

1. Introduzione

Il manuale PicBasic Pro (o PBP) è il linguaggio di programmazione della prossima generazione, che ti permette di programmare la famiglia μ PIC con tecnologia Microchip, in modo veloce e facile. Il linguaggio BASIC è molto più semplice da leggere e scrivere rispetto al complicato linguaggio assembler della Microchip. Il manuale PicBasic Pro è della generazione "BASIC II" e ha, per la maggior parte, terminologie e funzioni sia del BASIC I sia del BASIC II. Essendo un vero e proprio compilatore, i programmi si realizzano molto velocemente e più a lungo nel tempo in confronto ai loro Marchi equivalenti. PBP non è altrettanto compatibile con i Marchi BASIC come il suo originale Compilatore PicBasic lo è con il BS1. Furono prese decisioni al riguardo che speriamo migliorino il linguaggio per tutti. Una di tali decisioni fu di aggiungere un **IF..THEN..ELSE..ENDIF** invece di **IF..THEN (GOTO)** dei Marchi. Tali differenze sono spiegate più tardi nel nostro manuale. PBP si propone di creare files che girano su un PIC16F84-04/P con frequenza a 4MHz. Sono necessarie solo un minimo di altre parti: 2 condensatori da 22pf per il cristallo a 4MHz, una resistenza pull-up 4.7K allacciata al pin /MCLR e una tensione di alimentazione a 5 Volt. Molti μ PIC, oltre ai 16F84, con frequenza oltre 4MHz, potrebbero trovare impiego con il Compilatore PicBasic Pro.

1.1 I MicroPIC

Il Compilatore PicBasic Pro produce codici che possono essere programmati in una vasta varietà di μ PIC aventi da 8 a 44 pin e varie caratteristiche on-chip che includono convertitori A/D, hardware timers e porte seriali. Ci sono alcuni μ PIC che non lavoreranno con il Compilatore PicBasic Pro, le serie PIC16C5x includono PIC16C54-16C58 e PIC12C5xx. Questi μ PIC sono basati sull'originale centro 12-bit piuttosto che sui più correnti 14- e 16-bit. Il Compilatore PicBasic Pro richiede alcune delle caratteristiche disponibili solo sui centri 14- e 16-bit. Ci sono molti μ PIC, alcuni con pin compatibili con le serie '5x, che potrebbero essere utilizzati con il Compilatore PicBasic Pro. Attualmente, la

Compilatore PicBasic Pro

lista include le apparecchiature delle serie PIC12C67x, PIC14C000, PIC16C55x, 6xx, 7xx, 84, 9xx, PIC16F62x, 8x, PIC17Cxxx e PIC18Cxxx, con Microchip aggiungendo un rate ancor più rapido. Per una sostituzione diretta di un PIC16C54, 56 o 58, i PIC16C554, 558, 620, 621 e 622 lavorano bene con il compilatore e quasi allo stesso prezzo.* Per la lista di supporto più recente μ PIC vedi il file READ.ME.

*Il prezzo di vendita è stabilito da Microchip Technology Inc. e i suoi distributori.

Nonostante il proposito generale di sviluppo del μ PIC, usando il Compilatore PicBasic Pro, i Pic 16F84, 16F876 e 16F877 sono i correnti μ PIC di scelta. Questi microcontrollori usano tecnologie flash per permettere cancellazioni e riprogrammazioni atte a velocizzare programmi di correzione. Con il tasto del mouse nella programmazione software, il μ PIC può essere istantaneamente cancellato e riprogrammato più volte. Gli altri μ PIC, nelle serie 12C67x, 14C000, 16C55x, 16C6xx, 16C9xx, 17Cxxx e 18Cxxx sono programmabili one-time (OTP), e inoltre hanno una finestra al quarzo nel top (JW) che permette cancellazioni attraverso lunghe esposizioni alla luce ultravioletta.

