulla Porta B
                return

LEGGI_EEPROM    movf    EEADR_VAR,W ; Spostiamo l'indirizzo da leggere
                movwf   EEADR       ; sul suo registro
                bsf     STATUS,RP0  ; Passiamo al banco 1
                bsf     EECON1,RD   ; Ordine di lettura
                bcf     STATUS,RP0  ; Passaggio al banco 0
                movf    EEDATA,W    ; Spostiamo il dato sul
                movwf   EEDATA_VAR  ; registro ausiliario
                return

SCRIVERE        movlw   00          ; Spostiamo l'indirizzo da scrivere
                movwf   EEADR_VAR   ; sul suo registro
                call    SCRIVI_EEPROM ; Chiamata alla routine di scrittura
                return

SCRIVI_EEPROM
                movf    EEADR_VAR,W ; Porto l'indirizzo sul registro
                movwf   EEADR
                movf    EEDATA_VAR,W ; Porto il dato sul registro
                movwf   EEDATA
                bsf     STATUS,RP0  ; Passo al banco 1
                bcf     INTCON,GIE  ; Disabilito gli interrupt
                bsf     EECON1,WREN ; Abilito la scrittura
                movlw   0X55        ; Inizio la sequenza obbligatoria
                movwf   EECON2      ; di scrittura
                movlw   0XAA
                movwf   EECON2
                bsf     EECON1,WR   ; Ordine di scrittura
                bsf     INTCON,GIE  ; Abilito gli interrupt
                sleep               ; Entro in stato di riposo
                                    ; da cui uscirò con un interrupt
                return

INTER           bcf     EECON1,EEIF ; Resetto il flag
                bcf     EECON1,WREN ; Disabilito la scrittura
                bcf     STATUS,RP0  ; Passo al banco 0
                retfie              ; Ritorno dall'interrupt

                END
```

Soluzione del programma proposto. (continuazione).

L'istruzione RLF

L'obiettivo di questa istruzione è quello di ruotare verso sinistra il contenuto del registro f di una posizione, cioè di un bit. Il bit che esce dalla sinistra è il più significativo, verrà immagazzinato nel bit di riporto (C) del registro di STATO, e il bit che manca da destra, il meno significativo, sarà riempito con quello che era contenuto nel bit di riporto.

Come si può vedere nella figura del funzionamento dell'istruzione, se si esegue l'istruzione nove volte di seguito si torna alla situazione iniziale, qualunque siano i valori di partenza.

ESEMPI CON L'ISTRUZIONE RLF

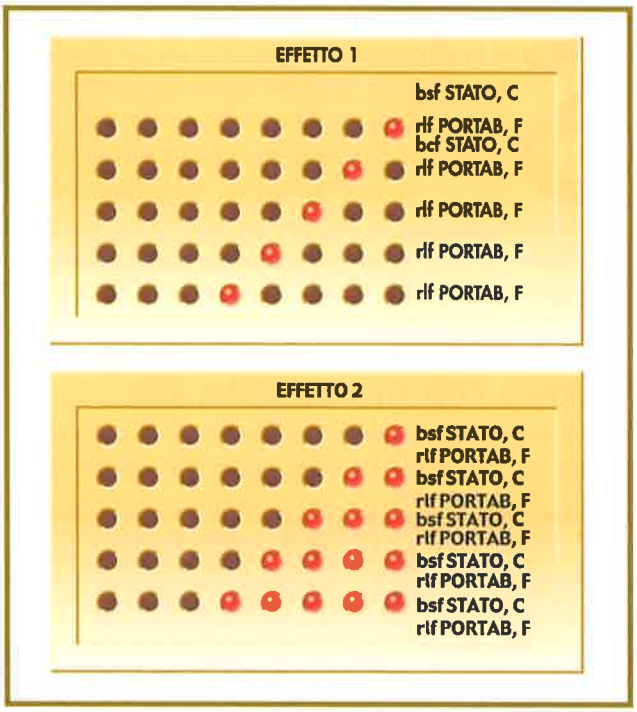
Il primo esempio che ci viene in mente è di creare effetti luminosi a partire da questa istruzione: da una luce che si va muovendo da destra a sinistra, fino ad una progressione di luci che si accendono sempre da destra a sinistra. Però oltre a tutto questo, la rlf ci permette di implementare la moltiplicazione dei numeri binari. Guardate l'esempio seguente: moltiplicare per due un numero è la stessa cosa che sommare questo numero a se stesso, oppure ruotare a sinistra di un bit il registro introducendo uno zero da destra. Provate ad implementare gli esempi in MPLAB, e verificate come funzionano con la loro simulazione.

MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
r l f	f	d

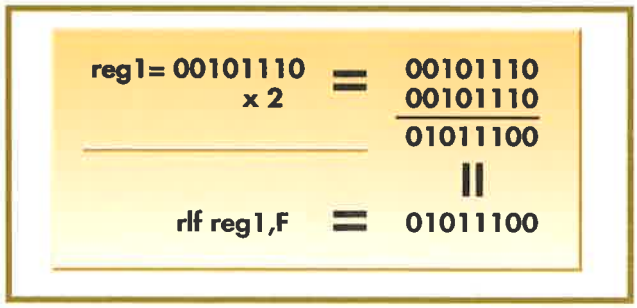
Operazione: rotazione a sinistra di un bit del registro **f**, utilizzando il bit di riporto **C**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro **f**.
Cicli: 1
Codice OP: 00 1101 dfff ffff
Flags: C

Caratteristiche dell'istruzione rlf.

Funzionamento dell'istruzione rlf.



Differenti effetti con l'istruzione rlf.



Moltiplicazione a base di rotazioni.

L'istruzione RRF

Se la rlf ruotava a sinistra di un bit in un registro, la rrf lo fa verso destra. La posizione che rimane libera a sinistra viene riempita con il valore del bit di riporto, che prenderà il valore del bit che esce da destra. A seconda del valore del parametro d, il risultato si scriverà nel registro di lavoro W, o nello stesso registro che viene usato per la rotazione. Così come con l'istruzione rlf, eseguendo nove volte la rrf si torna all'istruzione iniziale.

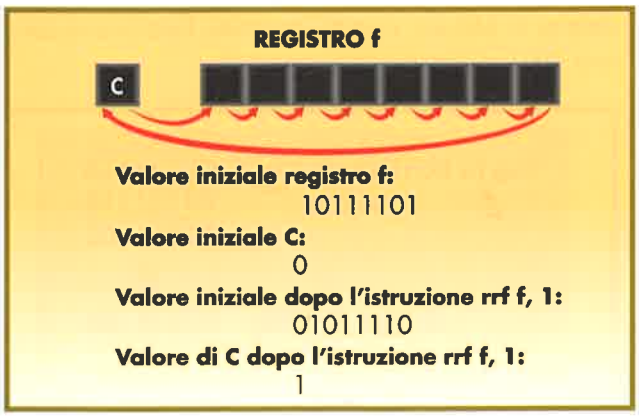
ESEMPI CON L'ISTRUZIONE RRF

Gli effetti da realizzare con l'istruzione rrf sono simili a quelli già visti con la rlf; l'interessante inizia con un mix

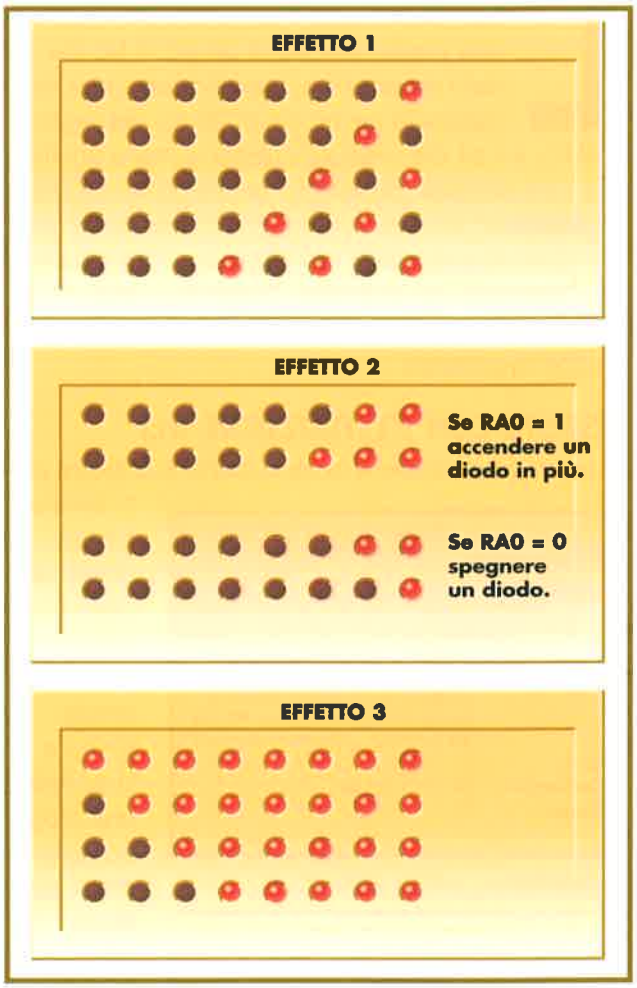
MNEMONICO	OPERANDO FONTE	OPERANDO DESTINAZIONE
r r f	f	d

Operazione: rotazione a destra di un bit del registro f, utilizzando il bit di riporto **C**. Se **d=0** il risultato si lascia nel registro di lavoro **W** e se **d=1** si lascia nello stesso registro **f**.
Cicli: 1
Codice OP: 00 1100 dfff ffff
Flags: C

Caratteristiche dell'istruzione rrf.



Rappresentazione del funzionamento dell'istruzione rrf.



Differenti effetti con le istruzioni di rotazione

delle due. Giocate con il bit di riporto e con le istruzioni di rotazione per ottenere gli effetti della figura e inventarne altre nuove.

L'operazione di divisione è più difficile da vedere intuitivamente di quella della moltiplicazione, però funziona in uguale modo, anche se all'inverso, cioè ruotando un registro verso destra si ottiene di dividere il numero per due.

Applicazione pratica: simulazione del primo programma

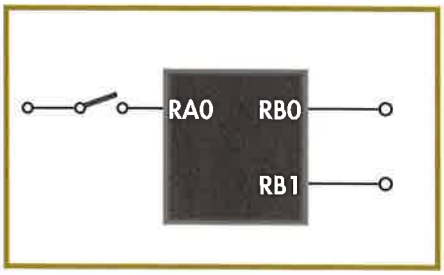
Supponiamo, a questo punto, di non avere dubbi sull'utilità del TMR0, sia per il modo temporizzatore che per il modo contatore, né sull'uso di interrupt; vediamo quindi i vantaggi derivanti dall'utilizzo contemporaneo di entrambe queste due risorse. Qui sotto proponiamo un esercizio in cui, fra le altre cose, il TMR0 provoca un interrupt quando termina la temporizzazione che avremo programmato per esso.

SCHEMA ELETTRICO

Lo schema elettrico è molto semplice: è stata scelta la porta A per collegare l'interruttore, e la porta B per i LED, anche se potrebbe essere al contrario, o un mix di entrambe le soluzioni. RB0 sarà il diodo che si accende o si spegne a seconda del valore dell'interruttore: l'interruttore a 1 farà accendere il LED, mentre a 0 lo farà spegnere. RB1 sarà la linea che supporta il diodo LED che lampeggia. Le linee di alimentazione, del quarzo e del reset, non sono riportate perché si danno per scontate.

Si tratta di simulare una macchina che ha un interruttore e due LED. Uno dei LED indica in ogni momento le variazioni che si producono sull'interruttore, e l'altro lampeggia ad una frequenza costante di 50 ms.

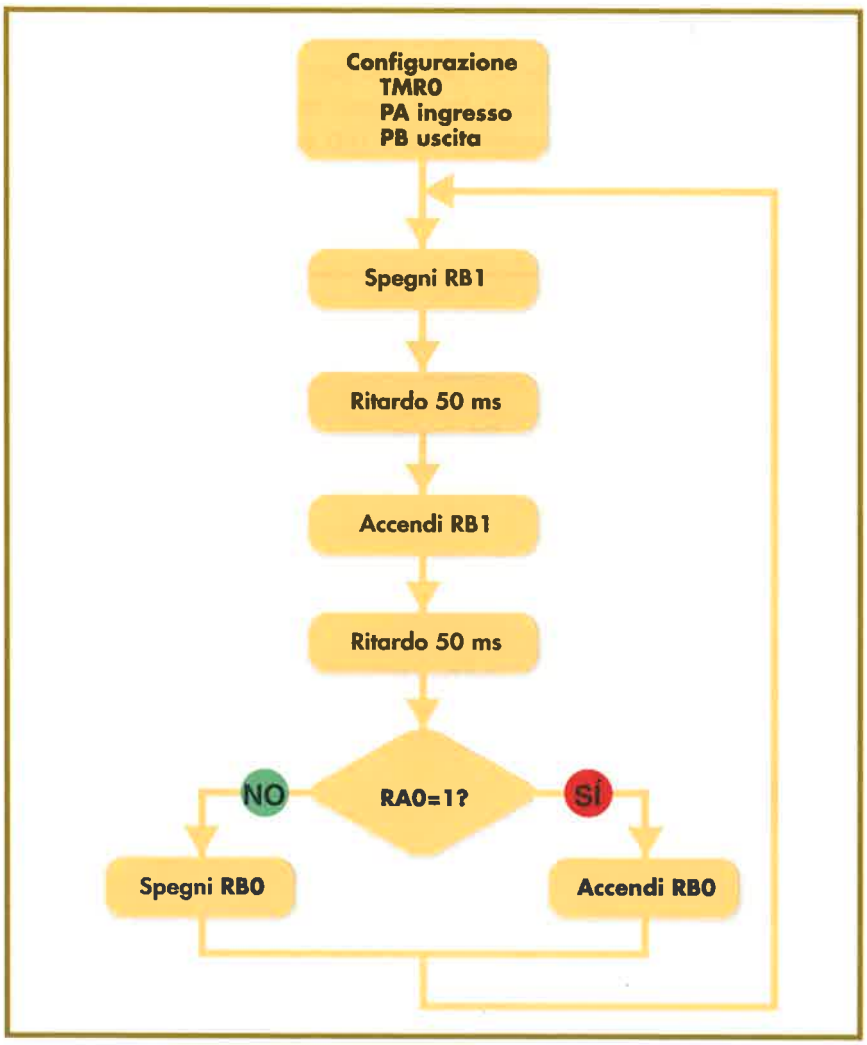
Enunciato dell'esercizio proposto.



Schema elettrico dell'esercizio proposto.

PRIMO ORGANIGRAMMA

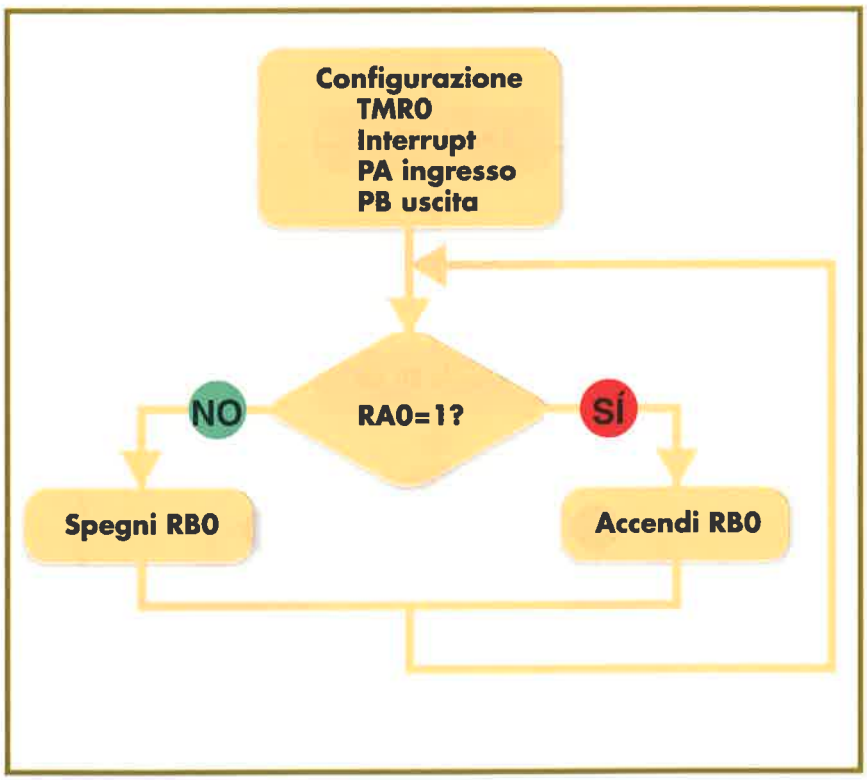
Nell'enunciato dell'esercizio non si dice nulla sull'obbligatorietà di lavorare con l'interrupt del TMR0, però nelle figure seguenti ne dimostreremo la convenienza, confrontando lo stesso esempio senza interrupt.



Primo organigramma dell'esercizio proposto.

Come si può vedere, le azioni sono prodotte in modo sequenziale. Un lampeggio corrisponde ad accendere un LED, attendere un certo tempo, spegnere il LED, attendere nuovamente lo stesso tempo, tornare ad accendere il LED e così via. Se il programma dovesse solo far lampeggiare un LED, questa sarebbe la sequenza infinita da realizzare. I ritardi sarebbero chiamate ad una subroutine realizzata dal TMR0, poi bisognerebbe aspettare che termini il conteggio per continuare con il programma. Nel programma però, si richiede anche di riflettere lo stato di un interruttore mediante l'accensione o lo spegnimento di un LED, e questo va fatto prima o dopo il lampeggio, ma in ogni caso in un momento diverso. Se decidiamo di utilizzare il TMR0 con interrupt, il primo organigramma si divide in due: uno con le azioni che riguardano il programma principale, e l'altro con le azioni da compiere nella routine di interrupt, quando il TMR0 va in overflow. Secondo questo organigramma si verifica continuamente il valore dell'interrupt per accendere o spegnere il LED corrispondente, e nello stesso tempo il TMR0 conta, per far lampeggiare l'altro LED ogni 50 ms. Le azioni quindi non sono sequenziali, ma simultanee, e solo quando il TMR0 termina di contare si ferma

Organigramma della routine di interrupt.



Secondo organigramma dell'esercizio proposto.

l'esecuzione del programma principale per accendere o spegnere il LED che lampeggia. Sono state implementate entrambe le soluzioni, con e senza interrupt, anche se in questo esempio non ne possiamo apprezzare la differenza, dato che il tempo che si perde nel lampeggio è di soli 100 ms. Se invece di cambiare stato ogni 50 ms. lo avessimo fatto ogni ora, avremmo avuto un'ora in cui anche cambiando valore all'interruttore, questo cambio non si sarebbe riflesso sul LED, dato che il processore sarebbe stato occupato nella routine di ritardo, e le transazioni dell'interruttore non sarebbero state raccolte. La verifica dell'interruttore per cambiare il valore del LED, se necessario, si avrebbe solo ogni due ore.

CONFIGURAZIONE DEI REGISTRI

Dopo aver analizzato i vantaggi di lavorare con interrupt, concentriamoci

```
                LIST    P=16F84         ; Utilizziamo il PIC16F84
                RADIX   HEX;            ; Sistema di numerazione esadecimale

TMR0            EQU     0X01
TOIF            EQU     0X02
STATUS          EQU     0X03
PORTA           EQU     0X05
PORTB           EQU     0X06
RP0             EQU     0X05
INTCON          EQU     0X0B
OPTION_REG      EQU     0X81
TRISA           EQU     0X85
TRISB           EQU     0X86

                ORG     0               ; Il programma inizia
                goto    INIZIO          ; all'indirizzo 0
                ORG     4               ; Vector di interrupt
                goto    INT

;Programma di elaborazione dell'interrupt.

