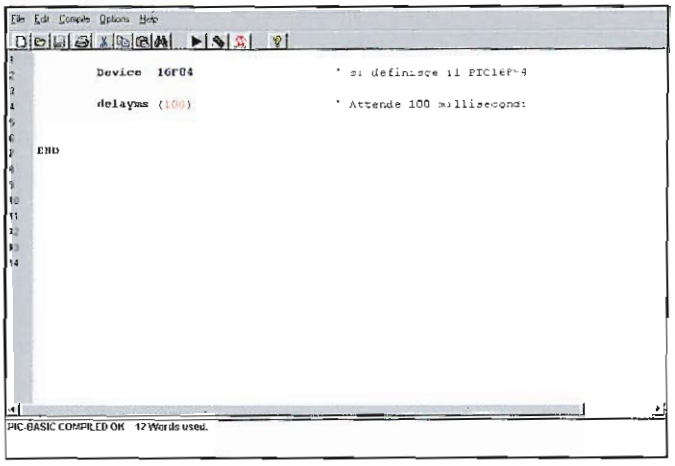
come questo, bisogna utilizzare direttamente un'istruzione assembler, come si può vedere nella figura, dato che ognuna di esse impiega, a essere eseguita, solo 1µs a 4 MHz.

Tempo in millisecondi

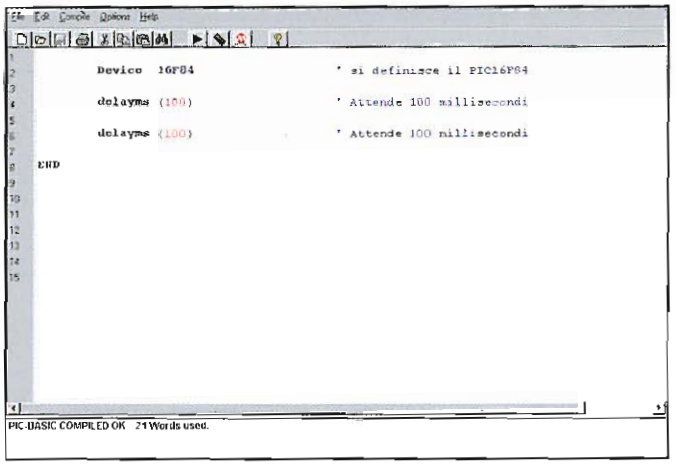
Esiste un'istruzione simile a "delayus", che invece di contare i microsecondi conta i millisecondi, permettendo così di realizzare temporizzazioni più lunghe. I limiti inferiori e superiori sono gli stessi, così come il modo di utilizzarli. Anche qui si suppone di avere un quarzo da 10 MHz, rispetto al quale dovremo fare la regolazione del parametro per ottenere gli stessi risultati a velocità diverse.

Tuttavia quella caratteristica ricordata in precedenza, per cui un'istruzione si scompone in diverse sottoistruzioni durante il processo di assemblamento, diventa ancora più palese con questa nuova istruzione. Come si può vedere nelle due figure in alto, ogni utilizzo dell'istruzione "delayms" dopo la procedura di assemblamento e di compilazione occupa 9 posizioni.

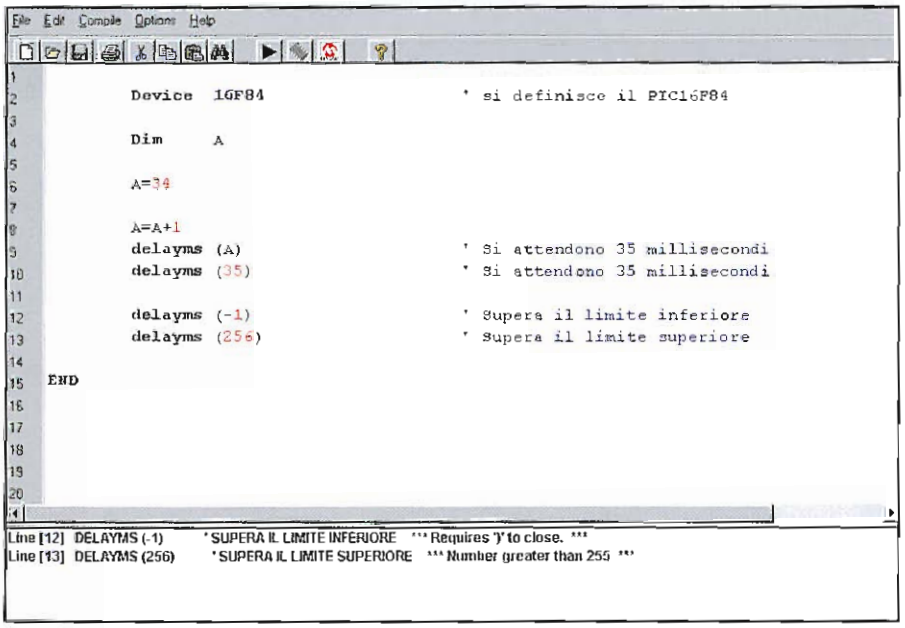
Questo rende molto oneroso, in termini di consumo di memoria di programma, l'utilizzo ripetuto di queste istruzioni, perché la memoria è già scarsa di per sé.
La soluzione sta nell'inserire l'istruzione in una subroutine, consumando il minimo indispensabile di memoria, ma potendola utilizzare tutte le volte che sarà necessario. Provate a