I dispositivi PIC16F84 e 16F87x contengono dai 64 ai 256 bytes di memoria eeprom non volatile, che può essere usata per immagazzinare i dati dei programmi e altri parametri anche quando ci si trova in POWER OFF (spento). Si può accedere in questa data area semplicemente usando i comandi READ e WRITE del Compilatore PicBasic Pro. Il codice di programma è permanentemente immagazzinato in quello del μ PIC, sia quando ci si trova in POWER ON che POWER OFF.

Attraverso l'uso di un μ PIC flash per testare il programma iniziale, il processo di correzione software potrebbe essere mandato avanti. Una volta che le principali operazioni di programmazione sono state eseguite in modo soddisfacente, può essere usato un μ PIC con più capacità o con caratteristiche del compilatore.

Sebbene molte caratteristiche del μ PIC siano discusse in questo manuale, per informazioni più complete, è necessario ottenere i data sheets appropriati o il CD-ROM da Microchip Technology. Fai riferimento all'Appendice B per le informazioni riguardanti chi contattare.

1.2 Notizie su questo manuale

Questo manuale non è un trattato completo del linguaggio BASIC. Descrive le istruzioni per inserire il PicBasic Pro e fornisce esempi su come usarlo. Se non hai confidenza con la programmazione BASIC, dovrai acquistare un libro sull'argomento. Altrimenti, salta dentro. BASIC è concepito come un linguaggio facile da usare; ci sono programmi esplicativi aggiuntivi, sul disco e sul sito web, che possono aiutarti ad iniziare.

La prossima sezione di questo manuale riguarda l'installazione del Compilatore PicBasic Pro e l'inserimento del tuo programma. A seguire c'è una sezione che descrive le differenti opzioni dei programmi.

I fondamentali della programmazione sono spiegati nei prossimi capitoli, seguiti da una sezione di riferimento in cui è preso in dettaglio ogni comando del PicBasic Pro. La sezione di riferimento mostra ogni prototipo dei comandi, una descrizione del comando stesso e alcuni esempi. Le parentesi graffe { } indicano parametri opzionali.

Il resto del manuale fornisce informazioni per programmatori di livello avanzato – tutti i funzionamenti interni del compilatore.

2. Si comincia

2.1. Installazione del software

I files del PicBasic Pro sono compressi in un self-extracting file sul dischetto. Devono essere decompressi sul tuo hard drive prima dell'uso.

Per decomprimere i files, crea una subdirectory sul tuo hard drive che chiamerai PBP o con n altro nome a tua scelta, digitando:

```
md PBP
```

al DOS prompt. Cambia alla directory:

```
cd PBP
```

Supponendo che il dischetto si trovi nel drive a:, decomprimi i files nella subdirectory PBP:

```
a:\pbpxxx -d
```

in cui xxx è la serie numerica del compilatore sul disco. Non dimenticare l'opzione -d alla fine del comando. Questo ti garantisce che sono state create le subdirectory nel PBP.

Assicurati che almeno 50 FILES e BUFFERS siano stati predisposti nel tuo CONFIG.SYS file. Secondo quanti FILES e BUFFERS siano già presenti nel tuo sistema, potrebbe essere necessario un numero ancora più elevato di allocazioni.

Per avere più ampie informazioni sulla decompressione dei files, vedi il file README.TXT. Per le informazioni più recenti sul Compilatore PicBasic Pro, leggi anche il file READ.ME che è decompresso nella subdirectory PBP sul tuo hard drive.

2.2. Il tuo primo programma

Per l'operazione del Compilatore PicBasic Pro avrai bisogno di un editor di testo o di un word processor per creare il file del tuo programma, qualcosa come un programmatore μ PIC tipo il nostro Programmatore μ PIC EPIC Plus Pocket, e lo stesso Compilatore PicBasic Pro. Naturalmente hai anche bisogno di un PC per lavorarci su.

Compilatore PicBasic Pro

La sequenza delle fasi è descritta come segue:

Primo, crea il source file BASIC per il programma usando il tuo word processor preferito. Se non ne hai uno, DOS EDIT (incluso in MS-DOS) o WINDOW NOTEPAD (incluso in Windows e Windows 95/98) possono sostituirlo. Il nome del source file dovrebbe (ma non è richiesto) finire con l'estensione .BAS.