INT             movf    PORTB,0
                xorlw   b'00000010'     ; Inverte il valore del LED
                movwf   PORTB
                movlw   b'00111101'     ; Carica di nuovo il TMR0
                movwf   TMR0
                bcf     INTCON,TOIF     ; Resetta il flag
                retfie                  ; Rientra

INIZIO          bsf     STATUS,RP0      ; Passa al banco 1
                movlw   b'00000000'     ; Porta B uscita
                movwf   TRISB
                movlw   b'00000001'     ; RA0 ingresso, il resto uscita
                movwf   TRISA
                movlw   b'00000111'     ; Configurazione del TMR0
                movwf   OPTION_REG
                bcf     STATUS,RP0      ; Passo al banco 0

                movlw   b'10100000'     ; Attivo gli interrupt
                movwf   INTCON          ; per TMR0
                movlw   b'00111101'
                movwf   TMR0            ; Inizializzo il conteggio

CICLO           btfss   PORTA,0         ; Verifica se RA0 = 1
                goto    SPEGNERE        ; se no, spegnere
                goto    ACCENDERE       ; se sì, accendere

SPEGNERE        bcf     PORTB,0
                goto    CICLO

ACCENDERE       bsf     PORTB,0
                goto    CICLO

                END
```

Schema elettrico dell'esercizio proposto.

INTCON	1	0	1	0	0	0	0	0
OPTION	x	x	0	x	0	1	1	1

CALCOLO DEL VALORE DEL TMR0

50ms=50.000µs	Trasformazione in microsecondi.
50.000/256=195,3125=195	Valore che si vuole contare.
11000011	Valore che si vuole contare in binario.
00111100	Complemento a 2 del valore da contare,
00111101	da inserire nel TMR0.

Registri necessari nel programma, con i relativi valori.

sulla realizzazione del programma. Nella figura sono riportati i registri necessari e il valore che assumono.

Sarà necessario configurare l'interrupt del TMR0, abilitando il bit GIE e il T0IE del registro INTCON. Se le temporizzazioni saranno di 50 ms, potremo caricare il divisore di frequenza con un valore di 1:256 e giocare con il valore che deve contare il TMR0, che sarà 195. Ricordate che una cosa è quello che il TMR0 conta, e un'altra è il valore con cui bisogna caricarlo per farlo così contare. Bisogna prestare attenzione agli altri parametri del registro OPTION, per fare in modo che il TMR0 conti come vogliamo.

PROGRAMMA

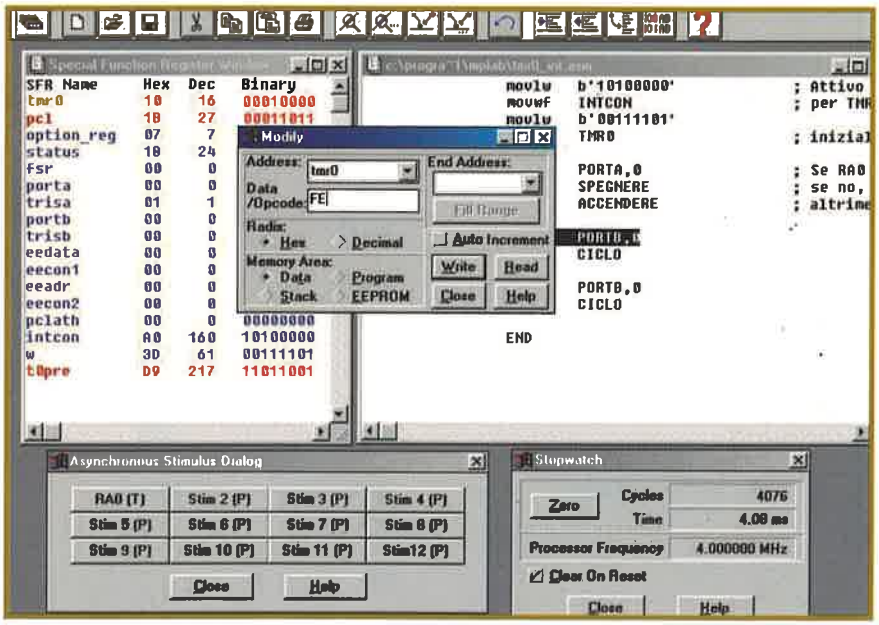
Dopo aver visto i preliminari, passiamo al programma in sé. Definite le variabili, e posizionato il vector di interrupt, iniziamo con la routine dedicata all'interrupt. Si utilizza l'istruzione "xorlw", mettendo un 1 nella posizione adeguata, per invertire il valore del LED collegato a RB1. Prima di rientrare dobbiamo caricare il TMR0 con il valore calcolato in precedenza, per fare in modo che conti nuovamente 50 ms, e porre a zero il flag che indica che il TMR0 è andato in overflow. Nel programma principale, dopo la configurazione dei registri e il carico del TMR0, controlliamo il valore del LED montato su RB0, a seconda dello stato di RA0; il programma rimane all'interno di un ciclo infinito.

SIMULAZIONE DEL PROGRAMMA

Per la simulazione ricorreremo ad un piccolo trucco: aprite le finestre del listato del programma, dei registri specifici, del clock e quella per l'introduzione di stimoli asincroni, quindi configurate la linea RB0 come interruttore. Simulate l'opzione di animazione e verificate che al variare del valore di RA0, lo faccia anche il bit 0 della porta B.

La verifica dell'interrupt invece è più complicata: dopo un po' di tempo la finestra del clock indica appena qualche millisecondo, e come ricordiamo il TMR0 è stato impostato per far variare RB1 ogni 50 ms, che a noi paiono pochissimi, ma per il simulatore significano diversi minuti. Fermate quindi la simulazione e utilizzate la finestra di "modify" per cambiare il valore di TMR0, impostandolo vicino all'overflow. Tornate a simulare il programma, e verificate che di lì a poco RB1 cambi di valore.

Tornate a modificare il TMR0 nello stesso modo se volete far commutare nuovamente RB1, oppure armatevi di pazienza e lasciatelo evolvere da solo.



Per provare il programma dovremo ricorrere ad un piccolo trucco.

Tabella riassuntiva delle istruzioni

Dopo aver visto tutte le istruzioni, il loro funzionamento e il modo di utilizzarle correttamente, presentiamo un riassunto che risulterà molto utile al momento di programmare, dato che raccoglie gli aspetti fondamentali di cui bisogna tenere assolutamente conto.

MNEMONICO	PARAMETRO	SPIEGAZIONE	ATTIVAZIONE DEI FLAGS
Istruzioni che gestiscono registri			
addwf	f, d	SOMMA W con f	C, DC, Z
andwf	f, d	AND di W con f	Z
clrf	f	CANCELLAZIONE di f	Z
clrw		CANCELLAZIONE di W	Z
comf	f, d	COMPLEMENTO di f	Z
decf	f, d	DECREMENTO di f	Z
incf	f, d	INCREMENTO di f	Z
iorwf	f, d	OR di W con f	Z
movf	f, d	SPOSTAMENTO di f	Z
movwf	f	SPOSTAMENTO da W a f	
rlf	f, d	ROTAZIONE a sinistra con carry	C
rrf	f, d	ROTAZIONE a destra con carry	C
subwf	f, d	SOTTRAE W da f (f-W)	C, DC, Z
swapf	f, d	INTERSCAMBIO dei 4 bit + significativi con i 4 - significativi	
xorwf	f, d	OR esclusivo di W con f	Z
Istruzioni che gestiscono i bit			
bcf	f, b	IMPOSTA a 0 il bit b di f	
bcf	f, b	IMPOSTA a 1 il bit b di f	
Istruzioni che gestiscono operandi immediati			
addlw	k	SOMMA di letterale con W	C, DC, Z
andlw	k	AND di letterale con W	Z
iorlw	k	OR di letterale con W	Z
movlw	k	SPOSTAMENTO di letterale a W	
sublw	k	SOTTRAE W da letterale (K-W)	C, DC, Z
xorlw	k	OR esclusivo di letterale con W	Z
Istruzioni di salto			
btfsc	f, b	TESTA bit b di f; SALTA se 0	
btfss	f, b	TESTA bit b di f; SALTA se 1	
decfsz	f, d	DECREMENTA f;	
		SALTA se 0	
incfsz	f, d	INCREMENTA f;	
		SALTA se 1	
Istruzioni di controllo e speciali			
call	k	CHIAMATA a subroutine	
clrwdt		CANCELLAZIONE del WATCHDOG	TO#, PD#
goto	k	SALTO ad un indirizzo	
nop		NO OPERATION	
retfie		RITORNO da interrupt	
retlw	k	RITORNO lasciando un letterale in W	
return		RITORNO da subroutine	
sleep		IMPOSTA il microprocessore in standby o stato di riposo	TO#, PD#

Istruzioni del PIC16F84.

PROGRAMMAZIONE

Tabella riassuntiva dei registri della memoria RAM

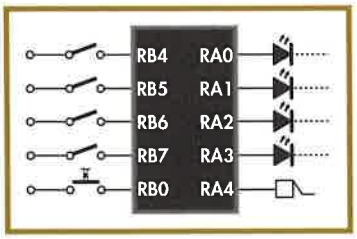
Al momento di realizzare un particolare programma, i registri sono tanto importanti quanto le istruzioni. Nella tabella pubblicata in questa pagina sono riassunti tutti i registri specifici della memoria RAM, con la situazione di ognuno dei loro bit.

IND.	NOME	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
		BANCO 0							
00h	INDF	Utilizza il contenuto di FSR per indirizzare la memoria (non è un registro fisico)							
01h	TMR0	Clock / Contatore in tempo reale a 8 bit							
02h	PCL	Byte meno significativo del PC							
03h	STATO	IRP	RP1	RP0	TO#	PD#	Z	DC	C
04h	FSR	Indirizzamento indiretto con INDF							
05h	PORTA A	—	—	—	RA4/T0CK	RA3	RA2	RA1	RA0
06h	PORTA B	RB7	RB 6	RB 5	RB 4	RB 3	RB 2	RB 1	RB0/INT
07h		Non utilizzato. Si legge come "0"							
08h	EEDATA	Registro dei dati per la EEPROM							
09h	EEADR	Registro degli indirizzi per la EEPROM							
0Ah	PCLATH	—	—	—	Si scrive nei 5 bit di PCH				
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
		BANCO 1							
80h	INDF	Utilizza il contenuto di FSR per indirizzare la memoria (non è un registro fisico)							
81h	OPTION	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	PCL	Byte meno significativo del PC							
83h	STATO	IRP	RP1	RP0	TO#	PD#	Z	DC	C
84h	FSR	Indirizzamento indiretto con INDF							
85h	TRISA	—	—	—	Configurazione Porta A				
86h	TRISB	Configurazione Porta B							
87h		Non implementato. Si legge come "0"							
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD
89h	EECON2	Registro di controllo EEPROM (non è un registro fisico)							
8Ah	PCLATH	—	—	—	Cinque bit di PCH				
8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Registri della memoria RAM.

Tutti gli interrupt insieme

Stiamo arrivando al punto culminante della programmazione con i microcontroller, mancano solo alcune piccole risorse per conoscerne tutti i suoi segreti. In questa occasione combineremo tutti gli interrupt in un unico programma. Le applicazioni che stiamo facendo sono sempre più complicate e professionali. Immaginate il seguente esercizio implementato.



Schema elettrico dell'esercizio proposto.

**Si vuole simulare il funzionamento dell'allarme di una macchina.
Si utilizzeranno quattro sensori collegati ognuno ad una porta, un cicalino, quattro LED e un pulsante. Aprendo una delle porte si attiverà un cicalino a intervalli regolari, sino a che non venga premuto il pulsante di stop. Inoltre si accenderà un LED corrispondente alla porta aperta e si scriverà questa informazione sulla EEPROM. Tutto il programma verrà fatto utilizzando gli interrupt.**

Enunciato del programma proposto.

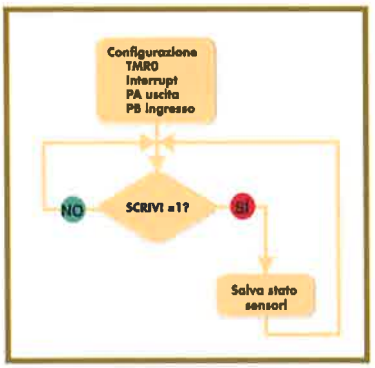
SCHEMA ELETTRICO

Anche lo schema elettrico si sta complicando progressivamente, in questo esercizio si utilizza la quasi totalità della Porta A e la maggior parte delle linee della porta B. La loro scelta, come mostra la figura, non è casuale, infatti, sugli ingressi da RB4 a RB7 dovranno essere collegati i sensori delle porte per generare interrupt; su RB0 invece collegheremo il pulsante di stop dell'allarme. I pin da RA0 a RA3 si utilizzano per mostrare lo stato dei sensori delle porte: RA0 mostrerà quello di RB4, RA1 quello di RB5 e così via. Questa scelta, anche se non è obbligatoria, faciliterà la programmazione. Collegando il cicalino su RA4, potremo configurare la porta A come uscita, e la porta B come ingresso, facilitano così ulteriormente le cose. Dovranno essere collegate anche le linee di alimentazione, il cristallo di quarzo e il reset.

ORGANIGRAMMA DEL PROGRAMMA PRINCIPALE

L'organigramma principale, come si può verificare nella figura a lato, è molto semplice, come accade quasi sempre quando si lavora con interrupt, dato che il peso del programma ricade su questi ultimi. L'unico interrupt che non è attivato dall'inizio, è il TMR0. La traduzione dell'organigramma in assembler sarà immediata. Uno degli interrupt porrà a 1 la variabile "scrivi" quando cambia il valore dello stato dei sensori, indicando che dobbiamo scrivere un nuovo valore nella EEPROM, quindi ci sarà una chiamata ad una subroutine che realizza questa funzione. In ogni caso per risolvere questo esercizio, bisognerà pensare molto bene a quale sarà il compito di ognuno degli interrupt.

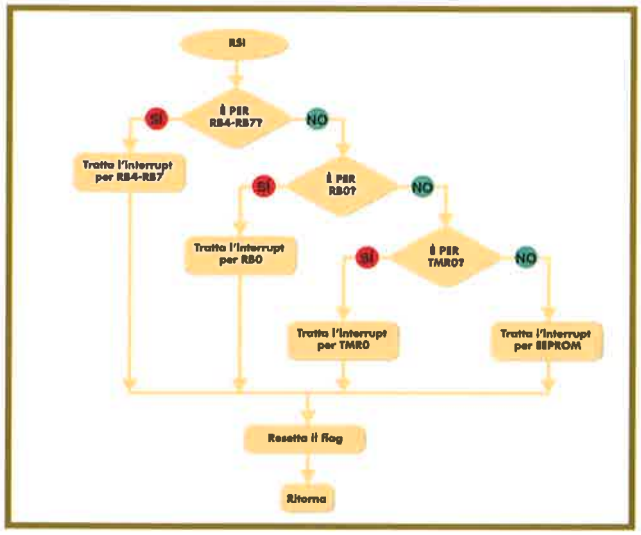
L'organigramma che rappresenta la routine di servizio all'interrupt, potrà servire da base per qualsiasi programma che lavori con più di un interrupt, dato che è molto generale. Se si lavora con solo due interrupt, potremo sopprimere le richieste corrispondenti a quelli che non sono utilizzati, e il resto sarà invariato. È importante rendersi conto che ogni interrupt è indipendente dagli altri, e che quando se ne produce uno non se ne possono produrre altri sino a che non sia terminato il trattamento del primo. Saranno le azioni corrispondenti ad ogni interrupt, a cambiare da un'applicazione ad un'altra. Vediamo l'organigramma corrispondente ad ogni interrupt per il nostro esercizio.



Organigramma del programma principale dell'esercizio proposto.

INTERRUPT PER EEPROM

Inizieremo dall'interrupt per fine di scrittura nella EEPROM, perché in questo caso è il più semplice. Quando si termina di scrivere il valore dello stato delle



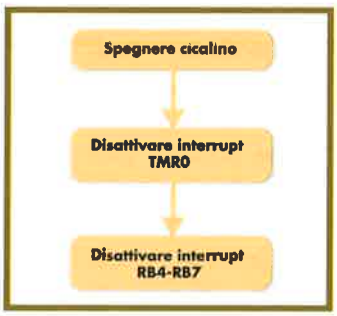
Organigramma della routine di interrupt.

porte della macchina, aperte o chiuse, dobbiamo solo ritornare a impostare il valore della variabile "scrivi" a 0, per fare in modo che non avvenga una nuova scrittura del valore sino a che quest'ultimo non cambia.

INTERRUPT PER TMR0

Per quanto riguarda il caso del TMR0, è già stato presentato un esempio simile. Se vogliamo che il cicalino si accenda e si spenga ad intervalli regolari, utilizzeremo il TMR0 per controllare questo tempo, e al prodursi dell'interrupt invertiremo il valore del cicalino. Prima di tornare al programma principale il TMR0 dovrà essere inizializzato nuovamente, in modo che, quando andrà in overflow, cambi un'altra volta il valore del cicalino.

Organigramma del trattamento dell'interrupt per EEPROM.



Organigramma del trattamento dell'interrupt per RB0.

INTERRUPT PER RB0

Se l'utilizzatore attiva il pulsante collegato su RB0, significa che vuole fermare l'allarme perché, ad esempio, è stato lui a provocare l'allarme, quindi spegnerà il cicalino. Non dobbiamo dimenticare che, se non facciamo nulla, il TMR0 continuerà a contare e ad andare in overflow, producendo interrupt e tornando ad accendere e spegnere il cicalino. Dovremo quindi disattivare gli interrupt per TMR0, per fare in modo che non se ne producano più. Faremo la stessa cosa per gli interrupt su RB4-RB7 per i sensori collegati alle porte. L'allarme sarà disattivato sino a che non si provochi un reset del sistema.

Organigramma per il trattamento dell'interrupt per TMR0.

Organigramma del trattamento dell'interrupt per RB4-RB7.

INTERRUPT PER RB4-RB7

Questo è l'interrupt più complicato del nostro esempio, dato che è quello che deve tener conto di più cose. La prima sarà di mostrare mediante i LED il valore delle porte: aperte o chiuse. Per fare in modo che il TMR0 possa cambiare il valore del cicalino lo dovremo inizializzare, e attivarlo su interrupt. Ponendo la variabile "scrivi" a 1, si otterrà che nel programma principale venga attivata la scrittura del nuovo valore delle porte nella EEPROM. Nel caso non venga premuto RB0, qualsiasi cambio si produca sulle porte, sia in apertura che in chiusura, tornerà a far saltare all'interrupt, e a scrivere un nuovo valore nella EEPROM.

PROGRAMMA

Il programma che risolve il nostro esercizio è una ricompilazione degli organigrammi commentati, una volta passati in assembler. Dopo averlo risolto ci sembrerà la più logica delle soluzioni.

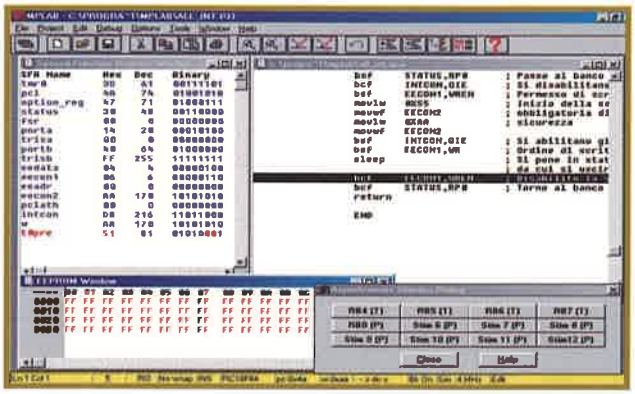
```
                LIST        P=16F84          ; Si utilizza il PIC16F84
                            RADIX HEX        ; Sistema di numerazione esadecimale

WR              EQU         1
WREN            EQU         2
EEIF            EQU         4
GIE             EQU         7

RBIF            EQU         0
INTF            EQU         1
TOIF            EQU         2
RBIE            EQU         3
TOIE            EQU         5

W               EQU         0
F               EQU         1
RP0             EQU         5

TMR0            EQU         0X01
STATUS          EQU         0X03
PORTA           EQU         0X05
PORTB           EQU         0X06
EEDATA          EQU         0X08
EEADR           EQU         0X09
INTCON          EQU         0X0B
OPTION_REG      EQU         0X81
TRISA           EQU         0X85
TRISB           EQU         0X86
EECON1          EQU         0X88
EECON2          EQU         0X89

EEADR_VAR       EQU         0X0C
EEDATA_VAR      EQU         0X0D
SCRIVI          EQU         0X0E

                ORG         0                ; Il programma inizia al-
                goto        INIZIO           ; l'indirizzo 0
                ORG         4                ; Vector di interrupt
                goto        INT

;Programma di trattamento dell'interrupt
INT             btfsc       INTCON,RBIF
                goto        T_RB4_RB7
                btfsc       INTCON,INTF
                goto        T_RB0
                btfsc       INTCON,TOIF
                goto        T_TMR0
                goto        T_EEPROM

T_RB4_RB7       swapf       PORTB,W          ; Il valore dei bit più significativi passa
                                             ; a quelli meno significativi
                andlw       b'00001111'      ; Si cancella il valore dei bit meno
                                             ; significativi
                movwf       PORTA            ; Si passa il valore degli allarmi alla PORTA A
                movlw       b'00111101'      ; Si carica di nuovo il TMR0
                movwf       TMR0
                bsf         INTCON,TOIE      ; Si attiva l'interrupt del TMR0
                bsf         SCRIVI,0         ; Si imposta SCRIVI a 1
                bsf         PORTA,4          ; Si attiva il cicalino collegato a RA4
                goto        TORNARE

T_RB0           bcf         PORTA,4
                bcf         INTCON,TOIE
                bcf         INTCON,RBIE
                goto        TORNARE

T_TMR0          movf        PORTA,W
                xorlw       b'00010000'      ; Si inverte il valore del cicalino
                movwf       PORTA
                movlw       b'00111101'      ; Si carica di nuovo il TMR0
                movwf       TMR0
                goto        TORNARE

T_EEPROM        bcf         SCRIVI,0
                goto        TORNARE
```

Soluzione del programma proposto.

SIMULAZIONE DEL PROGRAMMA

Nella simulazione, come sempre, dovremo aprire una serie di finestre per vedere come viene eseguito il programma. Utilizzeremo l'opzione di animazione per poter simulare l'apertura di una porta e vedere come viene mostrata l'informazione istantaneamente sulla porta A, oltre all'attivazione del cicalino.

Per verificare la scrittura dell'informazione nella EEPROM, dovremo attendere più tempo, così come per vedere cambiare valore al cicalino. Quando vorremo, potremo premere RB0 e osservare come l'allarme cessi di essere operativo, anche se continueremo ad aprire e a chiudere le porte.



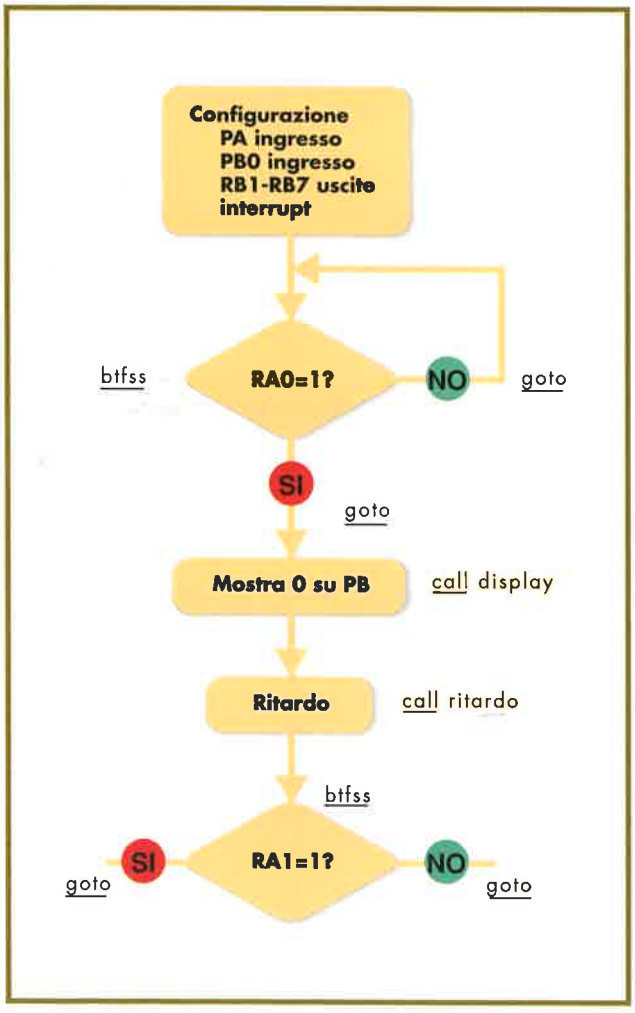
Un momento della simulazione.