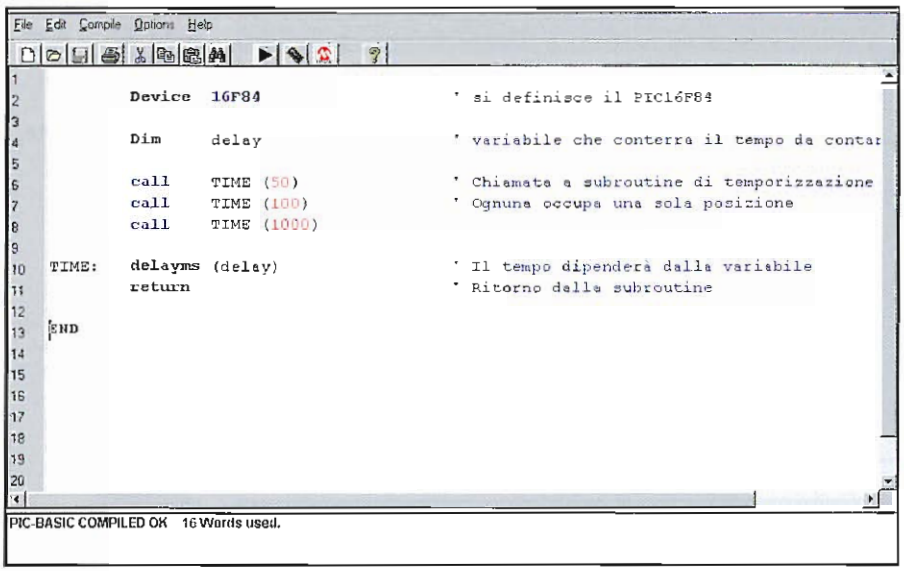
L'utilizzo di una singola istruzione "delayms" occupa 9 byte, più i 3 di apertura e chiusura.



Ogni ulteriore istruzione "delayms" occupa 9 byte aggiuntivi.



La nuova istruzione è simile a quella vista in precedenza.



L'utilizzo delle subroutines permette di risparmiare spazio nel caso di uso ripetuto dell'istruzione "delayms".

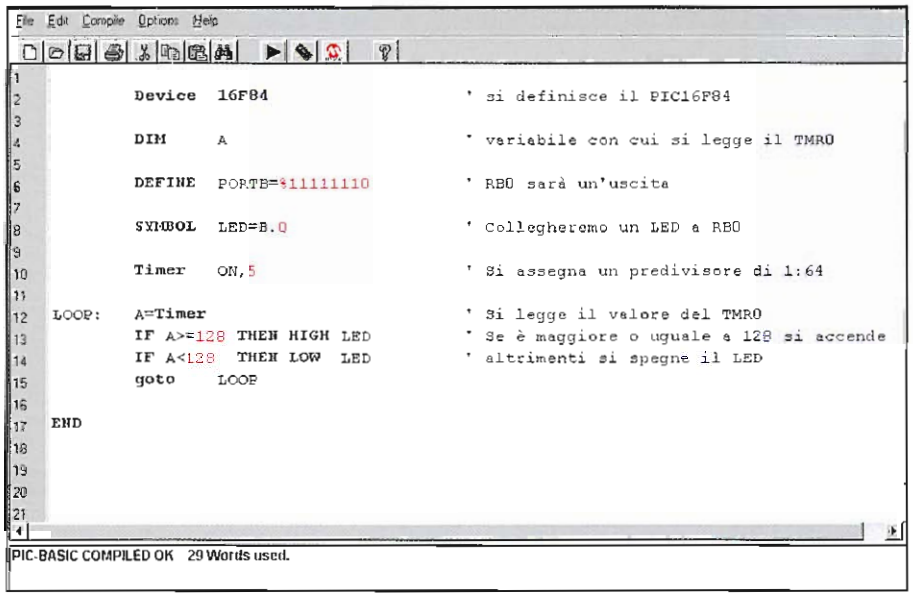
inserire nel programma della figura più istruzioni "call" e vedrete che ognuna di esse occupa solo una posizione, anche se il risultato sarà lo stesso che eseguire direttamente l'istruzione "delayms". Per contro, il tempo di esecuzione sarà maggiore, dato che bisognerà andare e tornare dalla subroutine, cosa che consuma cicli di clock.

In uno dei prossimi capitoli spiegheremo nel dettaglio l'utilizzo delle subroutines.

Il TMR0

Finora abbiamo utilizzato istruzioni per la temporizzazione "prefabbricate", che ci permettono di realizzare temporizzazioni senza doverci preoccupare delle operazioni realizzate internamente al PIC.

Tuttavia abbiamo a disposizione anche un'altra istruzione, che si avvicina di più al risultato del processo di assemblamento (come sicuramente ricordate, questo processo trasforma le istruzioni complesse del LetPicBasic in istruzioni assembler), si tratta di un passaggio intermedio, stiamo infatti parlando dell'istruzione "timer". Con il primo parametro di questa istruzione si abilita (ON) e si disabilita (OFF) il temporizzatore del PIC, il TMR0, mentre con il secondo gli si assegna un "predivisore di frequenza". Questo secondo parametro può variare da 0 a 7, assegnando un predivisore basato sulla potenza del 2, da 1:2 (per il valore 0) a 1:256 (per il valore 7). Quando si abilita con "on" il TMR0, lo si inizializza a 0 e questo inizia a contare a una velocità che dipenderà dal quarzo; per un quarzo del valore di 4 MHz si incrementerà ogni microsecondo. Il predivisore di frequenza permette di far incrementare più lentamente il timer, moltiplicando il tempo di ritardo per 2, 4, 8, 16 ecc. a seconda del valore assegnato al predivisore stesso. In qualsiasi momento è possibile acquisire il valore del TMR0 per controllare il tempo all'interno di un programma.



L'istruzione "timer" ci avvicina un po' di più alle istruzioni assembler.