Il text file che è stato creato deve essere un puro ASCII text. Non deve contenere nessun codice speciale che potrebbe essere inserito da word processors per i loro specifici propositi. Ti sono date, normalmente, l'opzione di salvare il file come puro DOS o ASCII (.TXT) dalla maggior parte di word processors.

Il programma seguente fornisce un buon primo test di un μ PIC nel mondo reale. Puoi digitarlo o puoi semplicemente copiarlo dalla subdirectory SAMPLES inclusa nel Compilatore PicBasic Pro. Il file è chiamato BLINK.BAS. Il source file BASIC, dovrà essere creato o spostato dalla stessa directory dove è allocato il file PBP.EXE.

'programma esempio per lampeggiare un LED connesso a PORTB.0 'circa 1 volta il secondo

```
loop: High PORTB.0          ' Accende il LED
      Pause 500             ' Ferma per .5 secondi

      Low PORTB.0          ' Spegne il LED
      Pause 500             ' Ferma per .5 secondi

      Goto loop            'Ricomincia il ciclo

End
```

Una volta soddisfatto perché il programma che hai scritto andrà perfettamente, puoi eseguire il Compilatore PicBasic Pro, inserendo PBP seguito dal nome del tuo text file ad un DOS prompt. Per esempio, se il text file che hai creato si chiama BLINK.BAS, al comando DOS prompt, inserisci:

PBP blink

Il compilatore mostrerà sul display un messaggio iniziale (copyright) e farà procedere il tuo file. Se gli piace, creerà un assembler source

code file (in questo caso chiamato BLINK.ASM) e automaticamente invocherà il suo assembler per completare il compito assegnato. Nel caso in cui tutto va bene, sarà creato il code file μ PIC finale (in questo caso, BLINK.HEX). Se hai preparato il compilatore in modo errato, risulterà una stringa di errori che bisognerà correggere nel tuo source file BASIC prima che tenti di nuovo la compilazione.

Per aiutarti a garantire che il tuo file originale sia perfetto, meglio cominciare scrivendo e testando una piccola parte del tuo programma, piuttosto che scrivere in una volta sola un'intera pagina compatta da 100 linee e quindi provando a correggerla poco a poco.

Se non la pensi diversamente, il Compilatore PicBasic Pro viene meno nella creazione di codici per il PIC16F84. Per compilare codice μ PIC, oltre che 'F84, usa semplicemente il comando **-P** descritto più tardi nel manuale, per specificare un processore differente. Per esempio, se intendi far girare il programma di cui sopra, BLINK.BAS, su un PIC16C74, compilalo usando il comando

PBP -p16c74 blink

2.3. Programma quel μ PIC

Mettendo il tuo programma compilato nel microcontrollore e testandolo abbiamo fatto due passi avanti.

Il Compilatore PicBasic Pro genera 8-bit Merged Intel HEX (.HEX) files standard, che possono essere usati con qualsiasi programmatore μ PIC, includendo il nostro EPIC Plus Pocket. I μ PIC non possono essere programmati con cavi di trasmissione a Marchio BASIC.

Il seguente esempio mostra come può essere programmato un μ PIC usando il nostro Programmatore EPIC con il software DOS. Se Windows 95/98/NT è disponibile, si raccomanda l'uso di EPIC versione Windows.

Assicurati che nel Programmatore EPIC non sia installato alcun μ PIC, allaccia il Programmatore EPIC alla porta parallela del PC, usando un maschio DB25, ad un cavo printer femmina DB25.

Inserisci l'adattatore AC nel Programmatore EPIC (oppure inserisci 2 pile nuove da 9-volt al programmatore, posizionandolo poi su "Batt ON").

A questo punto il LED sul Programmatore EPIC può essere on oppure off. Non inserire un μ PIC nella presa di programmazione, quando il LED è acceso, o prima che il software sia già partito.