```
TORNARE     movf    INTCON,W
            andlw   b'11110000'
            movwf   INTCON
            bsf     STATUS,RP0      ; Passo al banco 1
            bcf     EECON1,EEIF     ; Resetto il flag
            bcf     STATUS,RP0      ; Passo al banco 0
            retfie

INIZIO      bsf     STATUS,RP0      ; Passo al banco 1
            movlw   0XFF            ; Porta B ingresso
            movwf   TRISB
            movlw   00              ; Porta A uscita
            movwf   TRISA
            movlw   b'01000111'     ; Configurazione TMR0
            movwf   OPTION_REG      ; e fronte di interrupt

            movlw   b'11011000'     ; Attivazione degli interrupt
            movwf   INTCON          ; per RB4-RB7

            bcf     STATUS,RP0      ; Cambio al banco 0

CICLO       btfss   SCRIVI,0        ; Se SCRIVI=1
            goto    CICLO           ; se no, niente
            goto    SALVARE         ; altrimenti salva lo stato dei sensori

SALVARE     movlw   00              ; Porto l'indirizzo da scrivere
            movwf   EEADR_VAR       ; a una variabile ausiliaria
            swapf   PORTB,W         ; Si sposta il valore da scrivere, che sono
                                    ; i bit più significativi
            andlw   b'00001111'     ; Si cancella il valore dei bit più
                                    ; significativi
            movwf   EEDATA_VAR      ; ad una variabile ausiliaria
            call    SCRIVI_EEPROM   ; Si chiama la routine
            goto    CICLO

SCRIVI_EEPROM movf  EEADR_VAR,W     ; Si sposta l'indirizzo
            movwf   EEADR           ; al suo registro
            movf    EEDATA_VAR,w    ; Si sposta il dato
            movwf   EEDATA          ; al suo registro
            bsf     STATUS,RP0      ; Passo al banco 1
            bcf     INTCON,GIE      ; Si disabilitano gli interrupt
            bsf     EECON1,WREN     ; Permesso di scrittura
            movlw   0X55            ; Inizio della sequenza
            movwf   EECON2          ; obbligatoria di
            movlw   0XAA            ; sicurezza
            movwf   EECON2
            bsf     INTCON,GIE      ; Si abilitano gli interrupt
            bsf     EECON1,WR       ; Ordine di scrittura
            sleep                   ; Si pone in stato di riposo
                                    ; da cui si uscirà con un interrupt
            bcf     EECON1,WREN     ; Disabilito la scrittura
            bcf     STATUS,RP0      ; Torno al banco 0
            return

            END
```

Soluzione del programma proposto (continuazione).

Gli organigrammi delle subroutines e degli interrupt

Ogni volta che si pianifica un nuovo programma, la prima cosa da fare è realizzare un nuovo programma. Alcune volte è stato fatto quello generale, in modo più o meno dettagliato, altre volte quello delle routines di servizio all'interrupt. Però non è mai stato specificato nulla per quanto riguarda la realizzazione congiunta, né sono stati fatti gli organigrammi delle subroutines. Ora che abbiamo l'esperienza di diversi esempi, conosciamo tutte le istruzioni e la struttura della programmazione con il PIC, possiamo arrivare sino in fondo a questo utile strumento che sono gli organigrammi.



Traduzione automatica di un organigramma in codice assembler.

COSA NASCONDE UN ORGANIGRAMMA

Nella sua presentazione abbiamo già visto che un organigramma è formato da rettangoli e rombi uniti da frecce, e che queste devono seguire delle norme perché il tutto risulti sintatticamente corretto.

Questo però non basta, perché a partire dall'organigramma principale, da quello delle subroutines e da quello degli interrupt si dovrà poter passare al codice assembler quasi automaticamente. Questo tipo di correzione sarà la semantica.

Ogni rettangolo si trasformerà in diverse istruzioni assembler o in una sola istruzione "call", da cui deriverà una subroutine.

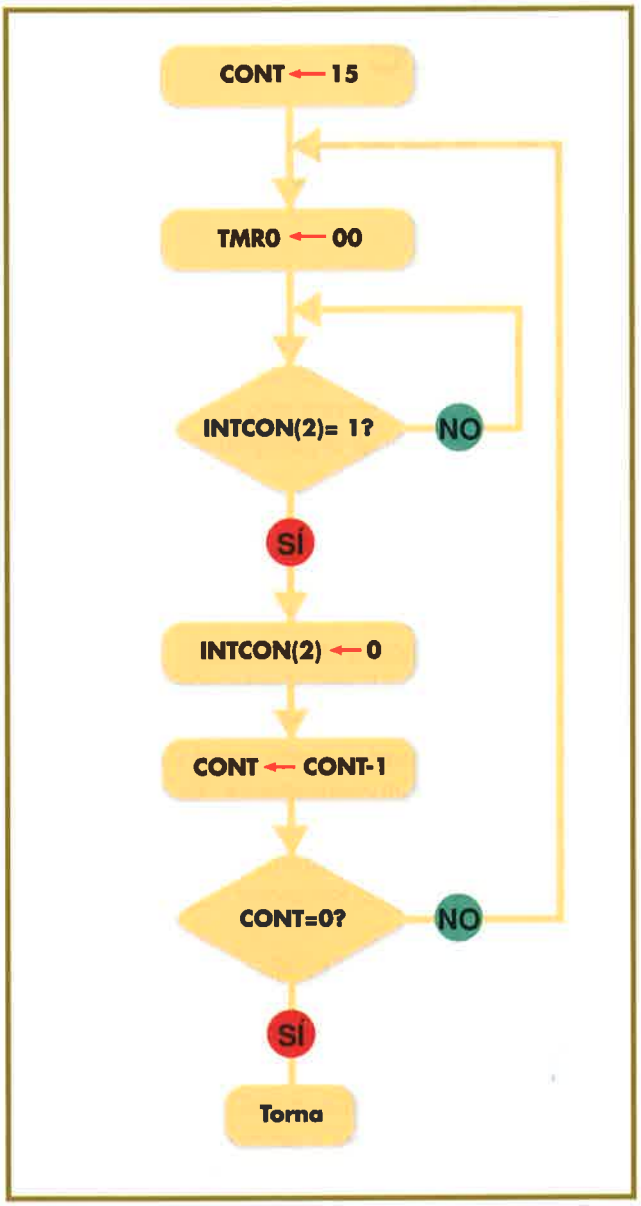
La scelta fra una cosa e l'altra dipenderà dal livello di complessità del programma, e dalla ripetitività o meno delle azioni da realizzare. Nel nostro esempio, mostrare un numero su un display o realizzare un ritardo sono sequenze molto utilizzate, quindi l'uso di subroutine è adeguato per presentare il programma in modo leggibile, e per risparmiare codice, evitando la ripetizione delle istruzioni.

I rombi a loro volta si trasformano in istruzioni di salto condizionale: btfss, btfsc, incfsz, ecc. dopo le quali si posizioneranno due istruzioni "goto", una per ogni ramo del rombo. La presenza degli interrupt è dichiarata nel primo rettangolo, però in nessuna parte dell'organigramma troveremo la chiamata alla routine di servizio dell'interrupt: ricordate che un interrupt non si chiama.

L'ORGANIGRAMMA DELLE SUBROUTINES

A prima vista, l'organigramma principale e quello della subroutine sono simili.

Si differenziano solo nel primo e nell'ultimo rettangolo.



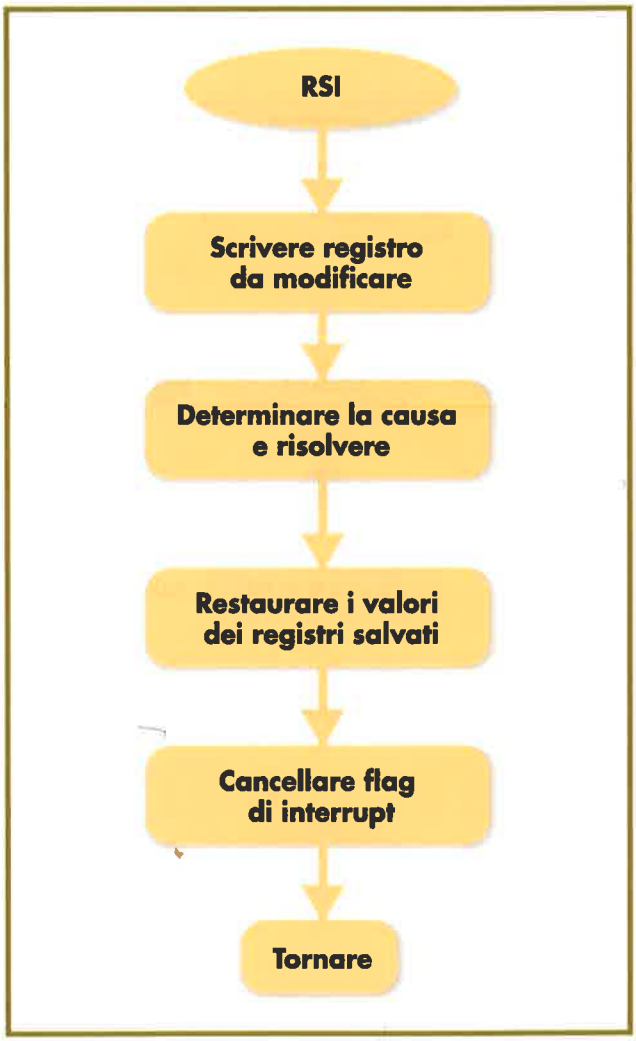
Esempio di un organigramma di una subroutine.

Nella subroutine normalmente non è necessario configurare i registri e le altre risorse, perché questo è già stato fatto nel programma principale; a volte però, quando si utilizzano le linee di ingresso/uscita per molte periferiche, è necessario riconfigurare le linee nella medesima subroutine. L'ultimo riquadro di una subroutine indicherà il ritorno al programma principale. Sopra l'organigramma si riporterà il nome della subroutine, così come viene chiamata dal programma principale.

Il resto delle norme, come terminare in un unico riquadro, o che i rombi abbiano due rami, sono le medesime dell'organigramma principale. Possono essere utilizzati sia gli organigrammi che le subroutines, anche se non sempre sono necessarie.

L'ORGANIGRAMMA DEGLI INTERRUPT

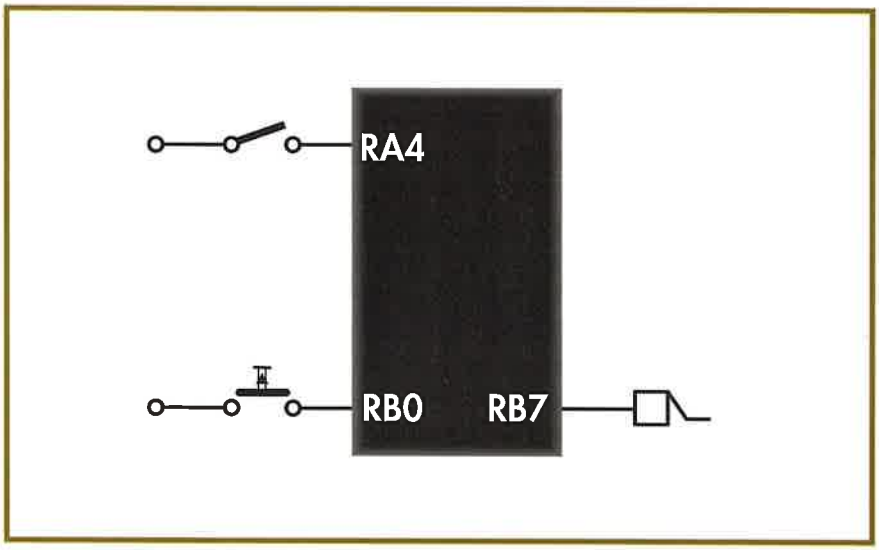
Così come c'è un solo organigramma principale, c'è una sola routine di servizio per gli interrupt. Se lavoriamo con più di un interrupt, la prima cosa da fare è verificare quale ne sia stata la causa, come abbiamo già visto in precedenza, testando ad uno ad uno i suoi flag sino a trovarla. Ogni causa anche se non era di per sé una subroutine, generava un organigramma che ne spiegava il funzionamento. A volte è necessario anche realizzare delle operazioni in più, come scrivere i valori di alcuni registri prima di risolvere l'interrupt, i quali potrebbero essere modificati nel corso dell'interrupt stesso.



Organigramma generale della routine dedicata all'interrupt.

Applicazione pratica: uscita dal modo di riposo

Abbiamo già dovuto ricorrere a questa preziosa risorsa (il modo di riposo) quando abbiamo lavorato con gli interrupt, come modo migliore per lavorare con essi. Bastava porre l'istruzione "sleep" per entrarci e avevamo diverse forme per uscirne. Si utilizzava nei casi in cui il processore doveva rimanere inattivo durante un periodo indeterminato di tempo. In questo caso ci occuperemo dei due bit situati nel registro di stato che serviranno per determinare qual è stata la causa che ha portato il processore in questo letargo.



Schema elettrico dell'esercizio proposto.

Si vuole realizzare un processo industriale che consiste nel contare gruppi di 10 pezzi.
Ogni pezzo attiva un sensore al suo passaggio.
Quando il gruppo è completo si ferma la macchina, fino a che non si preme un pulsante.
Se va in overflow il Watchdog, si attiverà un cicalino, ad indicare un'anomalia nel sistema.

Enunciato dell'esercizio proposto.

SCHEMA ELETTRICO

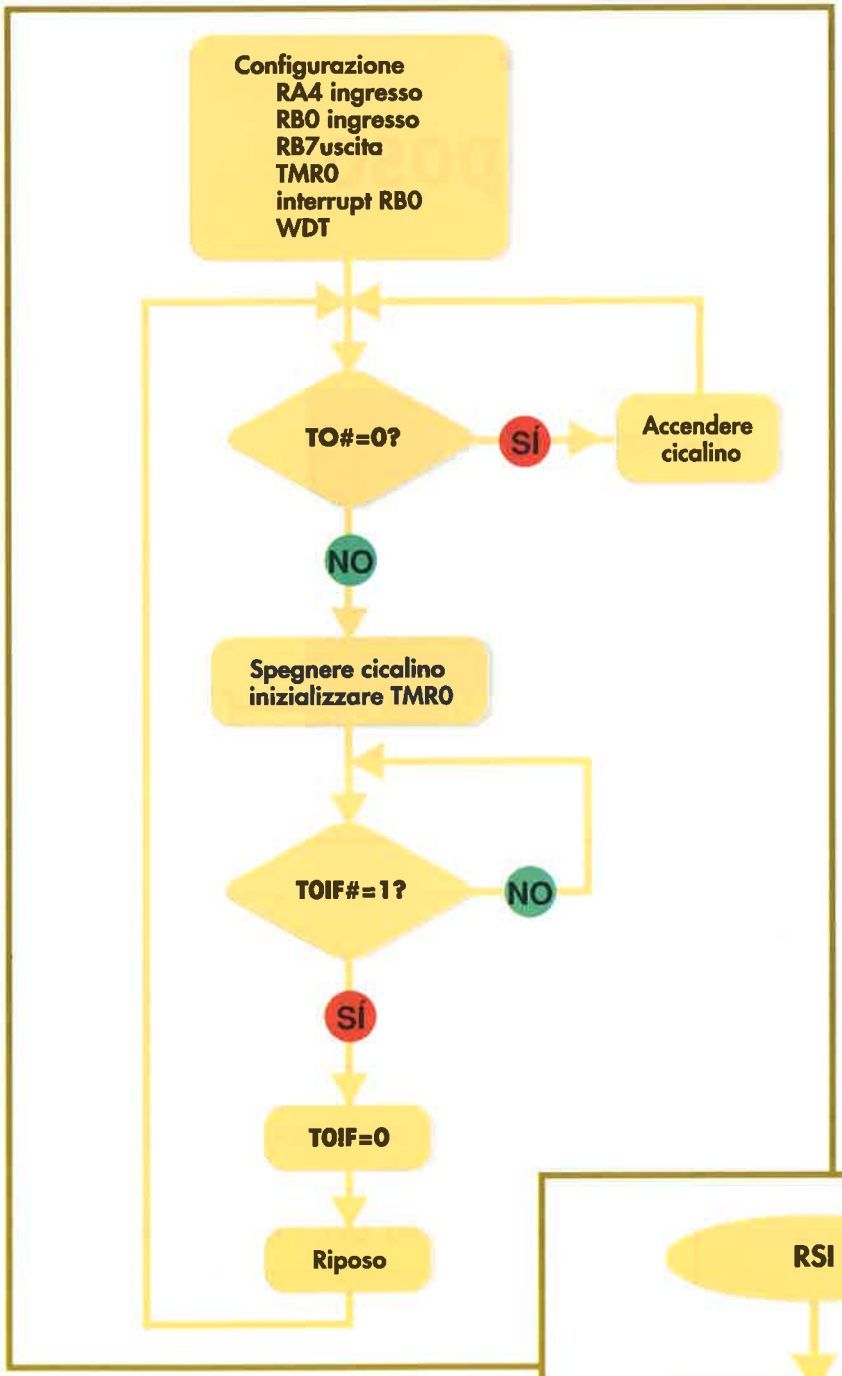
L'unica cosa chiara per il momento è il numero di ingressi e di uscite che sono necessarie. Un sensore dovrà contare il numero di pezzi, ricordiamo che il modo migliore per farlo è utilizzando il TMR0, per cui gli impulsi dovranno entrare tramite RA4. Inoltre avremo un pulsante per mettere in marcia il sistema ogni volta che si completa un gruppo di pezzi. Anche se non sappiamo ancora quale interrupt utilizzeremo, possiamo comunque prevedere RB0 assegnato a questo pulsante. Il cicalino utilizzerà un'altra linea, questa volta di uscita. Le linee di alimentazione, il quarzo e il reset si dovranno tenere in conto nel montaggio finale.

ORGANIGRAMMI DEL PROGRAMMA

Prima di fare l'organigramma dobbiamo leggere con attenzione l'enunciato e pensare a quello che già conosciamo. Anche se ora sappiamo lavorare con gli interrupt, non è detto che debbano essere utilizzati in tutti i programmi, né che si debba usare l'istruzione "sleep" ogni volta che c'è un interrupt. In questo caso, ad esempio, abbiamo pensato che non c'è ragione per implementare l'interrupt del TRM0, perché non ha ragione d'essere, inoltre dovremmo attendere all'interno di un ciclo che questo si produca: ricordate che non possiamo mettere in stato di riposo, perché il TRM0 smetterebbe di funzionare. Dopo ogni gruppo di dieci pezzi il sistema deve rimanere fermo sino a che viene

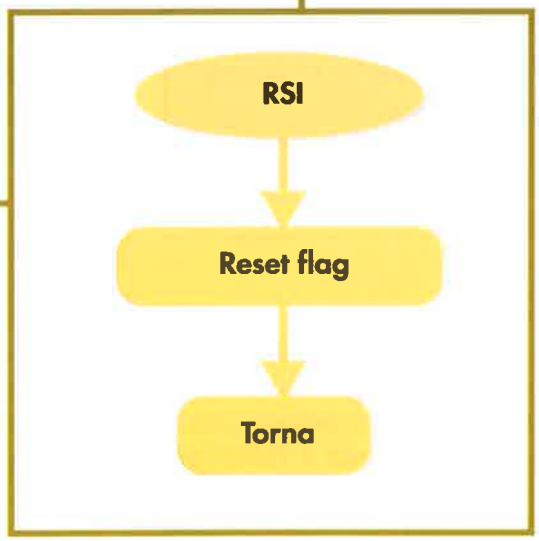
PROGRAMMAZIONE

Software



Organigramma del programma principale dell'esercizio proposto.

premuto un pulsante. Qui sì che possiamo attendere con una "sleep", dato che l'interrupt tramite RB0 ci toglierà da questo stato. Nell'enunciato, inoltre, si dice che dobbiamo indicare con l'attivazione di un cicalino le anomalie del sistema, anomalie che saranno rivelate quando il Watchdog andrà in overflow. Se all'inizio del programma si verificherà il valore del bit TO#, ed esso corrisponde a 1, sapremo che si è arrivati a questo punto per un overflow del Watchdog e non per l'evoluzione normale del programma. Nel nostro caso capiterà quando si tarda troppo tempo a contare il gruppo di 10 pezzi, oppure quando si tarda troppo a premere il pulsante dopo aver terminato con tale gruppo. Una volta attivato, il cicalino non si fermerà sino a che non si inizializzerà il sistema, e questo farà cambiare il bit TO#. Per quanto riguarda l'interrupt di RB0, serve unicamente per fare in modo che il sistema esca dallo stato di riposo, quindi il suo organigramma contiene solo le istruzioni principali. Visto che configuriamo solo questo interrupt, quando si produce, non dobbiamo ricercarne la causa.



Organigramma della routine di interrupt.

PROGRAMMA

Il programma si risolverà come riportato nella figura. La configurazione rappresentata nel primo quadro dell'organigramma si risolve con i corrispondenti valori in TRISA e TRISB per le porte di ingresso e uscita, e con i valori di registri OPTION e INTCON. Nel primo, il divisore di frequenza si configura al massimo, non per il TMR0, ma per il Watchdog. Si specifica, inoltre, che il TMR0 si incrementerà con gli impulsi esterni che arrivano tramite RA4, e che il fronte attivo sarà il fronte di discesa, cioè quando il pezzo ha

```
                LIST        P=16F84         ; Si utilizza il PIC16F84
                RADIX HEX                   ; Sistema di numerazione esadecimale

TMR0            EQU         0X01
T0IF            EQU         0X02
TO              EQU         0X04
STATUS          EQU         0X03
PORTB           EQU         0X06
RP0             EQU         0X05
INTCON          EQU         0X0B
OPTION_REG      EQU         0X81
TRISA           EQU         0X85
TRISB           EQU         0X86
INTF            EQU         0X01

                ORG         0           ; Il programma inizia
                goto        INIZIO      ; all'indirizzo 0
                ORG         4           ; Vector di interrupt
                goto        INTER
                ORG         5

INIZIO          bsf         STATUS,RP0  ; Si passa al banco 1
                movlw       b'00000001' ; RB0 ingresso, il resto uscita
                movwf       TRISB
                movlw       b'00010000' ; RA4 ingresso, il resto uscita
                movwf       TRISA
                movlw       b'11111000' ; Configurazione di TMR0 e WDT
                movwf       OPTION_REG
                movlw       b'10010000' ; Attivazione dell'interrupt
                movwf       INTCON
                bcf         STATUS,RP0  ; Si passa al banco 0

ALLARME         btfsc       STATUS,TO   ; Verifico il valore di TO#
                goto        NORMALE     ; Sequenza normale
                goto        ATTIVARE    ; Il WDT è andato in overflow

NORMALE         bcf         PORTB,7     ; Disattivo il cicalino
                movlw       0xF6        ; Carico il complemento a 2
                movwf       TMR0        ; del valore da contare

CICLO           btfss       INTCON,T0IF ; Sono passati 10 pezzi?
                goto        CICLO       ; No, aspetta
                bcf         INTCON,T0IF ; Si, resetta il flag
                sleep                   ; e metti in riposo

                goto        ALLARME

ATTIVARE        bsf         PORTB,7     ; Accendi il cicalino
                goto        ALLARME     ; Entra in un ciclo

INTER           bcf         INTCON,INTF ; Resetta il flag
                retfie                  ; Torna dall'interrupt