Inserisci:

EPIC

Al comando DOS prompt, per iniziare la programmazione. Il software EPIC dovrà girare da una pura sessione DOS, o da una sessione DOS a schermo pieno sotto Windows oppure OS/2. (Non è incoraggiante girare sotto Windows. Tutte le sue varietà alterano le fasi del sistema e giocano con la porta quando non guardi, il che può causare errori di programmazione.

Il software EPIC darà un'occhiata in giro per trovare dove è collegato il Programmatore EPIC e lo renderà pronto a realizzare un μ PIC. Se non trovi il Programmatore EPIC, controlla tutte le connessioni di cui sopra e verifica che non ci sia nessun μ PIC o un qualsiasi altro adattatore collegato al programmatore.

Digitando:

EPIC /?

al comando DOS prompt, sarà mostrata sul display una lista di opzioni disponibili per il software.

Una volta che lo schermo di programmazione è sul display, usa il mouse per aprire il file, oppure premi **Alt-O** sulla tastiera. Utilizza il mouse (o la tastiera) per selezionare BLINK.HEX, o qualsiasi altro file che vorresti programmare nel μ PIC dal dialog box.

Il file caricherà, e vedrai una lista di numeri nella finestra a sinistra. Questo è il tuo programma in μ PIC code. Alla destra dello schermo c'è un display informativo di configurazione che sarà programmato nel μ PIC. Prima di procedere, verifica che tutto sia corretto.

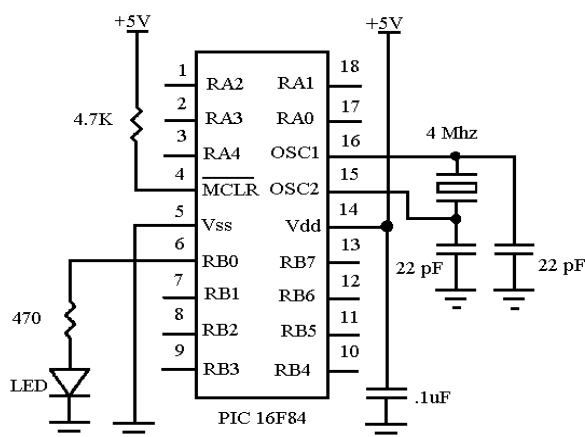
In generale, l'Oscillatore sarà impostato a XT per 4MHz crystal, e il Watchdog Timer sarà impostato su ON per i programmi PicBasic Pro. Quando si realizza la programmazione di un μ PIC windowed (JW), è molto

importante che **Code Protect** sia su **OFF**. Potresti non essere capace di cancellare un μ PIC windowed protetto da codice.

Quando tutto sembra meraviglioso, è tempo di inserire un μ PIC nella presa di programmazione e cliccare su Program, o premere **Alt-P** sulla tastiera. Prima di tutto, il μ PIC sarà controllato per garantire che sia vuoto, poi sarà inserito il tuo codice. Se il μ PIC non è vuoto, ed è un dispositivo flash, hai la possibilità di scegliere di proseguire oltre, senza prima dover cancellare. Una volta che la programmazione è completata e il LED è spento, puoi testare il tuo programma.

2.4. E' vivo

L'esempio riportato sotto ti dà un'idea delle poche cose necessarie alla connessione del μ PIC per lavorare. Fondamentalmente, tutto quello di cui hai bisogno è una resistenza pull-up sulla linea /MCLR, un cristallo 4 MHz con 2 condensatori, e di una fonte di energia a 5-volt. Abbiamo aggiunto un LED e resistenza per fornire l'output dal programma BLINK.



Costruisci e controlla questo semplice circuito su un protoboard e inserisci il μ PIC che hai appena programmato. La nostra linea di schede prototipi **PICProto** sono perfette per questo tipo di cose.

Collega la corrente. Il tuo μ PIC comincerà a vivere e il LED comincerà a lampeggiare una volta il secondo circa. Se non lampeggia, controlla tutti i collegamenti e assicurati che siano presenti 5 volts negli appropriati pin sul μ PIC.

Da questi semplici inizi, puoi creare la tua applicazione per la conquista mondiale.

2.5. Ho dei dubbi

I problemi più comuni riguardanti il procedimento del μ PIC, implica la certezza che i pochi componenti esterni siano dell'appropriato valore e perfettamente collegati al μ PIC. Di seguito, riportiamo alcuni suggerimenti per risolvere le cose e procedere.

Assicurati che il pin /MCLR sia collegato a 5 volts, reimpostando il circuito a voltaggio protetto, oppure semplicemente con una resistenza 4.7K. Se lasci il pin scollegato, il suo livello ondeggia a caso e a volte il μ PIC lavorerà, ma normalmente non avviene. Il μ PIC ha un circuito on-chip power-on-reset, perciò generalmente solo una resistenza pull-up esterna è collegata. Tuttavia, in alcuni casi, il μ PIC potrebbe non produrre energia in modo corretto e sarebbe necessario un circuito esterno. Per altre informazioni vedi i data books del Microchip PICmicro.

Assicurati di avere un buon cristallo, con condensatori correttamente collegati. I valori del condensatore possono essere difficili da leggere. Se i valori sono troppo diversi, l'oscillatore non funzionerà in modo corretto.

Un cristallo a 4MHz con due condensatori a disco in ceramica da 22pf (picofarad) è un buon inizio per la maggior parte dei μ PIC. Per altre riflessioni in proposito, dai ancora un'occhiata ai data books Microchip.

Parti piano. Scrivi programmi corti per testare le caratteristiche di cui non sei sicuro o che potrebbero darti problemi. Una volta che questi piccoli programmi lavorano correttamente, puoi costruirci su.

Prova a fare le cose in modo differente. A volte ciò che stai provando a fare sembra che lavorerà, ma poi ciò non accade, non importa quanto tu ne sia colpito duramente. Generalmente, c'è più di un modo per smembrare un programma. Prova ad affrontare il problema da un punto di vista diverso e forse riuscirai a trovare una spiegazione.

2.5.1. Punti specifici μ PIC

Assicurati di leggere i data sheets μ PIC. Alcuni dispositivi hanno caratteristiche che possono interferire con presunte operazioni su pin. Le parti del PIC16C62x (il 16C620, 621, e 622), ne sono un buon esempio. Questi μ PIC hanno comparatori analogici su PORTA. Quando questi chips saltano fuori, PORTA è impostato su un mode analogico. Ciò comporta che le funzioni dei pin su PORTA lavorino in maniera imprevista. Per convertire i pin in digitali, aggiungi semplicemente la linea:

CMCON = 7

vicino all'inizio del programma. Qualunque μ PIC con input analogici, come i dispositivi delle serie PIC16C7xx, PIC16F87x e PIC12C67x, salirà in mode analogico. Devi impostarli in mode digitale, secondo come vuoi usarli:

ADCON1 = 7

Un altro esempio di disastro potenziale è che PORTA, 4 pin, dimostra un comportamento inusuale quando è usato come output. Ciò perché il pin ha un canale output aperto, piuttosto che il solito stadio bipolare del resto dei pin output. Questo significa che può essere a terra quando è impostato a 0, ma ondeggerà quando impostato a 1, anziché andare verso l'alto. Per porre i pin nella maniera attesa, aggiungi una resistenza pull-up tra il pin e 5 volts. Il valore della resistenza dovrebbe essere tra 1K e 33K, secondo il drive necessario per l'input collegato. Quando è usato come un input, questo pin agisce come qualunque altro. Alcuni μ PIC, come i PIC16F873, 874, 876 e 877, permettono programmazioni a basso voltaggio. Tale funzione subentra in uno dei pin PORTB e può far funzionare in modo errato il dispositivo, se questo pin non è posto basso. E' meglio assicurarsi che la programmazione a basso voltaggio non sia abilitata nel momento in cui il μ PIC è programmato.

Compilatore PicBasic Pro

Tutti i pin del μ PIC sono impostati come inputs su power-up. Se hai bisogno di un pin output, impostalo ad output prima di usarlo, oppure usa un comando PicBasic Pro che lo faccia per te. Una volta ancora, rivedi i data sheets μ PIC per familiarizzare con un particolare dispositivo .