                END
```

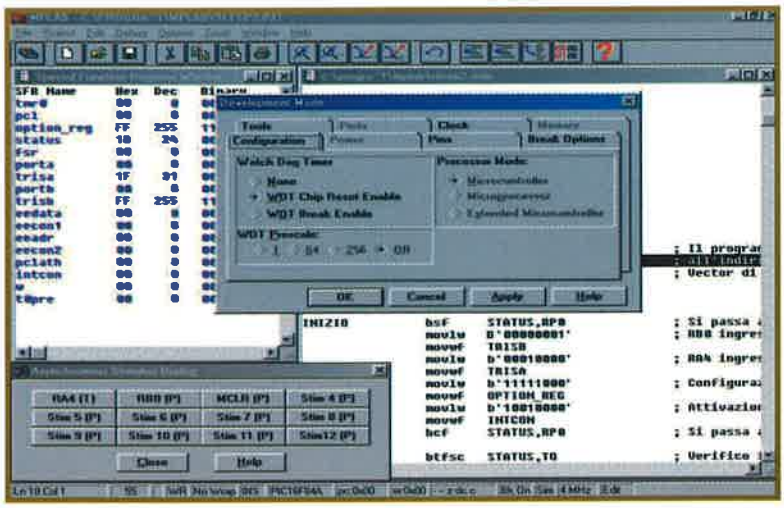
Soluzione del programma proposto.

terminato di passare davanti al sensore. L'interrupt per RB0 si configura a sua volta con il fronte di discesa, e si disattiveranno le resistenze di pull-up. Il resto del programma è una traduzione quasi immediata degli organigrammi.

SIMULAZIONE DEL PROGRAMMA

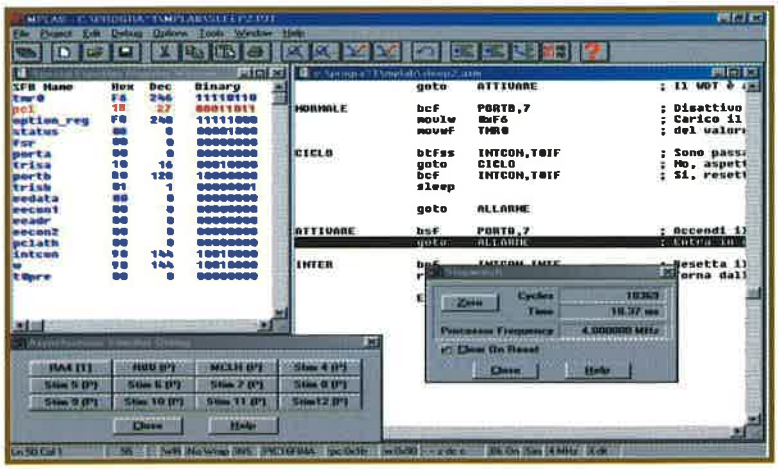
Per realizzare la simulazione in questo caso dobbiamo aprire le finestre degli stimoli asincroni, per simulare il passaggio dei pezzi e il pulsante di messa in marcia del sistema; notate che abbiamo aggiunto anche la linea di reset del microcontroller, per poter osservare una cosa molto particolare che commenteremo più avanti. Tramite la finestra dei registri faremo attenzione al valore che prenderanno i bit TO# e PD#, oltre all'evoluzione del TMR0 e all'attivazione del cicalino. Non dimenticate di assicurarvi che il Watchdog sia attivato.

Eseguite il programma con l'opzione di animazio-

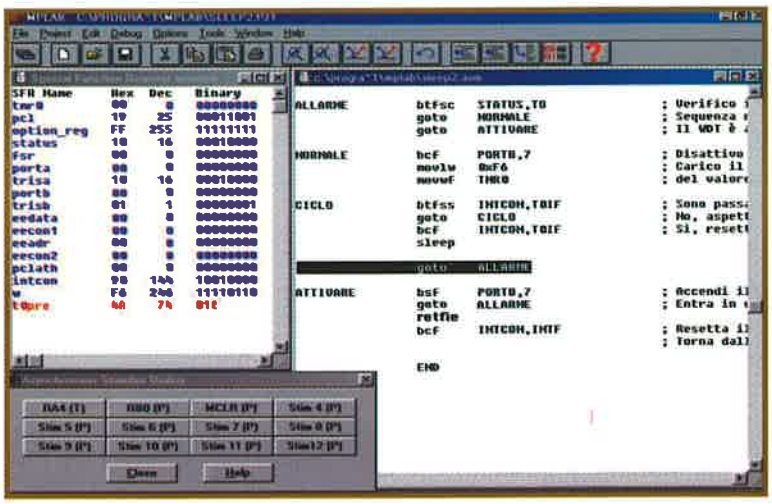


Preparazione delle finestre per la simulazione.

dovremo, come già altre volte, aspettare troppo tempo, trattandosi di una simulazione. Per questo utilizzeremo un trucco: cambieremo il valore del divisore di frequenza, mettendo in OPTION il valore '11111000'. Dovremo nuovamente assemblare il programma. Eseguire un'altra volta l'applicazione, però in modo rapido (Run) e questa volta non cliccare il pulsante RA4. Il Watchdog tarderà l'apprezzabile cifra di 18 ms ad andare in overflow. Prenderemo la finestra di clock e fermeremo la simulazione, nuovamente, per vedere i risultati. Il programma deve rimanere nel primo ciclo, e se si manda di nuovo in esecuzione con l'animazione vedremo che di lì non esce. TO# (bit 4 del registro di STATO) ne. Dopo la configurazione delle risorse, il programma entrerà in un ciclo aspettando che passino i 10 pezzi. Non uscirà da questo ciclo fino a quando non faremo 10 doppi clic sul pulsante RA4. Se si configura la linea come pulsante, invece di interruttore, basteranno la metà delle pulsazioni. Osservate che ogni volta che si realizza una doppia pulsazione si incrementa di 1 il TMR0. Si arriva così allo stato di riposo, momento in cui cambia il valore del bit PD# (bit 3 del registro di STATO) passando a 0. All'inizio del programma era a 1. Dopo breve tempo, cliccate RB0, vedrete che il programma inizia di nuovo. Se ora vogliamo verificare l'overflow del Watchdog,



Programma in stato di allarme, dopo che è andato in overflow il Watchdog.



Programma in attesa che sia premuto RB0.

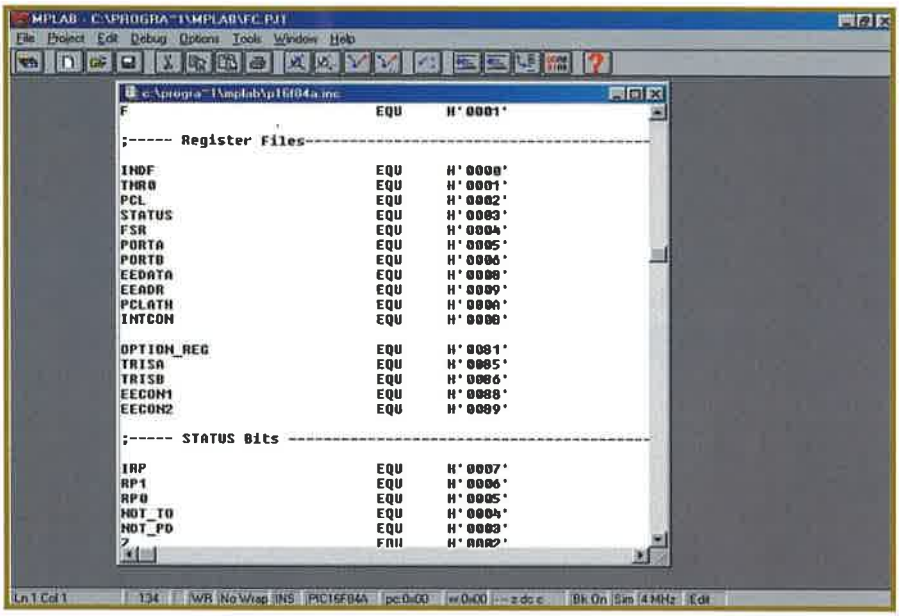
dovrà avere valore 0 per indicare che è stato prodotto un nuovo overflow, e il cicalino essere attivato. Pulseremo MCLR per resettare il sistema e vedremo che il programma prosegue rifacendo la stessa cosa. Questo perché si verifica all'inizio il valore del bit TO#, ed esso non cambierà fino a che non si scollega l'alimentazione: non basta pulsare il reset. Nella simulazione, scollegare l'alimentazione equivale a scegliere l'opzione Debug>System Reset. Iniziare una nuova esecuzione, introdurre dieci pezzi però senza premere il pulsante RB0. In circa 18 ms dovrebbe ripetersi la stessa cosa.

File di intestazione

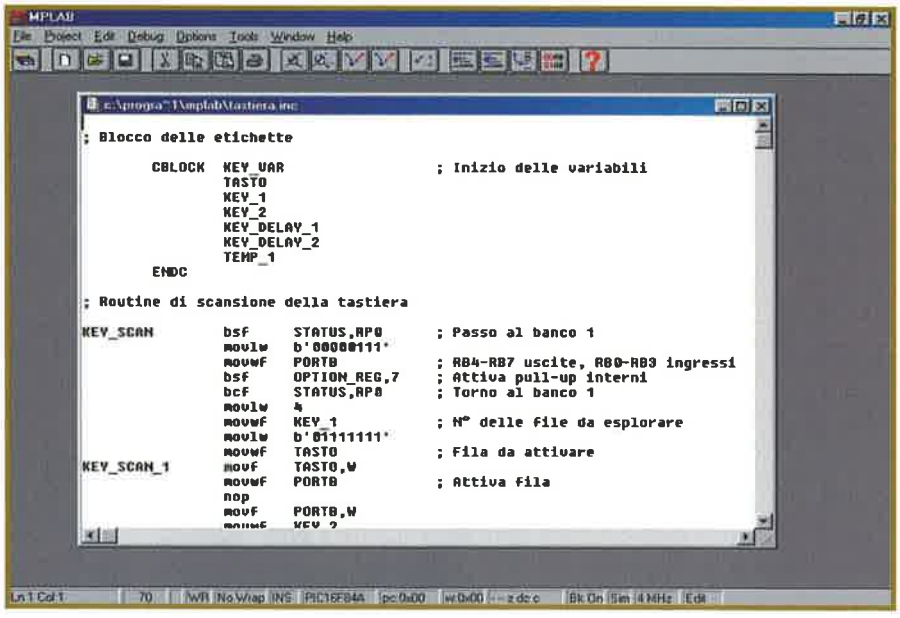
I file di intestazione sono utilizzati in forma di moduli da includere all'interno di un programma principale. I file di intestazione standard che fornisce Microchip definiscono i registri specifici di ogni microcontroller particolare, a seconda del file. Così, ad esempio, nel caso del PIC16F84, il suo file di intestazione è il p16F84.inc o il p16F84a.inc, a seconda del modello.

ALCUNI FILE DI INTESTAZIONE

Nel p16F84.inc, così come è mostrato nella figura, appaiono le definizioni sia dei registri specifici, che dei bit particolari di questi registri. Si può vedere il contenuto completo aprendo il registro tramite MPLAB, come qualsiasi altro file. L'uso dei file di intestazione ha il vantaggio di risparmiarci il lavoro per definire le variabili più comuni ogni volta che si fa un programma, inoltre migliora la chiarezza nella lettura. Con l'uso di file di intestazione standard, si facilita la comprensione da parte di un programmatore esterno, permettendogli pertanto di collaborare in un programma non scritto da lui. Però i file di intestazione, oltre alla definizione delle variabili, possono contenere subroutine o macro per il funzionamento dei diversi elementi, come le tastiere o gli LCD. L'utente può fare dei propri file di intestazione, anche se deve tenere



Un frammento del file di intestazione del p16F84.inc.

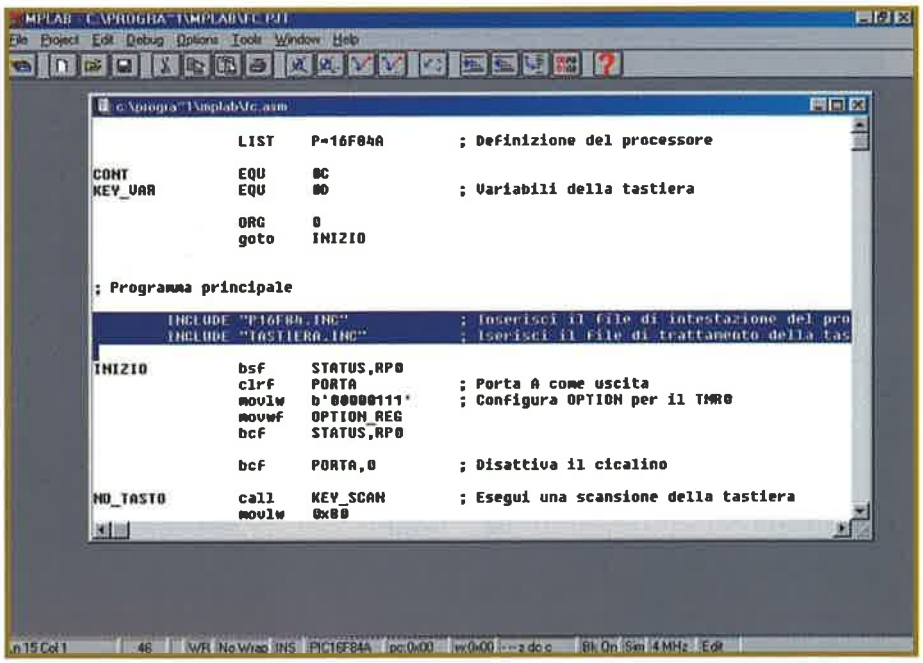


File di intestazione per l'utilizzazione della tastiera.

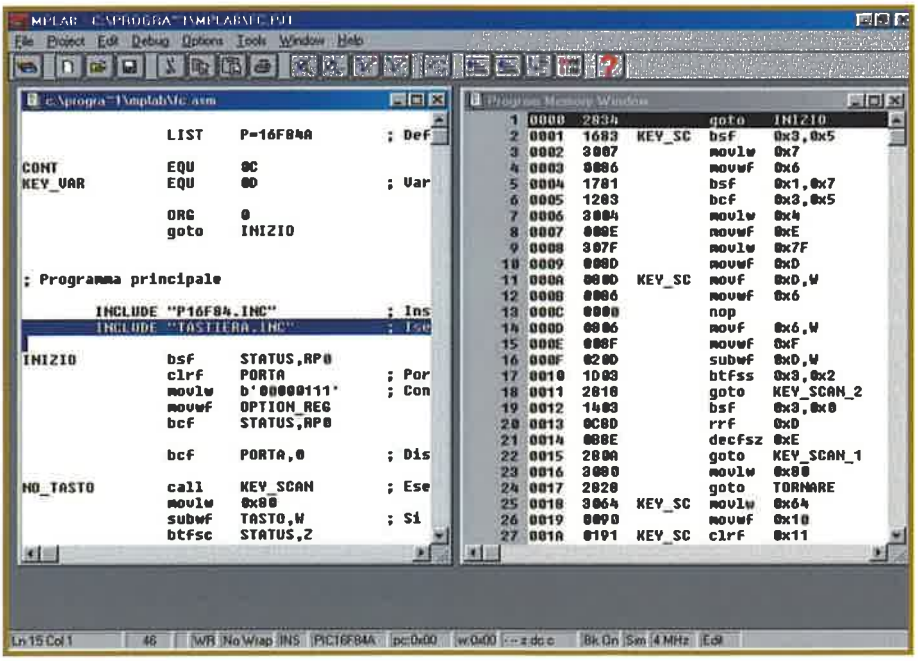
conto, comunque, che nei file utilizzati nello stesso programma, non devono coincidere definizione e subroutine.

L'USO DEI FILE DI INTESTAZIONE

L'uso dei file di intestazione è semplice: dobbiamo solo porre una direttiva INCLUDE nel programma principale e, in seguito, il nome del file fra virgolette doppie. Nel caso dei file di intestazione che definiscono i registri del microcontroller, prima di INCLUDE bisogna definire il proprio PIC. Per fare in modo che il file venga trovato al momento di assemblare il programma, deve essere nella stessa directory. Se osserviamo come si carica il file nella memoria di programma, vedremo che si ottiene lo stesso risultato copiando direttamente il file al posto della direttiva INCLUDE.



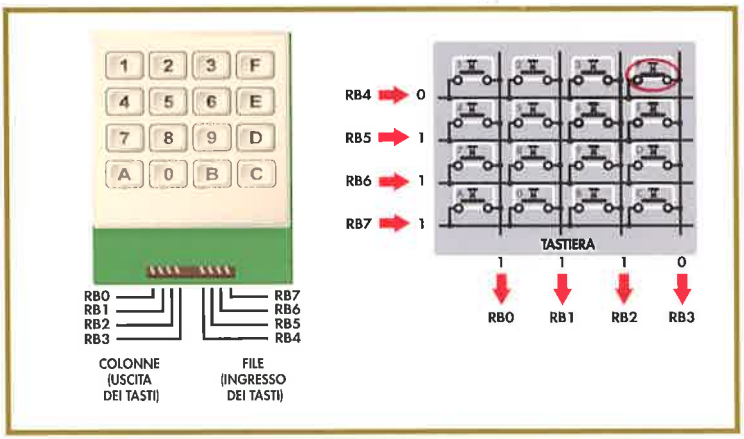
Uso dei file di intestazione.



Esempio di come si carica un INCLUDE nella memoria di programma.

Applicazione pratica: raccolta dei dati tramite la tastiera

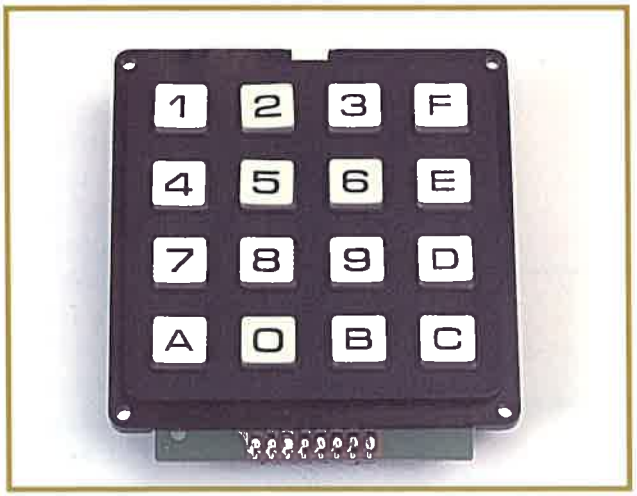
Se finora avete trovato interessanti i programmi con il PIC e iniziate ad immaginare le possibili applicazioni, questa periferica vi incanterà. In alcuni casi i semplici tasti sono sufficienti per introdurre dati in una applicazione, però mano a mano che le applicazioni diventano più complesse, avremo bisogno di sistemi più sofisticati, almeno in apparenza, dato che il loro uso e la programmazione continuano a rimanere semplici.



Schema dei collegamenti di una tastiera da 4x4 tasti.

LA TASTIERA COME PERIFERICA DI INGRESSO

Dal punto di vista elettrico una tastiera è un meccanismo in cui ognuno dei suoi tasti funziona come un pulsante. Comunque la distribuzione di questi pulsanti, insieme al modo con cui vengono gestiti dal programma, fa sì che l'uso della tastiera faciliti l'utente all'introduzione di dati tramite l'interfaccia più intuitiva, oltre a risparmiare linee di ingresso/uscita rispetto all'uso di tasti convenzionali. Così ad esempio una tastiera da 4x4 pulsanti, utilizza solamente otto linee per funzionare, al posto delle 16 che sarebbero necessarie con pulsanti indipendenti. Ci sono diversi modi di configurare una tastiera, a seconda di come sono collegate le linee e se si lavora o meno con interrupt. Per il momento dimentichiamoci di questo, per poter vedere in un modo più semplice il concetto di funzionamento. Dato che una tastiera esadecimale ha bisogno di otto linee di ingresso/uscita, scegliamo la porta B per il suo collegamento. Come possiamo osservare nella figura, le quattro linee meno significative (RB0-RB3) si configurano come ingressi verso il PIC, e le 4 più significative (RB4-RB7) come uscite dal PIC, anche se potrebbe essere il contrario. Il programma dovrà porre sequenzialmente a zero ognuna delle linee di uscita e leggere le linee di ingresso per rilevare i cambiamenti. Se non si preme nessun tasto, i valori raccolti dalle linee di ingresso saranno dei livelli alti, mentre se si preme qualche tasto, la linea di ingresso associata prenderà un livello basso. In questo modo ogni tasto ha associato un codice binario, che identificherà in modo univoco la sua pulsazione, e che sarà formato dai 4 bit che escono dalle linee RB4-RB7 e dai quattro ricevuti da RB0-RB3. Ogni codice sarà formato da 6 livelli alti e 2 bassi, coincidendo questi ultimi con la fila e colonna particolare di ogni



Tastiera modello ECO 16250 06 realizzata dalla ditta SECME.

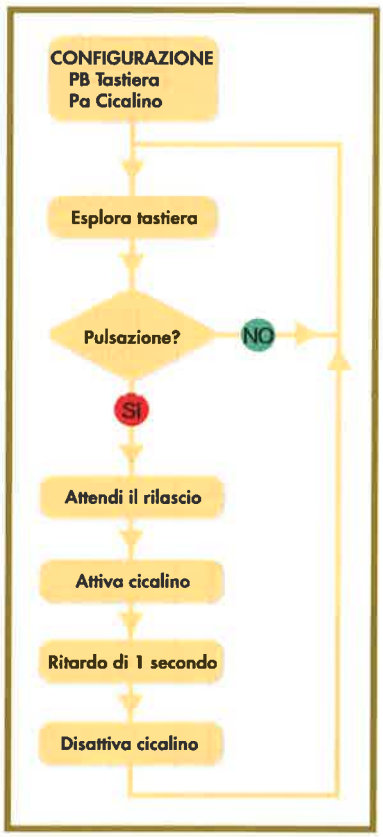
TASTO	FILE (INGRESSI) RB7	RB6	RB5	RB4	COLONNE (USCITE) RB3	RB2	RB1	RB0	CODICE HEX
0	0	1	1	1	1	1	0	1	7D
1	1	1	1	0	1	1	1	0	EE
2	1	1	1	0	1	1	0	1	ED
3	1	1	1	0	1	0	1	1	EB
4	1	1	0	1	1	1	1	0	DE
5	1	1	0	1	1	1	0	1	DD
6	1	1	0	1	1	0	1	1	DB
7	1	0	1	1	1	1	1	0	BE
8	1	0	1	1	1	1	0	1	BD
9	1	0	1	1	1	0	1	1	BB
A	0	1	1	1	1	1	1	0	7E
B	0	1	1	1	1	0	1	1	7B
C	0	1	1	1	0	1	1	1	77
D	1	0	1	1	0	1	1	1	B7
E	1	1	0	1	0	1	1	1	D7
F	1	1	1	0	0	1	1	1	E7

Codice associato ad ogni tasto di una tastiera esadecimale.

tasto. Nel programma di gestione della tastiera si avrà un'esplorazione della tastiera approssimativamente ogni 20 ms anche se, sia il tempo sia i codici, possono variare a seconda del modello e della posizione dei tasti.

PROVA DI FUNZIONAMENTO

L'uso della tastiera in un programma, sebbene non comporti complicazioni una volta compreso il suo funzionamento, implica l'apprendimento di nuove funzioni specifiche per questo compito. Quindi concentriamoci su di essa senza preoccuparci, per il momento, della scarsa utilità del programma in sé. Nell'organigramma che rappresenta l'enunciato proposto, si possono differenziare due parti: l'esplorazione della tastiera e le azioni dopo la rilevazione di un tasto. Il tasto si configurerà come è stato spiegato in precedenza, inoltre si abiliterà una linea di uscita per un cicalino. La scansione della tastiera verrà inserita in una subroutine. Nel caso che questa fornisca una risposta affermativa, e cioè che è stato premuto un tasto, si farà una chiamata a un'altra subroutine, per attendere che il tasto venga rilasciato. Nel nostro esempio non importa quale sia stato il tasto premuto, dato che il procedimento sarà sempre lo stesso: si attiverà il cicalino, per un periodo di tempo, e poi si inizierà nuovamente con la stessa sequenza.



Organigramma dell'enunciato proposto.

**Si tratta di verificare il corretto collegamento della tastiera.
Per questo ogni volta che si preme un pulsante si genera un bip tramite un cicalino piezoelettrico.**

Enunciato per la prova della tastiera.

IL FILE DI TRATTAMENTO DELLA TASTIERA

Il codice presentato nella figura, potrebbe apparire strano se confrontato con i programmi visti fino a questo momento: non ha direttive, né inizio, né fine. Questo perché non si tratta di un programma completo, ma di una serie di routines, nel nostro caso due, che saranno memorizzate come un file a parte, e saranno incluse nel programma principale.

Queste routines avranno bisogno di variabili ausiliarie, la cui definizione si fa all'inizio del file. Dato che si tratta di un file di carattere generale utilizzabile in qualsiasi programma che impieghi una tastiera, non si conosce la prima posizione libera del programma principale in cui collocare le variabili, perciò si utilizzerà la direttiva CBLOCK, che definisce un blocco, il cui nome nel nostro caso è KEY_VAR. In seguito ci preoccuperemo di collocare questo blocco.

Le due routines principali realizzate sono KEY_SCAN e KEY_OFF. La prima esplora la tastiera per file in cerca di un tasto premuto, per cui le attiva una ad una e raccoglie il risultato delle colonne. Se il risultato di quanto inviato e ricevuto è diverso, significa che è stato premuto un pulsante, e la routine fornirà il codice del pulsante premuto. Se testando le quattro file non si rileva nessun pulsante premuto, la routine fornisce il codice 0x80.

L'altra routine si assicura che sia stato premuto solamente un pulsante. Questo è necessario perché nel tempo in cui noi premiamo una volta il pulsante, il programma riesce ad esplorare diverse volte la tastiera, rilevando quindi diverse volte il pulsante premuto, e questo potrebbe generare degli errori.