Il nome dei pin port sui dispositivi PIC12C67x e 12CE67x è GPIO. Il nome per il registro TRIS è TRISIO.

```
GPIO.0 = 1
```

```
TRISIO = %101010
```

Il dispositivo PIC 17Cxxx TRIS registers è chiamato, effettivamente, DDR (data direction register). Invece di usare TRIS per impostare la direzione input/output, usa DDR:

```
DDRB = 0 Imposta PORTB a tutti gli output
```

Non c'è data direction register per PORTA.

Certi μ PIC hanno una memoria on-chip non-volatile, implementata come una EEPROM seriale interfacciata I2C. **READ** e **WRITE** non funzionano su dispositivi con EEPROM seriale interfacciato I2C come le parti di 12CE67x e 16CE62x. Usa le istruzioni **I2CREAD** e **I2CWRITE**.

Alcuni dispositivi μ PIC, particolarmente i PIC12C671, 672, 12CE673 e 674, hanno oscillatori RC on-chip. Questi congegni contengono un fattore di calibratura di oscillatore nell'ultimo location del code space. L'oscillatore on-chip può essere precisato acquisendo il dato da questa location, e spostandolo nel registro OSCCAL. Sono state create due **DEFINES** per eseguire questo compito automaticamente, ogni volta che il programma comincia:

```
DEFINE OSCCAL_1K 1          `Imposta OSCCAL per  
dispositivo 1K
```

```
DEFINE OSCCAL_2K 1          `Imposta OSCCAL per  
dispositivo 2K
```

Aggiungi uno di questi 2 **DEFINES** vicino all'inizio del programma PicBasic Pro per eseguire l'impostazione di OSCCAL.

Se un dispositivo cancellabile UV è stato cancellato, il valore non può essere letto dalla memoria. Se uno di questi **DEFINES** è utilizzato in un dispositivo cancellabile, il programma comincerà a girare continuamente. Per impostare il registro OSCCAL su una dispositivo cancellabile, accanto all'inizio del programma, aggiungi la linea:

```
OSCCAL = $a0      'Imposta registro OSCCAL in $a0
```

Il \$a0 è semplicemente un esempio. Per ottenere l'effettivo valore OSCCAL per quel particolare dispositivo, la parte avrebbe bisogno di essere letta prima di cancellare. Vedi i Microchip data sheets per maggiori informazioni su OSCCAL e qualunque altro registro μ PIC.

2.5.2. Il codice incontra messaggi a fine pagina

Molti μ PIC contengono code space di memoria frammentata in 2K o 8K pagine. Poiché molti file sono compilati e poi riuniti, possono iniziare ad utilizzare più della prima pagina. Siccome ogni pagina è utilizzata, PM, l'assemblatore, emetterà un messaggio per avvisare che il codice sta attraversando un particolare confine. Ciò è normale e non mette in allarme. PBP si preoccuperà della maggior parte dei problemi al posto tuo. L'unica cosa di cui devi essere consapevole sono le istruzioni BRANCH. Se un BRANCH prova ad accedere in una label sull'altro lato di un confine, non lavorerà correttamente. Dovrà essere usato, invece, BRANCHL. Esso può indirizzare le labels in qualunque code page.

2.5.3. Errori di out of memory

Compilare molti source code files PicBasic Pro, può mettere a dura prova la memoria del PC che lavora sul compilatore. Quando si manifesta un Out of Memory e i FILES e BUFFERS sono impostati come raccomandato, può essere usata un'altra versione del PBP. Per far uso di tutta la memoria disponibile su Windows 95, 98, NT e 2000 è stato compilato PBPW.EXE. Naturalmente, ti devi dirigere in un DOS shell da uno di questi ambienti Windows 32-bit, oppure essere nel MPLAB di Microchip. Per eseguire la versione Windows dalla linea di comando DOS, sostituisci PBPW con PBP.

PBPW blink

2.6. Stile di programmazione

Scrivere programmi leggibili e mantenibili, è un'arte. Esistono poche e semplici tecniche da seguire, che possono aiutarti a diventare un artista.

2.6.1. Commenti

Usa tanti commenti. Anche se può essere perfettamente ovvio per te, ciò che il codice sta facendo, dal momento che lo scrivi tu, qualcun altro, guardando il programma (o comunque te stesso, se in vita sarai quel qualcuno), può non avere alcuna idea di ciò che stavi provando a compiere. Mentre i commenti occupano spazio nel tuo source file BASIC, non prendono alcuno spazio aggiuntivo nel μ PIC, perciò puoi usarne liberamente.

Creare commenti ti dice qualcosa di utile riguardo a cosa il programma sta realizzando. Un commento come "Set Pin0 a 1", spiega soltanto la sintassi del linguaggio, ma non fa nulla per dirti perché hai bisogno di fare una cosa del genere. Qualcosa come "Accendi il LED Battery Low" può essere molto più utile.

Un blocco di commenti all'inizio del programma, e prima di ogni sezione di codice, può descrivere cosa succede in modo più dettagliato, rispetto allo spazio rimanente dopo ogni affermazione. Ma non includere un blocco di commenti al posto di una linea individuale – usa entrambi.

All'inizio del programma, descrivi cosa intendi fare, chi lo ha scritto e quando. Può essere utile anche avere una lista per riesaminare informazioni e date. Specificare a cosa è collegato ogni pin, può essere d'aiuto per ricordare a quale hardware questo particolare programma è destinato. Se s'intende lavorare con un cristallo non-standard o con compilatori ad opzioni speciali, assicurati di elencarlo.

2.6.2. Pin e nomi variabili

Crea un pin o un nome variabile che sia un po' più coerente di **Pin0** o **B1**. In aggiunta al libero uso di commenti, i pin descrittivi e i nomi variabili possono migliorare la leggibilità. Il seguente frammento di codice lo dimostra:

Compilatore PicBasic Pro

```
BattLED var PORTB.0           'LED low battery
Level var byte'Variabile conterrà il livello
                        'batteria

If level < 10 Then           'Se la batteria è low
High BattLED               'Accendi il LED
Endif
```

2.6.3. Labels

Le etichette (label) dovranno essere più significative di “label:” oppure “qui”. Persino una label come “loop” è più descrittiva (anche se sottilmente). Normalmente la linea o routine su cui stai lavorando, rende qualcosa di unico. Prova a dare finalmente un suggerimento alle sue funzioni con una label, e poi prosegui con un commento.

2.6.4. GOTO

Prova a non usare troppi GOTO. Dato che GOTO potrebbe diventare un inevitabile errore, fai in modo di minimizzare il suo uso per quanto possibile. Prova a scrivere il tuo codice in sezioni logiche e non girarci troppo intorno. GOSUBs ti possono essere d'aiuto per compiere tutto questo.

2.6.5. GOSUB

Per ragioni di “spazio codice”, è meglio, normalmente, mettere le subroutines all'inizio del programma, piuttosto che alla fine. Avrai bisogno di posizionare un GOTO all'inizio per passare a queste subroutines.

3. Opzioni della linea di comando

3.1. Uso

Il Compilatore PicBasic Pro può essere richiamato dalla line di comando DOS usando il seguente format:

PBP Options Filename

Per modificare la maniera in cui PBP compila lo specifico file, possono essere usate zero o più opzioni. `Options` inizia sia con un segno meno (-), che con uno slash (/). Il carattere seguente il meno o lo slash è una lettera, che seleziona `Option`. Se `Option` richiede ulteriori informazioni, possono seguire dei caratteri aggiuntivi. Ogni `Option` deve essere separata da uno spazio, mentre ciò non accade al suo interno.

Possono essere usate `Options` multiple allo stesso tempo. Per esempio, la command line:

PBP -p16c71 -ampasm blink

Porterà come conseguenza che il file `BLINK.BAS` sarà compilato usando `MPASM` come assemblatore, e preso per un processore `PIC16C71`.