```
; Blocco delle etichette

                    CBLOCK      KEY_VAR             ; Inizio delle variabili
                                TASTO
                                KEY_1
                                KEY_2
                                KEY_DELAY_1
                                KEY_DELAY_2
                                TEMP_1
                    ENDC

; Routine di scansione della tastiera

KEY_SCAN            bsf         STATUS,RP0          ; Passo al banco 1
                    movlw       b'00001111'
                    movwf       PORTAB              ; RB4-RB7 uscite, RB0-RB3 ingressi
                    bsf         TMR0_OPT_,7         ; Attiva pull-up interni
                    bcf         STATUS,RP0          ; Torno al banco 0
                    movlw       4
                    movwf       KEY_1               ; N° delle file da esplorare
                    movlw       b'01111111'
                    movwf       TASTO               ; Fila da attivare
KEY_SCAN_1          movf        TASTO,W
                    movwf       PORTAB              ; Attiva fila
                    nop
                    movf        PORTAB,W
                    movwf       KEY_2
                    subwf       TASTO,W             ; Legge le colonne
                    btfss       STATUS,Z            ; Qualche pulsante attivo?
                    goto        KEY_SCAN_2          ; Sì
                    bsf         STATUS,C            ; No, non c'è niente in questa fila
                    rrf         TASTO,F             ; Seleziona la fila successiva
                    decfsz      KEY_1,F             ; Salta se sono terminate le file
                    goto        KEY_SCAN_1
                    movlw       0x80
                    goto        TORNARE             ; Restituisci il codice 0x80 (nessun pulsante attivo)

KEY_SCAN_2          movlw       .100                ; Ciclo di temporizzazione di 20ms
                    movwf       KEY_DELAY_1         ; per evitare i rimbalzi del pulsante
KEY_SCAN_3          clrf        KEY_DELAY_2
                    clrwdt
KEY_SCAN_4          decfsz      KEY_DELAY_2,F
                    goto        KEY_SCAN_4
                    decfsz      KEY_DELAY_1,F
                    goto        KEY_SCAN_3
                    movf        TASTO,W             ; Dopo la temporizzazione si legge nuovamente
                    movwf       PORTAB              ; se il tasto è lo stesso. Così si
                    nop                             ; evitano i rimbalzi
                    movf        PORTAB,W
                    subwf       KEY_2,W
                    btfss       STATUS,Z            ; E' lo stesso?
                    goto        KEY_SCAN_1          ; No, prosegui con l'esplorazione
                    movf        KEY_2,W             ; Si scrive nella variabile di uscita TASTO
                                                    ; il valore trovato
TORNARE             movwf       TASTO
                    return                          ; Fine dell'esplorazione

; Routine che attende che il tasto sia rilasciato

KEY_OFF             movf        TASTO,W
                    movwf       TEMP_1              ; Memorizza provvisoriamente il valore del tasto
KEY_OFF_NO          call        KEY_SCAN            ; Verifica che sia rilasciato
                    movlw       0x80                ; Codice di nessun pulsante attivo
                    subwf       TASTO,W
                    btfss       STATUS,Z            ; Salta se il tasto è stato rilasciato
                    goto        KEY_OFF_NO
                    movf        TEMP_1,W
                    movwf       TASTO               ; Si riprende il tasto
                    return
```

Routine di trattamento della tastiera.

```
                LIST        P=16F84A        ; Definizione del processore
STATUS          EQU         03              ; Definizione delle variabili
PORTAA          EQU         05
PORTAB          EQU         06
TMR0_OPT        EQU         01
INTCON          EQU         0B
W               EQU         0
F               EQU         1
Z               EQU         2
C               EQU         0
RP0             EQU         5
CONT            EQU         0C
KEY_VAR         EQU         0D              ; Variabili della tastiera
                ORG         0
                goto        INIZIO
; Programma principale
                INCLUDE "TASTIERA.INC"      ; Inserisci il file di trattamento della tastiera
INIZIO          bsf         STATUS,RP0
                clrf        PORTAA          ; Porta A come uscita
                movlw       b'00000111'     ; Configura OPTION per il TMR0
                movwf       TMR0_OPT
                bcf         STATUS,RP0
                bcf         PORTAA,0        ; Disattiva il cicalino
NO_TASTO        call        KEY_SCAN        ; Esegui una scansione della tastiera
                movlw       0x80
                subwf       TASTO,W         ; Si compara TASTO con il valore 80 (nessuna pulsazione)
                btfsc       STATUS,Z
                goto        NO_TASTO        ; Non è stato premuto nessun tasto
                call        KEY_OFF         ; Si, è stato premuto. Si attende il rilascio
                bsf         PORTAA,0        ; Si attiva il cicalino
                call        RITARDO         ; Si mantiene attivo per 1 secondo
                bcf         PORTAA,0        ; Si disattiva il cicalino
                goto        NO_TASTO        ; Si ricomincia il ciclo
RITARDO         movlw       d'15'           ; Ciclo esterno che si ripeterà 15 volte
                movwf       CONT
CICLO1          movlw       00              ; Inizializzazione del ciclo interno
                movwf       TMR0_OPT
CICLO2          btfss       INTCON,2        ; Ha finito di contare?
                goto        CICLO2          ; No, continua il ciclo interno
                bcf         INTCON,2        ; Si, resetta il flag
                decfsz      CONT,1          ; Decrementa il contatore del ciclo esterno
                goto        CICLO1          ; Dato che non è 0, torna a ripetere il ciclo interno
                return                      ; Il contatore esterno è a 0, esci dalla routine
END
```

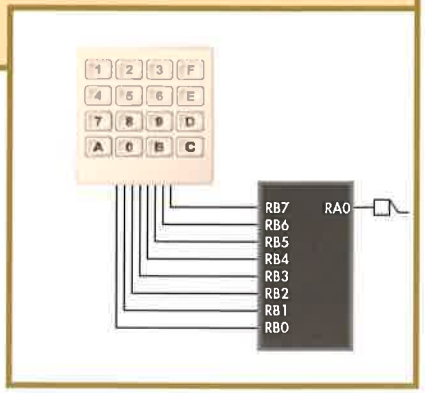
Programma completo dell'enunciato proposto.

PROGRAMMA PRINCIPALE

Il programma, una volta incluso nel nuovo file della tastiera, risulta molto semplice. Dopo aver configurato la porta A per l'uso del cicalino e il TMR0, si esplora la tastiera, e nel caso si rilevi un pulsante premuto si attiva il cicalino per un secondo, per poi spegnerlo. Si ripete il processo in modo infinito.

SIMULAZIONE

Nel caso della tastiera non possiamo realizzare una simulazione tramite MPLAB, dato che non esiste nessun dispositivo di ingresso che si possa configurare come si farebbe con una tastiera, associata a due linee. Pertanto, per verificare quanto visto, vi suggeriamo di realizzare il montaggio necessario sopra una scheda per prototipi, o su una scheda universale. Il programma vi aiuterà a verificare che le connessioni siano realizzate correttamente, dato che il cicalino suonerà premendo qualsiasi pulsante.



Schema completo dell'esercizio.

Direttive

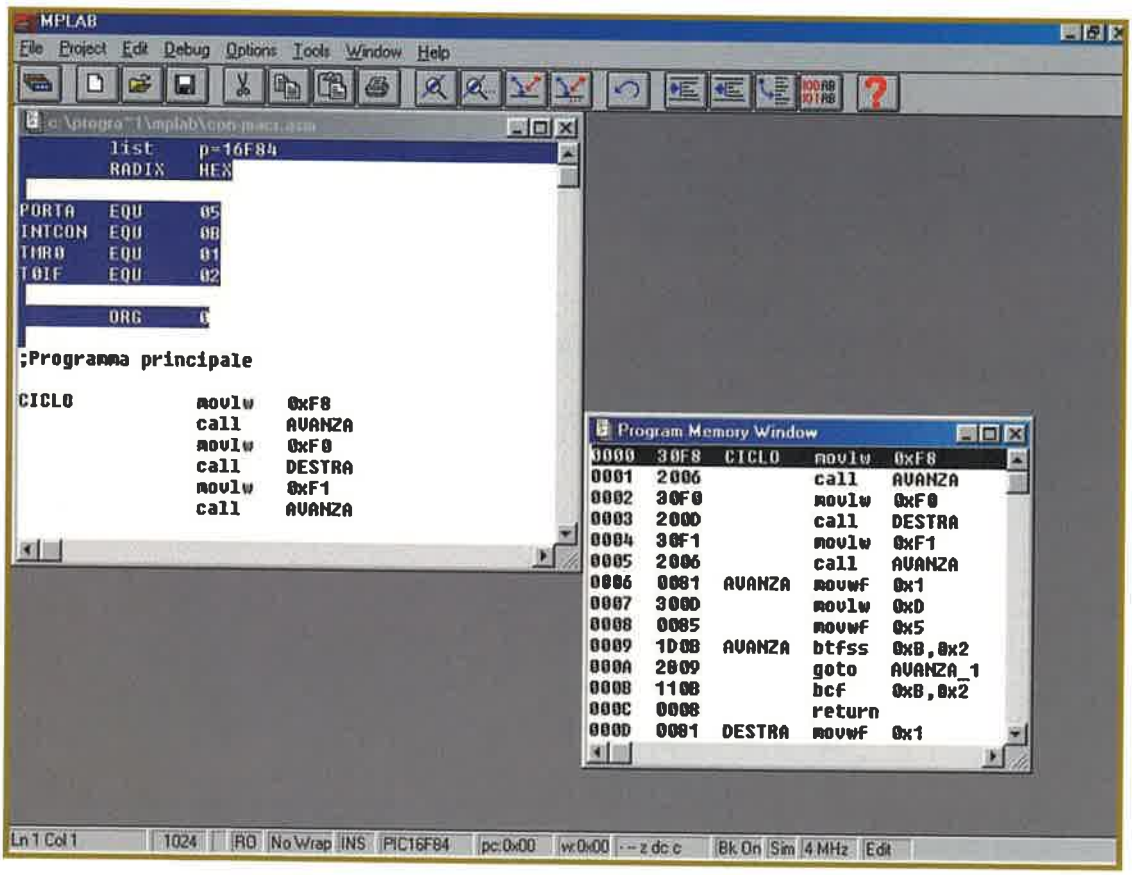
Alcune le abbiamo già viste, dato che ne abbiamo avuto bisogno nei programmi che abbiamo realizzato, però ora le tratteremo in modo approfondito, vedendo sino a che punto possono essere utili.

CONCETTO DI DIRETTIVA

Una direttiva è un comando assembler che si inserisce nel codice sorgente del programma, però non viene tradotto in codice macchina dal compilatore. Si utilizza per diversi scopi, in relazione al posizionamento delle variabili, e al controllo degli ingressi e delle uscite nell'assemblato. La maggioranza delle direttive ha diversi nomi e formati, per preservare la compatibilità con le versioni precedenti dei compilatori di Microchip, e per rendere compatibili i programmi realizzati da diversi utenti.

Il compilatore MPASM fornisce cinque tipi di direttive:

- Direttive di Controllo, che permettono l'assemblamento di frammenti di codice secondo determinate condizioni.
- Direttive di Dati, che controllano il posizionamento dei dati nella memoria e danno la possibilità di riferirsi ad essi con nomi significativi.
- Direttive di Listato, che servono per configurare la forma in cui si presenterà il file assembler: titoli, impaginazione, ecc.
- Direttive di Macro, che controllano l'esecuzione e



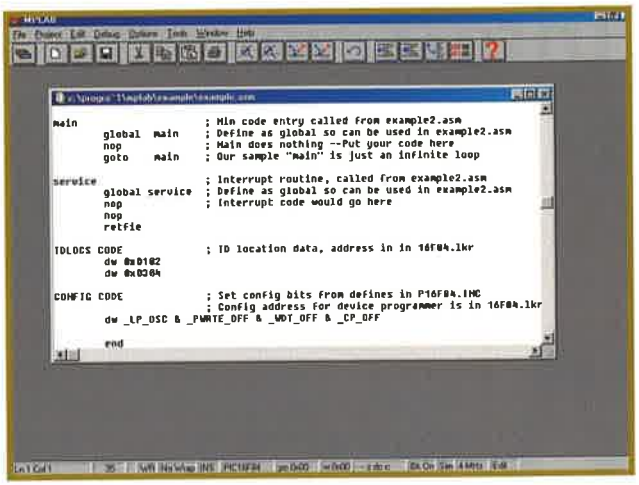
Le direttive non vengono tradotte in codice macchina quando si assembla un programma.

il posizionamento dei dati quando si utilizzano macro nella programmazione.

• Direttive di File Object, che si utilizzano quando si vuole creare un file object.

USO DELLE DIRETTIVE

Nella maggioranza dei casi i programmatori devono utilizzare sempre le stesse direttive, dato che sono molto specifiche quelle dei dati e delle macro, e solo in presenza di situazioni speciali, come ad esempio se si vuole stampare il listato di un file con dimensioni differenti dallo standard, si prova a trovare una soluzione con l'uso di altre direttive. Anche se in seguito non le utilizzeremo, conviene fare un ripasso di tutte le direttive esistenti, per questo le presentiamo nella tabella con una breve descrizione per ognuna.



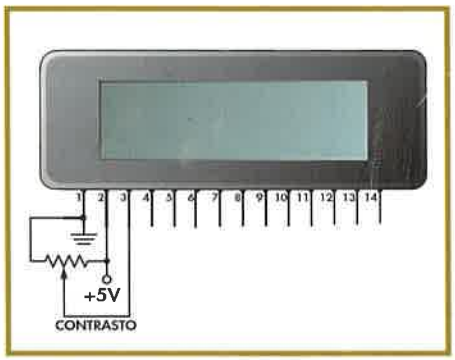
Esempio di un programma che utilizza diverse direttive meno frequenti.

DIRETTIVA	DESCRIZIONE
BADRAM	Specifica posizione invalida della RAM
BANKISEL	Genera codice per la selezione del banco nella memoria RAM e nell'indirizzamento indiretto
BANKSEL	Genera codice per la selezione del banco nella memoria RAM
CBLOCK	Definisce un blocco di costanti
CODE	Inizia la sezione del codice eseguibile
CONFIG	Specifica la configurazione dei bit
CONSTANT	Dichiara le costanti simboliche
DA	Fornisce una catena di caratteri nella memoria di programma
DATA	Crea dati numerici e di testo
DB	Dichiara un dato di un byte
DE	Definisce un dato della EEPROM
#DEFINE	Definisce un'etichetta di testo per sostituzione
DT	Definisce una tabella
DW	Dichiara un dato di una parola
ELSE	Inizia il blocco alternativo di una sentenza IF
END	Termina il programma
ENDC	Termina un blocco di costanti
ENDIF	Termina un blocco condizionale IF
ENDM	Termina la definizione di macro
ENDW	Termina un ciclo WHILE
EQU	Definisce una costante
ERROR	Fornisce un messaggio di errore
ERRORLEVEL	Pone il livello di errore
EXITM	Esce da una macro
EXPAND	Espande una macro
EXTERN	Dichiara un'etichetta esterna
FILL	Riempie la memoria con un valore
DATA	Inizia la sezione di inizializzazione dei dati
IF	Inizia il blocco condizionale
IFDEF	Esegue se il simbolo è stato definito
IFNDEF	Esegue se il simbolo non è stato definito
#INCLUDE	Include un file sorgente
LIST	Lista una serie di opzioni
LOCAL	Dichiara variabili locali di macro
MACRO	Dichiara la definizione di una macro
MAXRAM	Specifica il massimo indirizzo della RAM
MESSG	Crea un messaggio definito dall'utente
NOEXPAND	Disabilita l'espansione di macro
ORG	Stabilisce l'origine del programma
PAGESEL	Genera il codice per selezionare la pagina in ROM
PROCESSOR	Stabilisce il tipo di processore
RADIX	Stabilisce il codice di numerazione per default
RES	Riserva memoria
SET	Definisce una variabile
SPACE	Inserisce linee vuote
SUBTITLE	Specifica un sottotitolo di un programma
TITLE	Specifica un titolo di un programma
#UNDEFINE	Cancella un'etichetta di sostituzione
WHILE	Inizia il blocco WHILE se la condizione è vera

Tabella di alcune direttive fornite dal MPASM.

Applicazione pratica: visualizzazione dei dati tramite LCD

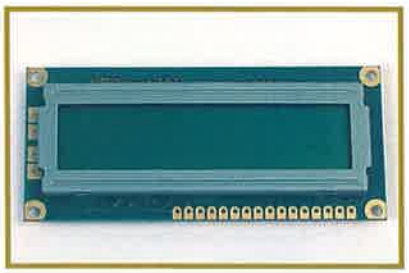
Come la tastiera è la più comune tra le periferiche di ingresso nei progetti con microcontroller, il display a cristalli liquidi LCD è il più comune tra le periferiche di uscita. Come utenti lo potremo trovare sempre più spesso sui dispositivi commerciali, come ad esempio, macchine distributrici di prodotti. Quindi, non vi piacerebbe imparare a programmarli per fornire messaggi del tipo "introdurre l'importo", "può passare", ecc.? Questa è la nostra proposta.



Collegamenti di un LCD da 2x16 caratteri.

LCD COME PERIFERICA DI USCITA

Un LCD è un dispositivo molto potente, flessibile e facile da gestire. È capace di rappresentare lettere, numeri e altri caratteri ASCII, o simboli specifici programmati dall'utente, tutto questo in uno spazio, ad esempio, di 2 linee per 16 colonne, a seconda del modello. Il modulo LCD contiene un microcontroller, che gestisce il suo funzionamento e ha il compito di tradurre i comandi mandati tramite il programma in ordini compresi dal display. Il display LCD con cui vogliamo lavorare riceve l'informazione tramite 14 pin, otto dei quali ricevono i dati o i codici di controllo (D0/D7), a seconda del momento, tre sono incaricati di controllare il funzionamento del modulo (RS, R/W#, E), e i tre rimanenti sono per l'alimentazione (Vss, Vdd) e il contrasto (V0). Quando il display LCD è collegato a un PIC16F84, le linee dei dati di solito corrispondono con la porta B e le tre di controllo con le linee meno significative della porta A, RA0-RA2. Le linee dei dati dovranno essere bidirezionali, cioè a volte ingressi e a volte uscite, dato che questo è necessario per passare dati all'LCD o per ricevere informazioni sul suo stato interno. Le linee di controllo si configurano come uscite.



Display LCD modello WM-C1602M della ditta WINTEK.

COMANDO	RS	R/W'	E	DH7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tempo di esecuzione
CLEAR DISPLAY	0	0	1	0	0	0	0	0	0	0	1	1,64 ms
HOME	0	0	1	0	0	0	0	0	0	1	X	1,64 ms
ENTRY MODE SET	0	0	1	0	0	0	0	0	1	I/D	S	40 ms
DISPLAY ON/OFF	0	0	1	0	0	0	0	1	D	C	B	40 ms
CURSOR DISPLAY SHIFT	0	0	1	0	0	0	1	S/C	R/L	X	X	40 ms
FUNCTION SET	0	0	1	0	0	1	DL	N	F	X	X	40 ms
SET CGRAM ADDRESS	0	0	1	0	1	INDIRIZZI DELLA CGRAM						40 ms
SET DDRAM ADDRESS	0	0	1	1	INDIRIZZI DELLA DDRAM							40 ms
READ BUSY FLAG S ADDRESS	0	0	1	BF	INDIRIZZI DELLA CGRAM O DDRAM							40 ms
WRITE DATA TO	0	0	1	CODICE ASCII PER LA RAM								40 ms
CG O DD READ DATA TO CG O DD	0	0	1	CODICE MEMORIZZATO NELLA RAM								40 ms

** **S**: Se vale 1, sposta la visualizzazione ogni volta che si scrive un dato. Se vale 0, funziona in modo normale. * **I/D**: Se vale 1, si incrementa l'indirizzo del cursore; se vale 0, si decrementa. * **S/C**: Se vale 1, sposta la visualizzazione; se vale 0, sposta il cursore. * **R/L**: Se vale 1, lo spostamento è a destra; se vale 0, a sinistra. * **BF**: Se vale 1, il modulo LCD è occupato; se vale 0, è disponibile. * **DL**: Se vale 1, si lavora con bus dei dati a 8 bit; se vale 0, con bus a 4 bit. * **N**: Se vale 1, la presentazione si fa in due linee; se vale 0, in una. * **F**: Se vale 1, il carattere è di 5 x 10 pixel; se vale 0, di 5 x 7. * **B**: Se vale 1, lampeggia il cursore (se è ON). * **C**: Se vale 1, il cursore è attivo (ON). * **D**: Se vale 1, il display è attivo. * **X**: Indeterminato.*

Tabella dei codici di controllo del display LCD.

A seconda della combinazione dei dati delle linee di controllo, l'LCD capirà se deve rappresentare un carattere, cambiare la posizione del cursore (che gli si domanda se ha finito con quello che sta facendo) ecc. Combinando tutte le opzioni si possono ottenere effetti come il lampeggiamento dei caratteri rappresentati, l'apparizione e sparizione dei caratteri dai lati del display, la rappresentazione di figure in movimento ecc.

ESEMPIO DI FUNZIONAMENTO

Il primo passo da fare quando si sta cominciando ad acquisire un nuovo linguaggio di programmazione, è generare un messaggio che dica "MONTY". Fino ad ora non avevamo questa possibilità, dato che l'unica periferica per rappresentare numeri e lettere era il display a 7 segmenti, anche se in modo abbastanza limitato. Il display LCD ci darà questa opportunità in modo comodo e semplice. Come possiamo osservare, l'organigramma è semplice. Noi dovremo solo inviare i dati all'LCD e indicare dove metterli, e lui si incaricherà del resto.

Si vuole verificare il funzionamento del display LCD. Per questo si scriverà il messaggio "MONTY".

Enunciato dell'esempio di funzionamento dell'LCD.

Organigramma dell'enunciato dato.

LE ROUTINES DI CONTROLLO DELL'LCD

Così come il modulo della tastiera, il display LCD ha bisogno di una serie di routines che di solito vengono incluse come file indipendenti del programma principale. Oltre a includere i file dovremo dichiarare le variabili che verranno utilizzate, tenendo sempre conto che le posizioni non siano già occupate da altri file. Notate nella tavola a lato che, oltre alla dichiarazione delle variabili, abbiamo introdotto le dichiarazioni di etichetta mediante la direttiva "#define".

```
#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 disponibile
                            Lcd_Temp_2
            ENDC

UP_LCD      bsf             STATUS,RP0          ; Banco 1
            clrf            PORTB               ; RB<0-7> uscite digitali
            clrf            PORTA               ; Porta A uscita
            bcf             STATUS,RP0          ; Banco 0
            OFF_COMANDO                         ; RS=0
            DISABLE                             ; E=0
            return

LCD_BUSY    LEGGERE                             ; Pone LCD in Modo RD
            bsf             STATUS,RP0
            movlw           h'FF'
            movwf           PORTB               ; Porta B ingresso
            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                             ; Disabilita LCD
            bsf             STATUS,RP0
            clrf            PORTB               ; Porta B uscita
            bcf             STATUS,RP0
            SCRIVERE                            ; Pone LCD in Modo WR
            return

LCD_E       ENABLE                              ; Attiva E
            nop
            DISABLE                             ; Disattiva E
            movlw           .14
            movwf           Lcd_Temp_1
LCD_E_1     decfsz          Lcd_Temp_1,F        ; Perde 40 µs per la costante di
            goto            LCD_E_1             ; tempo Tc dei nuovi moduli LCD di
            return                              ; Wintek

LCD_DATO    OFF_COMANDO                         ; Disattiva RS (modo comando)
            movwf           PORTB               ; Valore ASCII da mettere sulla PORTB
            call            LCD_BUSY            ; Aspetta che si liberi LCD
            ON_COMANDO                          ; Attiva RS (modo dato)
            goto            LCD_E               ; Genera impulso di E

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

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

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 trattamento dell'LCD.

UP_LCD:	Configurazione delle linee del PIC per LCD
LCD_BUSY:	Lettura del Flag Busy e indirizzo
LCD_E:	Impulso di Enable. Nei nuovi LCD questo segnale deve stare a "0", 45 mS prima di tornare ad essere attivato a "1"
LCD_DATO:	Scrittura dei dati in DDRAM o CGRAM. Invia dato presente in W
LCD_REG:	Scrittura dei comandi dell'LCD. Invia il comando presente in W
LCD_INI:	Inizializzazione dell'LCD inviando il comando "Function Set" 3 volte consecutive con un intervallo di 5 mS. LCD rimane cancellato e il cursore nella prima posizione
LCD_DELAY:	Routine di temporizzazione di 5 mS. Si utilizzano le variabili Lcd_Temp_1 e LCD_Temp_2 al posto del TMR0. Questo rimane libero per le applicazioni dell'utente.

Significato delle routines dell'LCD.

Ogni volta che si utilizza, si fa corrispondere all'etichetta che segue il valore successivo. Così ad esempio, dopo questo #define, ogni volta che nel programma si scrive "ENABLE", è come se si ponesse "bsf PORTA,2". Questo avvicina un po' di più l'assembler ai linguaggi di alto livello. Nella tabella allegata troviamo diverse routines implementate e le loro funzioni, insieme al modo di utilizzarle. Le più importanti sono: UP_LCD, LCD_INI, LCD_REG, e LCD_DATO. Queste si richiamano alle altre per il loro funzionamento, però non è più necessario che lo faccia anche l'utente nelle sue applicazioni.

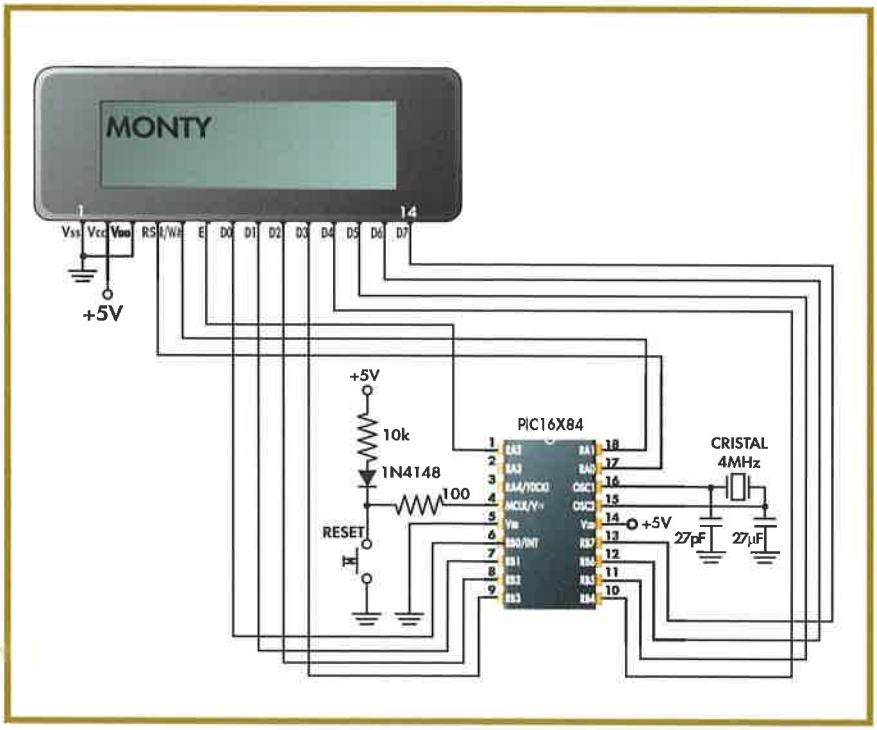
PROGRAMMA PRINCIPALE

Nel programma, per prima cosa bisogna configurare il PIC per l'uso della tastiera: e questo va fatto ogni volta che lavoriamo con il display, dato che è normale che condivida le linee con le altre periferiche e che queste le utilizzino in un altro modo. In seguito dovremo inizializzare l'LCD. Questo processo è necessario solo la prima volta che si utilizza un LCD nel programma. Dopo aver iniziato a mandare ordini o dati all'LCD, la procedura è sempre uguale. Quando si tratta di un ordine, si deve introdurre il codice di controllo in W e chiamare la routine LCD_REG, mentre se si tratta di un dato, si metterà quest'ultimo in W e si chiamerà LCD_DATO. I comandi più utilizzati, come ad esempio cancella LCD e porta il cursore alla prima posizione, saranno introdotti a loro volta in una subroutine, però internamente faranno la stessa cosa. Nel nostro caso, dato che vogliamo prendere una frase completa, invece di dare ordini per il posizionamento del cursore per ogni lettera in modo individuale, abbiamo realizzato un ciclo che percorre la tabella dei dati e ce li mostra.

Questo ciclo è implementato come routine (MENS) e necessita, come parametro, dell'indirizzo della catena che si vuole visualizzare.

PROVA

Il montaggio per la verifica di questo esercizio è mostrato in figura, e segue la configurazione tipica usata per la maggior parte da ditte che utilizzano moduli che integrano un display LCD. MPLAB non dispone di risorse per simulare il funzionamento di questo tipo di periferiche.



Schema completo dell'esercizio.

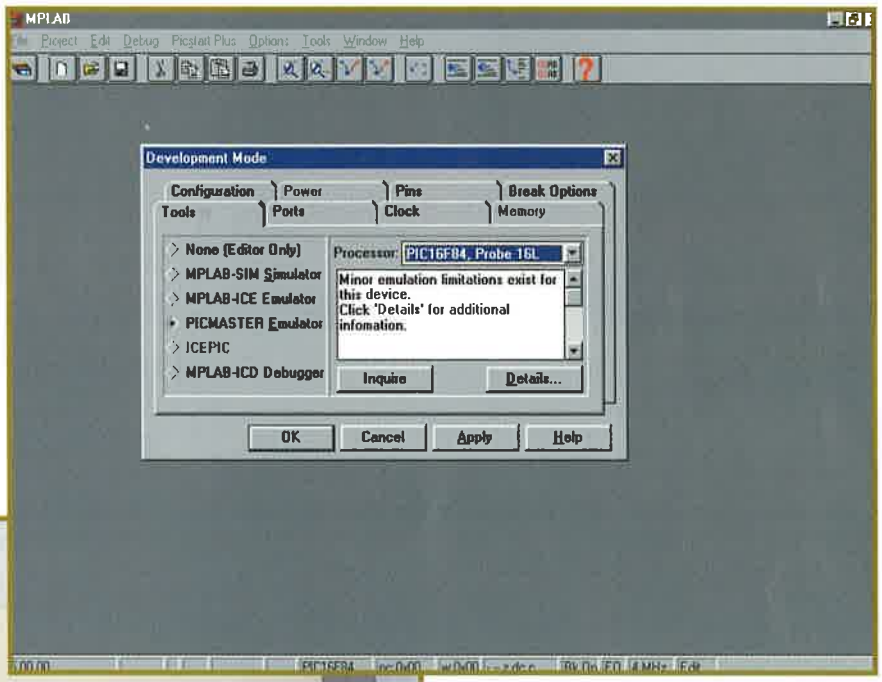
```
                LIST        p=16F84A        ; Tipo di processore
                INCLUDE     P16F84A.INC     ; Definizione dei registri interni

Lcd_var         EQU         0x20            ; Inizio delle variabili delle routine LCD
Delay_var       EQU         0x22            ; Variabili temporali per la temporizzazione

                ORG         0x00            ; Vector di Reset
                goto        Inizio

                ORG         0x05            ; Salva il Vector di interrupt

                INCLUDE     LCD_CXX.INC     ; Include le routine di gestione dell'LCD


Delay           sleep                       ; Temporizza 36,4 ms sino a che il WDT vada in overflow
                decfsz      Delay_var,F     ; È stato ripetuto il numero di volte desiderato?
                goto        Delay           ; No. Attendi altri 18 ms
                return                      ; Sì.

; Programma principale

Inizio          clrf        PORTB           ; Cancella i latch di uscita
                clrf        PORTA           ; Cancella i latch di uscita
                bsf         STATUS,RP0      ; Seleziona il banco 1
                clrf        TRISB           ; Porta B configurata come uscita
                clrf        TRISA           ; Porta A configurata come uscita
                movlw       b'00001001'
                movwf       OPTION_REG      ; Prescaler di 2 al WDT
                bcf         STATUS,RP0      ; Seleziona il banco 0

                call        UP_LCD          ; Configura la porta per LCD
                call        LCD_INI         ; Inizializza LCD
                movlw       b'00001100'
                call        LCD_REG         ; LCD ON cursore e blink OFF

Loop            clrwdt                      ; Resetta il WDT
                movlw       0x86
                call        LCD_REG         ; Pone il cursore nella 7° posizione
                movlw       'M'
                call        LCD_DATO        ; Visualizza "M"
                movlw       'O'
                call        LCD_DATO        ; Visualizza "O"
                movlw       'N'
                call        LCD_DATO        ; Visualizza "N"
                movlw       'T'
                call        LCD_DATO        ; Visualizza "T"
                movlw       'Y'
                call        LCD_DATO        ; Visualizza "Y"

                movlw       .28
                movwf       Delay_var
                call        Delay           ; Temporizza 1"

                movlw       b'00000001'
                call        LCD_REG         ; Cancella il display
                movlw       .28
                movwf       Delay_var
                call        Delay           ; Temporizza 1"
                goto        Loop

                END                         ; Fine del programma
```

Programma completo dell'enunciato proposto.

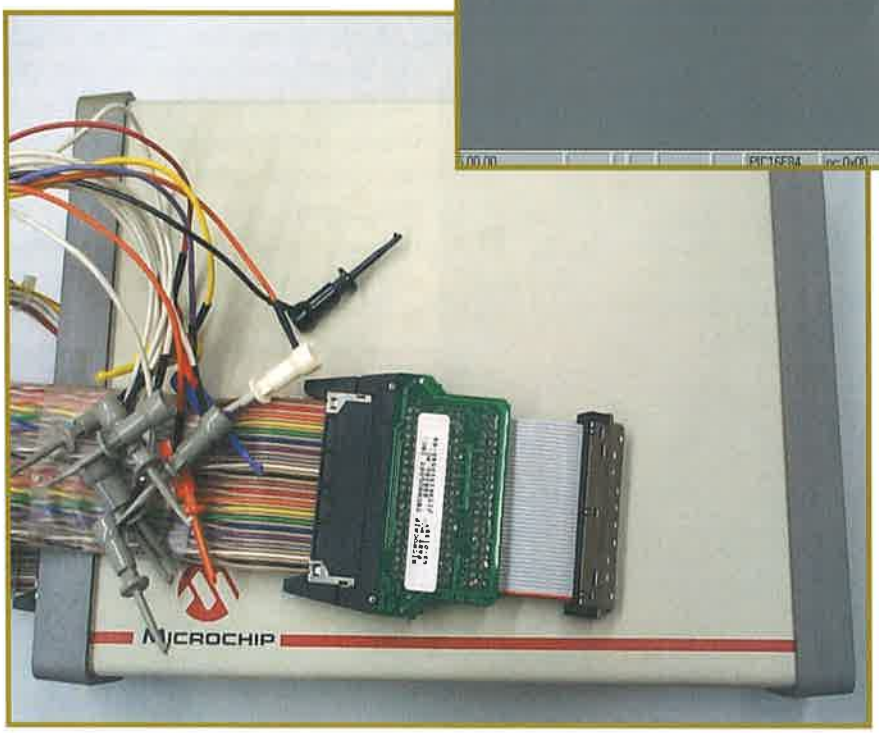
Strumenti di lavoro: caratteristiche avanzate di MPLAB (I)

Quando abbiamo presentato MPLAB, accennammo al fatto che era un ambiente di sviluppo per il lavoro integrato con i PIC, dato che con lo stesso programma si possono realizzare tutte le funzioni necessarie allo sviluppo di un progetto, anche se alcune di esse necessitano di un hardware specifico.

Ora vediamo le caratteristiche più interessanti di due di questi moduli addizionali, accessibili tramite MPLAB.



Per lavorare con l'emulatore dovremo cambiare l'ambiente di sviluppo.



Fotografia dell'emulatore PICMASTER.

L'EMULATORE IN CIRCUITO PICMASTER

Un emulatore è uno strumento hardware che serve per mettere a punto programmi che funzionano come se si stessero eseguendo in tempo reale. MPLAB dispone di strumenti software per poter utilizzare, fra gli altri, l'emulatore PICMASTER. Un emulatore si accomuna e si differenzia dal simulatore e dalla applicazione reale in diverse cose:

- L'emulatore contiene linee di ingresso/uscita e dispositivi, che

PICSTART Plus per la scrittura dei microcontroller.

hanno la stessa funzionalità del processore reale, mentre il simulatore è un modello limitato delle caratteristiche del microcontroller.

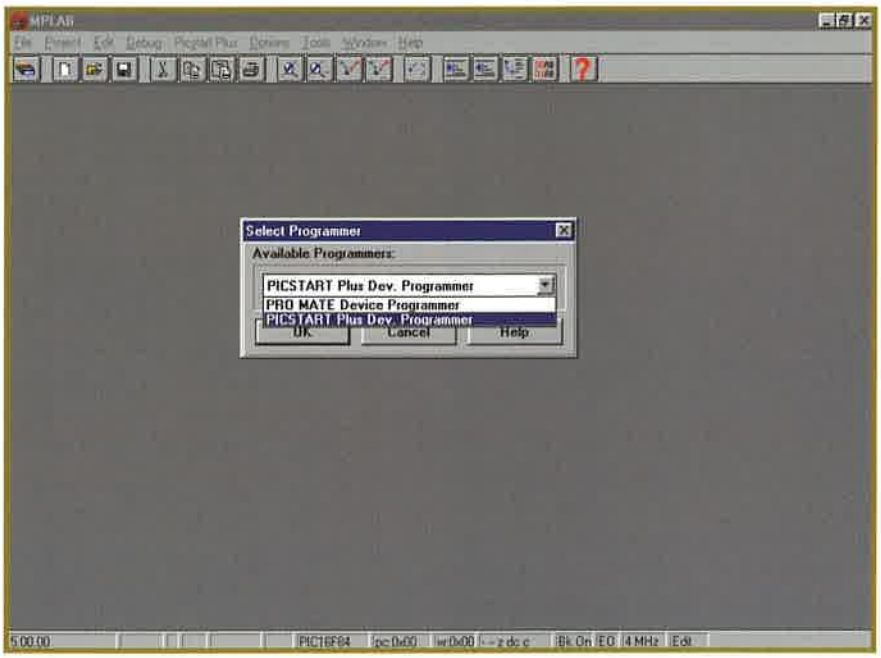
• La velocità di esecuzione dell'emulatore e la risposta agli stimoli esterni, o eventi interni, è in tempo reale.

• Il vantaggio di utilizzare un emulatore rispetto al circuito reale, è che il primo può mostrare ciò che accade all'interno del microcontroller in tempo reale; serve per mettere a punto il programma, potendo inoltre attivare dispositivi come un oscilloscopio a tale scopo.

• La "testa di prova" dell'emulatore è intercambiabile a seconda del PIC da emulare, e si introduce nello zoccolo destinato al microcontroller come si farebbe con un PIC normale. Le linee addizionali che l'emulatore porta all'esterno, si possono collegare alle periferiche, per vedere il valore di determinate linee, o per l'introduzione di eventi esterni. Sarà necessario configurare il PIC-MASTER seguendo le specifiche del fabbricante, che dipendono dal microcontroller da emulare. Bisognerà cambiare anche l'ambiente di sviluppo come già abbiamo fatto iniziando a simulare i programmi, configurare le porte, il clock, ecc. Per sapere se l'emulatore funziona in modo adeguato, possiamo fare una prova nell'opzione Tools>Verify PICMASTER.

LO SCRITTORE PICSTART PLUS

Un altro hardware molto interessante, che si può anche gestire tramite MPLAB, è lo scrittore PICSTART Plus. Dopo le fasi di simulazione ed emulazione arriva il momento di provare l'applicazione sull'hardware reale e fra le molteplici offerte del mercato, esiste la possibilità di utilizzare lo scrittore che offre Microchip. È pensato per essere utilizzato con prototipi o piccole serie, dato che per grandi produzioni, la programmazione "in circuit" è più comoda. Il PICSTART Plus si collega al computer con un'interfaccia seriale, per poter ricevere il programma tramite lo stesso ambiente di sviluppo.



Dovremo scegliere il programmatore da utilizzare.

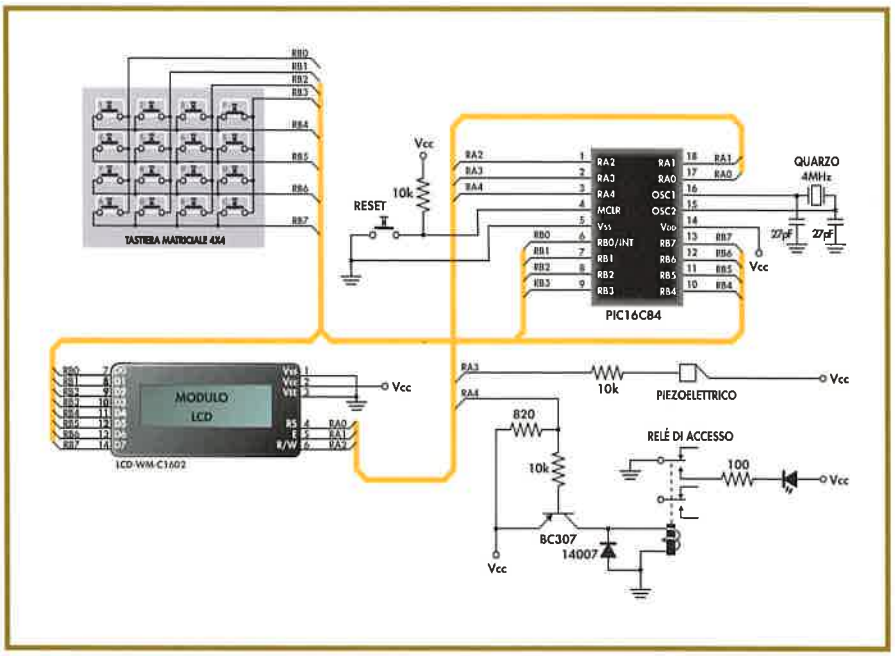
• Nello stesso modo è possibile leggere l'informazione contenuta in un microcontroller non protetto per metterlo a punto, modificarlo o copiarne il codice. Dopo aver installato il dispositivo dovremo scegliere il programmatore da utilizzare. Nella barra dei menù ne apparirà uno nuovo per il lavoro con il PICSTART Plus che permetterà di scrivere, leggere e cancellare i microcontroller.

Applicazione pratica: un programma completo

È giunto il momento di mettere in pratica in un unico programma tutto ciò che abbiamo imparato. Uniremo l'introduzione dei dati tramite la tastiera, con la visualizzazione degli stessi tramite LCD e con le altre semplici periferiche che saranno necessarie. Per quanto riguarda le subroutines abbiamo cercato un'applicazione in cui la loro utilizzazione non solo abbia senso, ma sia indispensabile per un buon funzionamento.

Si vuole controllare l'accesso ad un recinto. Per fare questo disponiamo di una tastiera, in cui, dopo aver premuto il pulsante "A" di "accesso", si dovrà introdurre un codice formato da quattro digit. Se il codice è corretto si attiverà un relè per due secondi, il quale simulerà l'apertura di una porta. Si daranno tre opportunità per l'introduzione del codice. Questo sarà memorizzato nella EEPROM, e potrà essere cambiato con un pulsante della tastiera "C" di "cambio". Un cicalino piezoelettrico emetterà un "beep" premendo ogni pulsante. Un display LCD, da parte sua, genererà messaggi che guideranno l'utente nel suo compito.

Enunciato dell'esercizio proposto.



Schema elettrico dell'esercizio proposto.

SCHEMA ELETTRICO

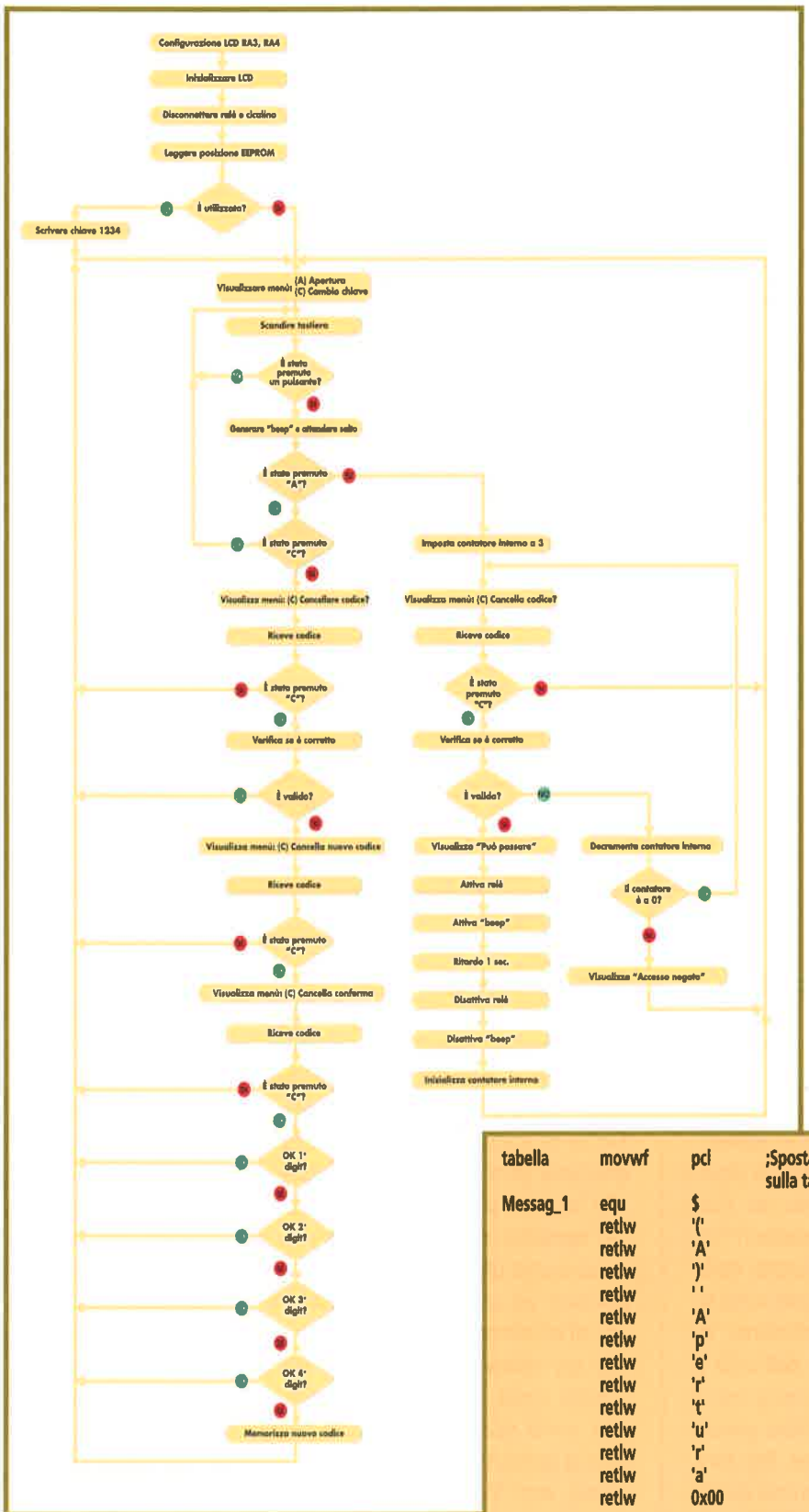
Leggendo l'enunciato dell'esercizio è possibile che abbiate pensato "e dove li trovo io così tanti piedini?" In effetti se contiamo il numero degli ingressi/uscite delle periferiche da utilizzare, ci sembrerà che quelle offerte dal nostro piccolo PIC16F84 non siano sufficienti. In ogni caso grazie alle caratteristiche proprie dell'LCD e della tastiera, avremo otto linee che si utilizzeranno a volte come ingressi a volte come uscite, in altre parole saranno condivise da entrambe le periferiche. Per fare in modo che questo sia possibile, il programma dovrà configurare queste linee a seconda delle periferiche da utilizzare in quel determinato momento.

ORGANIGRAMMA

Come possiamo osservare dall'organigramma esposto, il programma non è così semplice come quelli visti finora. Ciò è dovuto al fatto che sono contemplati diversi casi, e che in qualsiasi momento è permessa l'opzione di cancellazione. Per arrivare a questo organigramma prima abbiamo dovuto pensare molto bene alle opzioni permesse, e fare un primo organigramma più semplice per complicarlo successivamente con l'introduzione delle eccezioni. Quindi all'inizio è normale pensare ad un codice di un solo digit, non permettere più di un errore, e supporre che la EEPROM abbia sempre un valore memorizzato, ecc. In ogni caso, a dispetto della dimensione dell'organigramma, notate che sono state rispettate le regole della struttura, e non ci sono diversi punti di fine, ingressi a metà di funzione, ecc. Per questa ragione la traduzione in programma assembler non dovrebbe comportare alcun problema.

PROGRAMMAZIONE

Software



Organigramma dell'esercizio proposto

PROGRAMMA

Questo programma risulta molto lungo. Inoltre notate che si parte da ciò che si conosce e si hanno a disposizione tre file, di cui due contengono le routines per il trattamento della tastiera e dell'LCD, il terzo contiene la definizione dei registri del PIC. Di questi abbiamo già parlato in precedenza.

La prima cosa che appare nel programma dopo la configurazione delle variabili, sono i messaggi che devono apparire sull'LCD. Tutti i messaggi sono raggruppati in una tabella simile a quella utilizzata con il display a sette segmenti, ma presenta la caratteristica di essere divisa in frammenti e per accedere ad ognuno dei messaggi indipendentemente, si assegna all'etichetta che simbolizza il suo nome, il valore del PC in quella posizione. In questo modo, potremo cambiare il testo dei messaggi o posizionarlo da un'altra parte del codice, senza che influenzi il programma che lo sta chiamando. Inoltre si dovranno utilizzare le routines di lettura e scrittura della EEPROM per leggere e scrivere la chiave. In questo programma sono presentate in modo diverso da come le abbiamo viste negli altri esercizi, quindi allo stesso modo delle routines dell'LCD e della tastiera si potrebbero inserire in un file a parte. Il resto di domande e funzioni, si segue a partire dall'organigramma, che corrisponde con la parte principale del programma.

Esempio di come appariranno i messaggi.

```
tabella     movwf   pcl     ;Spostamento
                            sulla tabella

Messag_1    equ     $
            retlw   '('
            retlw   'A'
            retlw   ')'
            retlw   ' '
            retlw   'A'
            retlw   'p'
            retlw   'e'
            retlw   'r'
            retlw   't'
            retlw   'u'
            retlw   'r'
            retlw   'a'
            retlw   0x00
```

```
Messag_1_1  equ     $
            retlw   '('
            retlw   'C'
            retlw   ')'
            retlw   ' '
            retlw   'C'
            retlw   'a'
            retlw   'm'
            retlw   'b'
            retlw   'i'
            retlw   'o'
            retlw   ' '
            retlw   'C'
            retlw   'l'
            retlw   'a'
            retlw   'v'
            retlw   'e'
            retlw   0x00
```

1

```
                LIST        P=16F84A        ; Tipo di processore
CBLOCK                      0X11
                            digit_1
                            digit_2
                            digit_3
                            digit_4         ; Variabili per i digit del codice
                            di_tem_1
                            di_tem_2
                            di_tem_3
                            di_tem_4        ; Variabili temporali per i digit
                            cont_err        ; Contatore di errori
                            cont_tasti      ; Contatore tasti premuti
                            temp_1
                            temp_2          ; Variabili temporali
                            Delay_1
                            Delay_2
                            Delay_3         ; Variabili per la temporizzazione
ENDC

                INCLUDE     "P16F84A.INC"   ; Definizione dei registri interni

                #DEFINE     _z              STATUS,2
                #DEFINE     _rp0            STATUS,5
                #DEFINE     _rd             EECON1,0
                #DEFINE     _wr             EECON1,1
                #DEFINE     _wren           EECON1,2
                #DEFINE     _eeif           EECON1,4

                ORG         0               ; Vector di inizio
                goto        INIZIO
                ORG         5

tabella         movwf       PCL             ; Spostamento sulla tabella
Messag_1        equ         $
                DT          "(A) Apertura", 0x00
Messag_1_1      equ         $
                DT          "(C) Cambio codice", 0x00
Messag_2        equ         $
                DT          "Codice", 0x00
Messag_2_1      equ         $
                DT          "(C) Cancellare", 0x00
Messag_3        equ         $
                DT          "Nuovo Codice", 0x00
Messag_4        equ         $
                DT          "Confermare", 0x00
Messag_5        equ         $
                DT          "Può passare", 0x00
Messag_6        equ         $
                DT          "Accesso negato", 0x00

                INCLUDE     "tastiera.inc"
                INCLUDE     "lcd_cxx.inc"

; Delay: Questa routine realizza una temporizzazione variabile, che dipende dal valore
; dell'accumulatore nel momento in cui si legge.

Delay                       movwf       Delay_1
Delay_loop      decfsz      Delay_3,f
                goto        Delay_loop
                decfsz      Delay_2,f
                goto        Delay_loop
                decfsz      Delay_1,f
                goto        Delay_loop
                return

; Messaggio: Questa routine porta nell'LCD il messaggio il cui inizio è indicato
; nell'accumulatore

Messaggio       movwf       temp_1          ; Salva posizione della tabella
Messaggio_1     movf        temp_1,W        ; Recupera posizione della tabella
                call        tabella         ; Chiama carattere di uscita
                movwf       temp_2          ; Memorizza il valore della tabella
                movf        temp_2,f
                btfss       _z              ; Guarda se è l'ultimo
                goto        No_ultimo
                return
No_ultimo       call        LCD_DATO        ; Visualizza sull'LCD
                incf        temp_1,f        ; Carattere seguente
                goto        Messaggio_1     ; Ripete con il carattere successivo

; Controllo: Attende che siano premuti i quattro digit del codice, li memorizza
; in digit_1 .. digit_4 e visualizza — sull'LCD. Il tasto C permette di cancellare

Controllo       movlw       4
                movwf       cont_tasti      ; Inizializza contatore
                movlw       digit_1
                movwf       FSR             ; Punta all'inizio del buffer della tastiera
N_tasto         call        KEY_SCAN        ; Esplora la tastiera
                movlw       0x80
```

2

```
                subwf       N_tasto,W
                btfsc       _z              ; Qualche pulsante premuto?
                goto        N_tasto         ; NO
                call        UP_LCD          ; Riconfigura LCD
                movlw       "-"
                call        LCD_DATO        ; Visualizza il carattere -
                call        KEY_OFF         ; Genera Beep e aspetta che si liberi
                call        0x0C
                subwf       N_tasto,W
                btfsc       _z              ; Controlla se è il tasto C (Cancellare)
                goto        Si_canc
                goto        No_canc
Si_canc         bsf         temp_1,1        ; Attiva flag di cancellazione
                return

No_canc         bcf         temp_1,1        ; Disattiva il Flag di cancellazione
                movf        N_tasto,W
                movwf       INDF            ; Scrivi il tasto nel buffer
                incf        FSR,f           ; Posizione successiva del buffer
                decfsz      cont_tasti,f    ; Aggiorna contatore dei tasti
                goto        N_tasto         ; Ripete il processo
                return

; Okey: Verifica se il codice introdotto nel buffer coincide con quello della
; EEPROM. In caso affermativo il flag del carry va a 0 altrimenti va a 1

Okey            bcf         temp_1,0        ; Cancella il flag
                movlw       4
                movwf       cont_tasti      ; Numero di byte da verificare
                movlw       digit_1
                movwf       FSR             ; Primo digit
                clrf        EEADR           ; Prima posizione della EEPROM
Okey_1          call        EE_Read         ; Legge il byte della EEPROM
                movf        EEDATA,W
                andlw       0x0F
                subwf       INDF,W          ; Lo compara con il buffer
                btfss       _z              ; Controlla se è uguale
                bsf         temp_1,0        ; No, attiva il flag di errore
                incf        EEADR,f         ; Posizione successiva della EEPROM
                incf        FSR,f           ; Digit successivo
                decfsz      cont_tasti,f    ; Ripete la verifica
                goto        Okey_1
                return

; EE_Read: Legge un byte della EEPROM

EE_Read         bsf         _rp0            ; Selezione del banco 1
                bsf         _rd             ; Ordine di lettura
                bcf         _rp0            ; Selezione del banco 0

; EEWRite: Scrive un byte nella EEPROM

EE_Write        bsf         _rp0            ; Seleziona il banco 1
                bsf         _wren           ; Permesso di scrittura
                movlw       0x55
                movwf       EECON2
                movlw       0xAA
                movwf       EECON2          ; Sequenza secondo Microchip
                bsf         _wr             ; Ordine di scrittura
                bcf         _wren           ; Blocca ulteriori scritture
Wait            btfss       _eeif           ; Verifica la fine della scrittura
                goto        Wait
                bcf         _eeif           ; Resetta flag di fine scrittura
                bcf         _rp0            ; Seleziona il banco 0

INIZIO          call        UP_LCD
                call        LCD_INI         ; Inizializza LCD
                call        DISPLAY_ON_CUR_OFF

                movlw       b'00011000'
                movwf       PORTA           ; Scollega relè e cicalino

                clrf        EEADR           ; Posizione iniziale della EEPROM
                call        EE_Read         ; Legge byte dalla EEPROM
                movlw       0xff
                subwf       EEDATA,W
                btfss       _z              ; Verifica se è utilizzato
                goto        Si_usata

; Carica nella EEPROM il codice 1234 per default
```

Programma risolto.

3

```
No_usata    clrf    EEDATA
            movlw   4           ; Numero di byte da scrivere
            movwf   temp_1
No_usata_1  incf    EEDATA,f    ; Codice da scrivere
            call    EE_Write    ; Scrivi dato
            incf    EEADR,f
            decfsz  temp_1,f
            goto    No_usata_1

Si_usata    call    LCD_INI     ; Inizializza LCD
            movlw   Messag_1    ; Offset del messaggio 1
            call    Messaggio   ; Visualizza "(A) Apertura"
            movlw   0xC0
            call    LCD_REG     ; Regola posizione del messaggio
            movlw   Messag_1_1  ; Offset del messaggio 1.1
            call    Messaggio   ; Visualizza "(C) Cambio Codice"
No_tast     call    KEY_SCAN    ; Esplora la tastiera
            movlw   0x80
            subwf   N_tasto,W
            btfsc   _z          ; C'è il tasto
            goto    No_tast
            call    KEY_OFF     ; Genera beep e attende che si liberi
            movlw   0x0A
            subwf   N_tasto,W
            btfsc   _z          ; E' il tasto A (Apertura)?
            goto    Si_A
No_A        movlw   0x0C
            subwf   N_tasto,W
            btfss   _z          ; E' il tasto C (Cambio codice)?
            goto    No_tast

Si_C        call    UP_LCD      ; Riconfigura LCD
            call    LCD_INI     ; Aggiorna LCD
            movlw   Messag_2_1  ; Offset del messaggio 2.1
            call    Messaggio   ; Visualizza "(C) Cancellare"
            movlw   0xC0
            call    LCD_REG     ; Riposiziona il cursore dell'LCD
            movlw   Messag_2    ; Offset del messaggio 2
            call    Messaggio   ; Visualizza "Codice"
            call    Controllo   ; Aspetta che si introduca un codice valido

            btfsc   temp_1,1    ; Cancellazione avvenuta?
            goto    Si_usata    ; Si

            call    Okey        ; Verifica se è corretta
            btfsc   temp_1,0    ; E' valida?
            goto    Si_usata    ; No, torna all'inizio
            call    UP_LCD      ; Si, riconfigura LCD
            call    LCD_INI     ; Cancella e inizializza LCD
            movlw   Messag_2_1  ; Offset del messaggio 2.1
            call    Messaggio   ; Visualizza "(C) Cancella"
            movlw   0xC0        ; Riposiziona LCD
            call    LCD_REG
            movlw   Messag_3    ; Offset del messaggio 3
            call    Messaggio   ; Visualizza "Nuovo Codice"
            call    Controllo   ; Attende il nuovo codice
            btfsc   temp_1,1    ; Cancellazione avvenuta?
            goto    Si_usata    ; Si

            call    UP_LCD      ; Riconfigura LCD
            call    LCD_INI     ; Aggiorna LCD
            movlw   Messag_2_1  ; Offset del messaggio 2.1
            call    Messaggio   ; Visualizza "(C) Cancellare"
            movlw   0xC0        ; Riposiziona LCD
            call    LCD_REG     ;
            movlw   Messag_4    ; Offset del messaggio 4
            call    Messaggio   ; Visualizza "Confermare"
            movf    digit_1,W
            movwf   di_tem_1
            movf    digit_2,W
            movwf   di_tem_2
            movf    digit_3,W
            movwf   di_tem_3
            movf    digit_4,W
            movwf   di_tem_4    ; Salva in modo temporaneo 1° codice

            call    Controllo   ; Legge il secondo codice

            btfsc   temp_1,1    ; Cancellazione avvenuta?
            goto    Si_usata    ; Si
```

4

```
            movf    digit_1,W
            subwf   di_tem_1,W  ; Verifica il 1° digit
            btfss   _z          ; Uguale?
            goto    Si_usata    ; No

            movf    digit_2,W
            subwf   di_tem_2,W  ; Verifica il 2° digit
            btfss   _z          ; Uguale?
            goto    Si_usata    ; No

            movf    digit_3,W
            subwf   di_tem_3,W  ; Verifica il 3° digit
            btfss   _z          ; Uguale?
            goto    Si_usata    ; No

            movf    digit_4,W
            subwf   di_tem_4,W  ; Verifica il 4° digit
            btfss   _z          ; Uguale?
            goto    Si_usata    ; No

; Scrive nella EEPROM il nuovo codice

            movlw   4
            movwf   cont_tasti  ; Numero di byte da scrivere
            movlw   digit_1
            movwf   FSR         ; Indice dei digit
            clrf    EEADR       ; Prima posizione della EEPROM
Altro_digit movf    INDF,W      ; Carica digit
            movwf   EEDATA
            call    EE_Write    ; Scrive nella EEPROM
            incf    FSR,f       ; Digit successivo
            incf    EEADR,f     ; Posizione successiva della EEPROM
            decfsz  cont_tasti,f
            goto    Altro_digit
            goto    Si_usata

Si_A        movlw   3
            movwf   cont_err    ; Stabilisce il numero dei tentativi

Altro_ancora call   UP_LCD      ; Riconfigura LCD
            call    LCD_INI     ; Aggiorna
            movlw   Messag_2_1  ; Offset del messaggio 2.1
            call    Messaggio   ; Visualizza "(C) Cancella"
            movlw   0xC0
            call    LCD_REG     ; Riposiziona il cursore dell'LCD
            movlw   Messag_2    ; Offset del messaggio 2
            call    Messaggio   ; Visualizza "Codice"
            call    Controllo   ; Attende che si introduca il codice

            btfsc   temp_1,1    ; Cancellazione avvenuta?
            goto    Si_usata    ; Si

            call    Okey        ; Verifica se è corretta
            btfsc   temp_1,0    ; E' valida?
            goto    No_Ok       ; No
            call    UP_LCD      ; Riconfigura LCD
            call    LCD_INI     ; Aggiorna
            movlw   Messag_5    ; Offset del messaggio 5
            call    Messaggio   ; Visualizza "Può passare"
            bcf     PORTA,4     ; Si attiva il relè
            bcf     PORTA,3     ; Attiva beep
            movlw   0x0A
            call    Delay       ; Temporizza 2 sec.
            bsf     PORTA,4     ; Disattiva il relè
            bsf     PORTA,3     ; Disattiva beep
            clrf    cont_err    ; Pone a zero il contatore di tentativi
            goto    Si_usata    ; Ripete il processo

No_Ok       decfsz  cont_err,f  ; Prova fallita
            goto    Altro_ancora ; Ripete un altro tentativo
            call    UP_LCD      ; Riconfigura LCD
            call    LCD_INI     ; Aggiorna
            movlw   Messag_6    ; Offset del messaggio 6
            call    Messaggio   ; Visualizza "Accesso negato"
            movlw   0x10
            call    Delay       ; Temporizza
            goto    Si_usata    ; Ripete il processo

DISPLAY_ON_CUR_OFF
            movlw   b'00001100' ; LCD on cursore off
            call    LCD_REG
            return

            END
```

Programma risolto (continuazione).

Strumenti di lavoro: caratteristiche avanzate di MPLAB (II)

Per concludere questa sezione dedicata a MPLAB, approfondiremo alcuni strumenti con cui abbiamo già lavorato e vedremo così altre possibilità previste in questo ambiente che facilitano la programmazione dei microcontroller PIC.

MPASM COME GENERATORE DI MODULI

Conosciamo MPASM come un assemblatore che trasforma il codice sorgente che intende l'utente, in un codice adeguato al microcontroller. Però questo assemblatore usato con l'MPLINK, può generare moduli precedentemente compilati e situati in differenti parti del programma. Il risultato di scrivere un codice oggetto e di assemblarlo come un modulo per unirlo successivamente ad altri, o creare direttamente un file HEX eseguibile, è leggermente differente.

MODULI DI CODICE

Possiamo definire come moduli diverse cose. Quando sono istruzioni della memoria di codice dovranno essere precedute dalla dichiarazione di una sezione di codice come mostrato nella figura. In alcuni casi, è necessario specificare la situazione fisica che occuperà il codice, come ad esempio con il vettore di interrupt.

MODULI DEI DATI

Anche i dati della memoria RAM possono dividersi in differenti sezioni, disposte in cinque tipi differenti: UDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR e IDATA. Nell'esempio si mostra come si possa operare una dichiarazione dei dati nel modo in cui

```
Codice assoluto:

Start       CLRW
            OPTION
            :

Codice nei moduli:

CODE
Start       CLRW
            OPTION
            :
```

Codice definito come modulo all'interno di un programma.

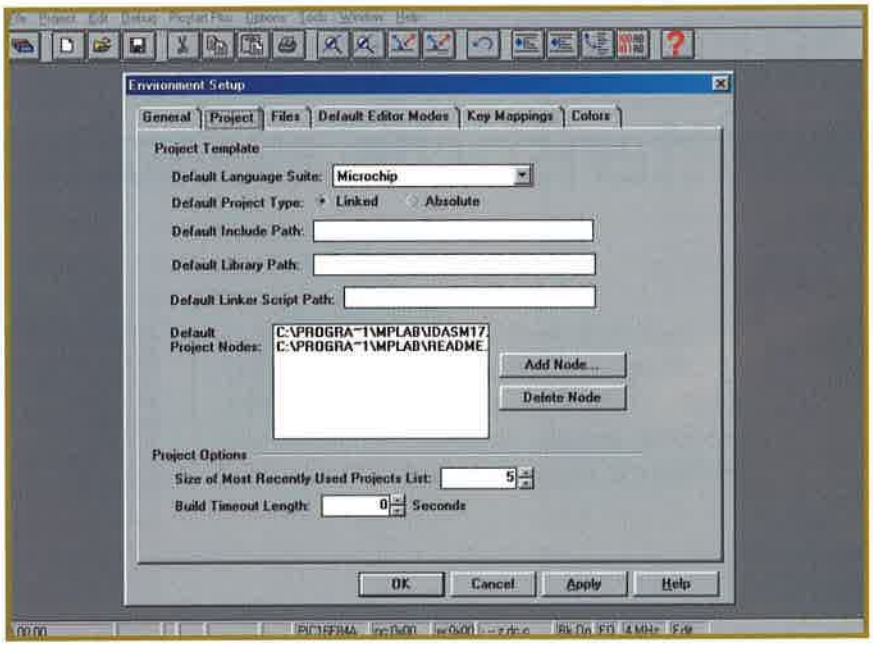
```
Dati assoluti:

CBLOCK      0x20

            InputGain, OutputGain   ; Controllo di ciclo
            HistoryVector           ; Si deve inizializzare a 0
            Temp1, Temp2, Temp3     ; Variabili temporali
ENDC

Dati in sezioni:

            IDATA                   ; Dati inizializzati
HistoryVector DB 0

            UDATA                   ; Dati non inizializzati
InputGain   RES 1
OutputGain  RES 1

            UDATA_OVR               ; Dati temporali
Temp1       RES 1
Temp2       RES 1
Temp3       RES 1
```

Dati definiti in sezioni all'interno di un programma.

siamo abituati e nelle differenti sezioni, a seconda dell'uso che vogliamo fare di ognuno, in modo che il programma li gestisca in modo differente. Per ognuna delle sezioni si usa un tipo di direttiva, per esempio nella IDATA sono permesse le direttive DB, DW e DATA.

ESPORTAZIONE E IMPORTAZIONE DI MODULI

Quando vogliamo che un'etichetta definita in un modulo o routine si utilizzi in un altro, la si deve esportare utilizzando la direttiva GLOBAL, per ciò è necessario averla dichiarata in precedenza. A loro volta, i moduli che utilizzano etichette dichiarate in altri moduli dovranno essere importati con la direttiva EXTERN. Dato che si lavora con differenti banchi di dati, dobbiamo tenere presente in quale banco si alloggia ognuna delle variabili prima di fare riferimento ad esse. Questo compito però può essere demandato all'assembler utilizzando la direttiva BANKSEL seguita dal nome della variabile prima di fare qualcosa con essa, perché sia l'assembler ad occuparsi di cambiare banco.



Per poter utilizzare moduli in un progetto dobbiamo indicarlo in MPLAB.

Definizioni delle variabili globali in un modulo:

```
                        UDATA
InputGain       RES 1
OutputGain      RES 1
                        GLOBAL InputGain, OutputGain
Filter
                        Global Filter
                :       ; Filter code
```

Utilizzo di variabili definite in un altro modulo:

```
                EXTERN  InputGain, OutputGain, Filter
                UDATA
Reading         RES 1
                CODE
                ....
                MOVLW   GAIN1
                MOVWF   InputGain
                MOVLW   GAIN2
                MOVWF   OutputGain
                MOVF            Reading,W
                CALL            Filter
```

Dati e routine definiti in modo generale per essere utilizzati in altri moduli.

L'MPASM CON MPLINK E MPLIB

Una volta editato il modulo in oggetto, per assemblarlo si seleziona l'opzione "Object File" e il file avrà estensione .obj.

Il programma MPLINK si incaricherà di radunare e posizionare tutti i moduli realizzati, generando l'eseguibile. Il vantaggio di un progetto realizzato in questo modo è la riutilizzazione del codice con il conseguente risparmio di tempo e risorse.

Da parte sua, MPLIB aiuta anche il mantenimento del codice, visto che permette di fare librerie.

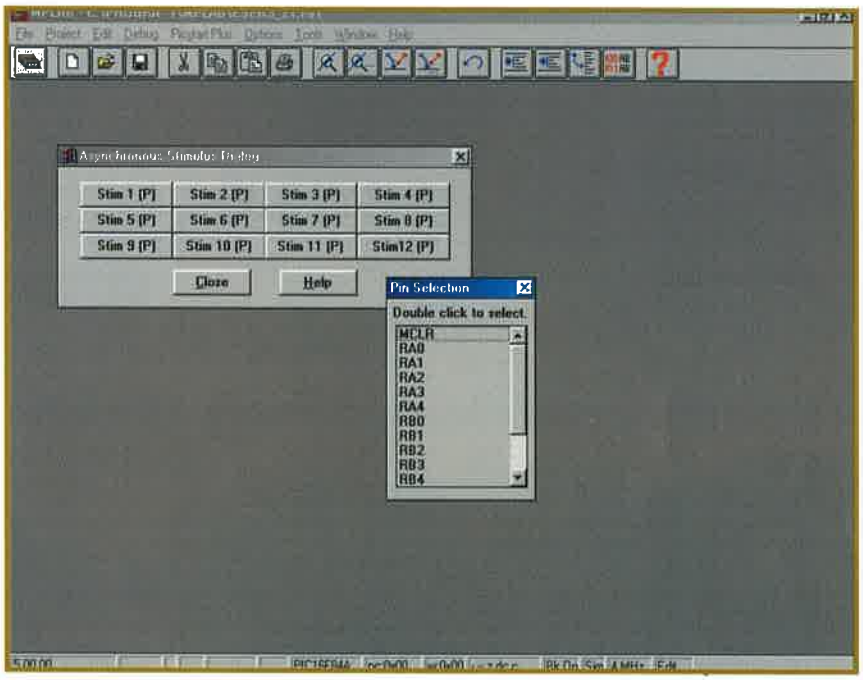
Una libreria è una collezione di moduli object, normalmente con alcune caratteristiche comuni riunite in un unico file, come per esempio, la libreria "math.lib" per il lavoro con operazioni matematiche.

Per poter utilizzare in un progetto differenti moduli di questo tipo al posto di uno solo, lo dovremo indicare come opzione, in Option >Environment Setup, e all'interno di questo, nella scheda Project.

Strumenti di lavoro: caratteristiche avanzate dell'MPLAB-SIM I

Data l'importanza che riveste il provare un programma dopo la sua compilazione e prima di programmarlo direttamente sul microcontroller; anche se abbiamo già fatto i primi passi in questo settore, crediamo che il simulatore MPLAB-SIM, all'interno degli strumenti supportati dall'ambiente di sviluppo MPLAB, meriti un discorso a parte. Sino ad ora abbiamo eseguito un programma, abbiamo introdotto differenti valori sulle linee di I/O e abbiamo guardato i risultati sui registri corrispondenti. In definitiva abbiamo verificato che i programmi andassero bene, e che facessero quello che ci si attendeva da loro. Però, avete provato voi a fare alcune delle modifiche che vi abbiamo proposto?

E in caso affermativo, che cosa è successo quando il programma non ha funzionato? Avete trovato la causa?



Dobbiamo chiamare ogni linea di ingresso/uscita come propone il simulatore.

A volte questo è difficile, perché è necessario conoscere le caratteristiche avanzate del simulatore che ci aiutano nel nostro compito.

Inizieremo vedendo quelle di cui dobbiamo tenere conto al momento di lavorare con il simulatore e che dipendono esclusivamente da questo, in altre parole, quello che ci permette di fare e che restrizioni presenta MPLAB-SIM.

VELOCITÀ DI ESECUZIONE

L'MPLAB-SIM simula l'esecuzione di un microcontroller e quello degli ingressi/uscite, a una velocità che dipende dal computer su cui gira il programma. Di conseguenza, per lo stesso programma si devono considerare differenti opzioni, per fare in modo che la simulazione non risulti né troppo rapida, né troppo lenta.

PIN INGRESSI/USCITE

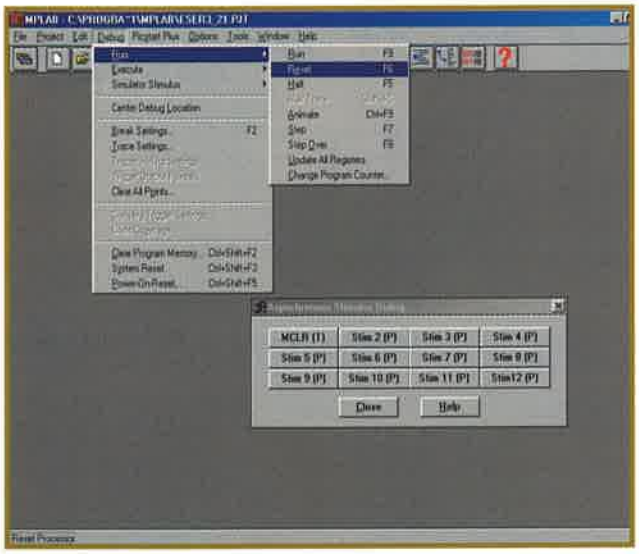
MPLAB-SIM può lavorare con molti e diversi microcontroller, fra cui il PIC16F84 e, a seconda del dispositivo che si sta utilizzando, dobbiamo tenere conto di diverse cose.

Alcuni microcontroller, ad esempio, hanno multiplexate le loro linee di ingressi/uscite, in modo che ognuna realizzi diverse funzioni e per ognuna di esse c'è un nome.

Per modificare queste linee, sia direttamente che con l'opzione di introdurre stimoli esterni, si devono utilizzare i nomi proposti per esse, altrimenti il simulatore non le riconoscerà.

INTERRUPT

Nel caso del PIC16F84 il simulatore supporta tutti gli interrupt overflow del TMR0, cambio di 4 linee di maggior peso della porta B, interrupt esterno per RB0/INT e



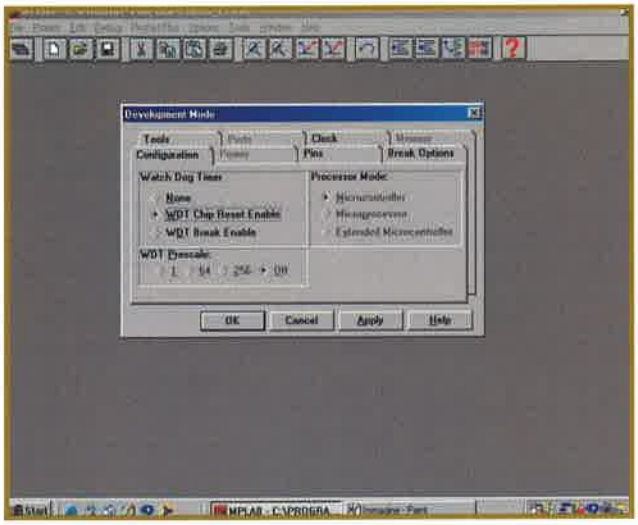
Sono contemplate diverse cause di reset.

termine della scrittura della EEPROM. Questo permetterà di utilizzare la risorsa di "stato di riposo", da cui si esce con un interrupt.

RESET

Il simulatore inoltre supporta tutte le condizioni di reset.

Il reset tramite la linea MCLR si può simulare con l'introduzione di un livello basso, seguito da uno alto, nell'opzione di stimoli esterni, o direttamente nell'opzione del reset dentro l'opzione Debug>Run. Inoltre abbiamo il reset del sistema e il reset per connessione dell'alimentazione.



Per utilizzare il Watchdog dobbiamo attivarlo come si farebbe per la scrittura del PIC.

WATCHDOG

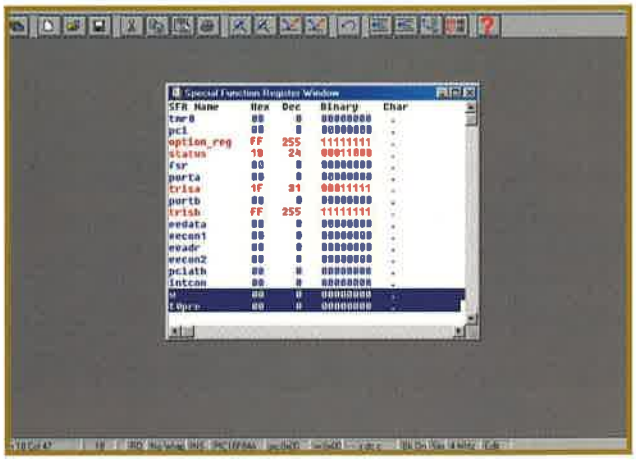
Anch'esso è simulato nell'MPLAB-SIM, sempre e quando sia stato attivato in Options>Development Mode.

REGISTRI SPECIALI

Lavorando con i PIC ci sono alcune risorse che, anche se devono essere prese in considerazione, non dispongono di una posizione all'interno della finestra dei registri, poiché non sono registri di per sé, bensì bit all'interno dei registri.

Si dispone di questi tramite la finestra di registro di funzioni speciali, come il TOPRE, per rappresentare il prescaler del TMR0.

Il registro di lavoro W è un caso speciale che si trova in questa finestra.



Ci sono alcune risorse che si possono solo osservare all'interno della finestra dei registri di funzioni speciali.

PERIFERICHE

Oltre alle linee di ingresso/uscita, all'interno del simulatore sono disponibili le seguenti periferiche:

- **Timer0.**

Questa periferica, insieme agli interrupt che provoca, è completamente implementata nel simulatore, potendosi incrementare con il clock interno o mediante impulsi esterni. In quest'ultimo caso il tempo in cui la linea rimane a 1 e a 0, deve essere di almeno un ciclo, per essere tenuta in conto.

- **Memoria dei dati EEPROM.**

Anche questa risorsa è simulata, insieme ai suoi cicli di lettura e scrittura. È approssimativamente di 10 ms. Le funzioni dei bit di controllo WRERR e WREN sono anch'esse simulate.

Strumenti di lavoro: caratteristiche avanzate del MPLAB-SIM II

Se avete letto la scheda precedente, sapete già cosa potete o non potete aspettarvi dal simulatore MPLAB-SIM. Vediamo ora come sfruttarlo al meglio, a seconda delle nostre esigenze e in ogni occasione. Ricordate che per avere accesso agli strumenti di simulazione, è necessario cambiare l'ambiente di sviluppo.

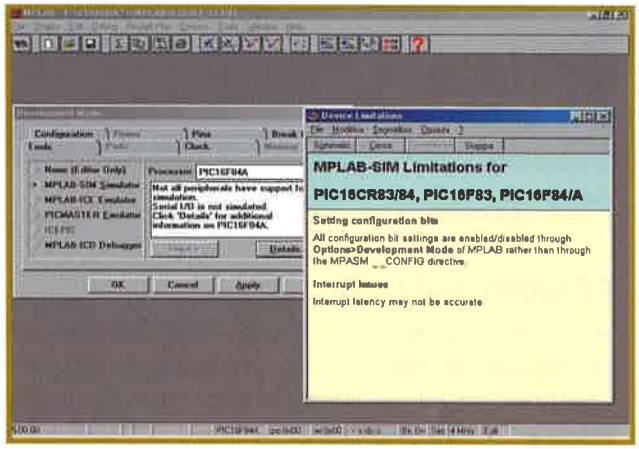
Da questa stessa finestra si potranno inoltre verificare se esistono delle limitazioni nella simulazione del dispositivo scelto, oltre ad ottenere "dettagli" sul medesimo.

Se non si cambia l'ambiente di sviluppo, le opzioni di simulazione appariranno disattivate.

MENÙ DEBUG

Il menù Debug contiene tutte le opzioni necessarie, quando si sta simulando un programma per la sua messa a punto. Le opzioni più importanti sono rappresentate anche in una barra di icone come quella riportata in figura. Questa barra si ottiene cliccando sull'icona di sinistra della videata. Quando si passa sulle icone con il mouse, sulla linea inferiore di MPLAB appare il commento per ognuna di esse.

Se non si cambia l'ambiente di sviluppo le opzioni della simulazione appaiono disattivate.

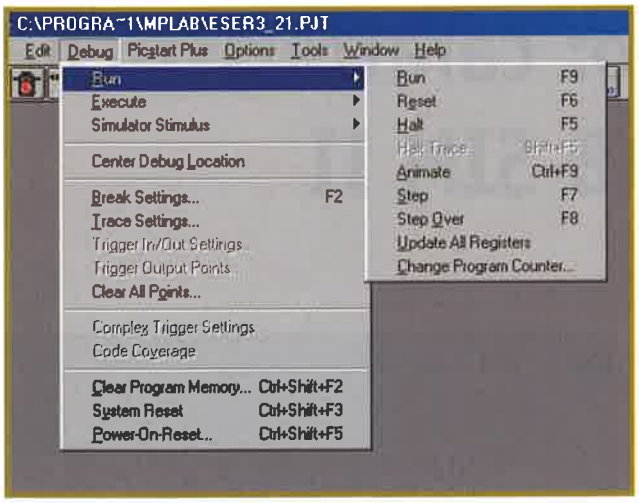


Cambiando l'ambiente di simulazione si possono chiedere "dettagli" del dispositivo.

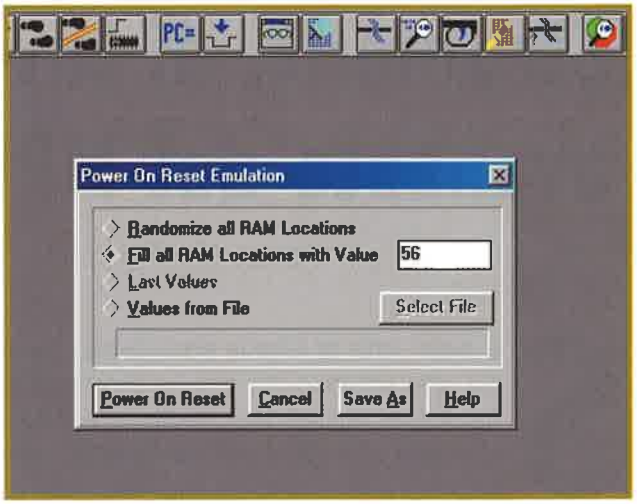
OPZIONE RUN

L'opzione Run permette di controllare l'esecuzione del programma. Si può eseguire in modo continuo (Run) sino a che non si trova un "punto di rottura" o si clicca l'opzione di fermata (Halt), evidenziando i valori dei registri durante l'esecuzione (Animate), o istruzione a istruzione (Step). Si può inoltre eseguire una subroutine da sola, e fermarsi subito dopo (Step Over). Per fermare il processore si può usare l'opzione Halt con cui si visualizzeranno le informazioni delle finestre corrispondenti. Un'altra opzione di fermata è Halt Trace, che fermerà il "tracciato" del programma, però questo continuerà ad eseguirsi. I registri si possono inoltre rendere visibili con Update All Registers. Per iniziare nuovamente si può scegliere l'opzione Reset, che provoca una reinizializzazione del processore, come se si premesse il pulsante MCLR, oppure iniziare da un punto preciso, altrimenti può essere interessante cambiare il valore del PC (Change Program Counter).

Opzioni più importanti del menù Debug.



Opzioni del menù Run.



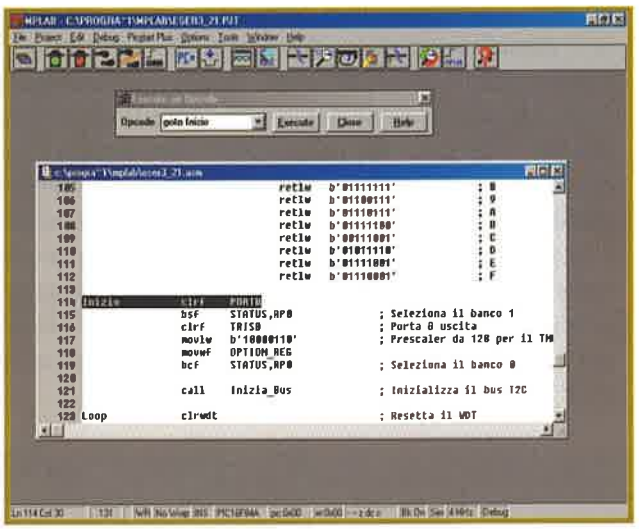
Si possono scegliere i valori dei registri dopo un reset.

OPZIONE EXECUTE

Al momento di eseguire il programma, potrebbe essere interessante non farlo girare completamente, ma verificarlo fino ad una particolare istruzione (Execute an Opcode), o eseguirlo fino al verificarsi di una determinata condizione (Conditional Break).

OPZIONE SIMULATOR STIMULUS

Oltre a generare stimoli per il simulatore in modo interattivo (Asynchronous Stimulus), si possono descrivere i segnali che arrivano ai pin di ingresso come file di testo (Pin Stimulus), o utilizzare i file di testo per introdurre valori da 8 bit direttamente sui registri (Register Stimulus). Se gli impulsi devono essere periodici, come ad esempio un treno di impulsi, si utilizzerà l'opzione Clock Stimulus.



Può essere interessante eseguire una sola istruzione.

BREAK POINT E TRACE POINT

I punti di rottura (Break Point) e i punti di traccia (Trace Point) sono due elementi fondamentali per mettere a punto un programma. La definizione di un punto di rottura permette di combinare i diversi modi di esecuzione, in modo che, ad esempio, il programma venga eseguito in modo rapido (Run) sino ad un certo punto di rottura, in cui il programma si ferma e si può simulare passo a passo. Se inoltre si definisce un range di indirizzi di programma, si può sapere in ogni momento che valori hanno i registri quando sono eseguite quelle istruzioni. Entrambe le opzioni permettono di visualizzare lo stato del processore, quando si stanno cercando possibili cause di un mal funzionamento del programma. Queste funzioni si possono annullare facilmente con Clear All Points.

ALTRE OPZIONI

Esistono altre opzioni che possono essere utili in determinati momenti; ad esempio simulare la cancellazione del microcontroller con Clear Program Memory, resettare il sistema completo come se iniziassimo in quel momento a lavorare con MPLAB (System Reset), o generare un reset che ponga tutti i registri con un valore determinato o casuale, a seconda di cosa si sceglie. L'opzione Center Debug Location permette di eseguire un salto nella simulazione del programma, collocando il contatore di programma alla metà del codice che si sta verificando.