La prima voce che non comincia con un meno, si suppone che sia il `Filename`. Se non è specificata nessuna estensione, è utilizzata `.BAS`. Se una traiettoria è precisata, quella `directory` è ricercata per il file nominato. Senza badare a dove il file sorgente viene trovato, i files generati da PBP sono posizionati nella `directory` corrente.

Se la compilazione è eseguita senza errori, PBP lancia l'assemblatore (`PM.EXE`) automaticamente. PBP prevede di trovare `PM.EXE` nella stessa `directory` di `PBP.EXE`. Se la compilazione presenta degli errori, oppure viene utilizzata l'opzione `-s`, l'assemblatore non viene lanciato.

Se PBP è richiamato senza alcun parametro, sul display apparirà una schermata riassuntiva d'aiuto.

3.2.0 Opzioni

<u>Option</u>	<u>Descrizione</u>
A	Usa un differente Assembler
C	Inserisce source lines come Commenti nell'assembler file
E	Errori di output in un file
H (?)	Schermata d'aiuto
I	Usa una directory Include differente
L	Usa un Library file differente
O	Opzione per assembler
P	Specifica il target Processor
S	Skip execution di assembler dove fatta
V	Verbose mode

3.2.1. Opzione -A

PBP ha la capacità di usare, come suoi assembler, sia PM, che è incluso con PBP, che MPASM di Microchip. Per specificare MPASM (che deve essere acquistato da Microchip), usa **-ampasm** sulla command line:

PBP -ampasm filename

Se non è specificato alcun assembler sulla command line, è usato PM. Vedi la sezione sulla programmazione di assembly language, per avere maggiori informazioni.3.2.2. Opzione -C

Con l'opzione **-C**, il PBP inserisce le source file lines PicBasic Pro come commenti nell'assembly language source file. Questo può essere utile come strumento di correzione o di conoscenza, come dimostra l'istruzione PicBasic Pro, seguita dalle istruzioni che essa genera per l'assembly language.

PBP -c filename

3.2.3. Opzione -E

Con l'opzione -E, PBP invia tutti gli errori ad un file, filename.er.

PBP -e filename

3.2.4. Opzione -H o -?

Con l'opzione -H o -?, PBP mostra sul display una schermata di aiuto. Questa schermata viene mostrata anche se sulla command line non è specificata nessun'opzione o filename.

3.2.5. Opzione -I

L'opzione -I permette di selezionare l'indirizzo include per files usati da PicBasic Pro.

3.2.6. Opzione -L

L'opzione -L ti permette di selezionare la Library usata da PicBasic Pro. Generalmente, questa opzione non è necessaria, in quanto il Library file mancante è selezionato in un configuration file per ciascun tipo. Per maggiori informazioni, vedi le sezioni avanzate che si trovano più avanti, in questo manuale.

PBP -lpbpps2 filename

Questo esempio chiede al PBP di compilare filename usando la PicStic2 library.

3.2.7. Opzione -O

Con l'opzione -O, le lettere che la seguono passano all'assembler sulla command line, come opzioni. Alcune opzioni pertinenti al PM sono elencate nella tavola seguente:

Opzione PM	Descrizione
OD	Genera Listing, Symbol Table, e Map File
OL	Genera soltanto Listing

PBP -ol filename

Questo esempio dice al PBP di generare un `filename.lst` file, dopo una compilazione ben riuscita.

3.2.8. Opzione -P

Se non è altrimenti, PBP compila programmi per il PIC16F84. Se il programma richiede un processore differente come target, il suo nome deve essere specificato sulla command line usando l'opzione **-P**.

Per esempio, se il processore target desiderato è un PIC16C74, la command line assomiglierà a qualcosa come:

PBP -p16c74 filename

3.2.9. Opzione -S

Normalmente, quando PBP compila un programma con successo, automaticamente lancia l'assemblatore. Questo è fatto per convertire l'output assembler di PBP in immagine eseguibile. L'opzione **-S** previene questo, lasciando l'output nel `.ASM` file generato.

Dal momento che **-S** previene l'assemblatore dall'essere invocato, le opzioni che sono semplicemente passate all'assemblatore usando l'opzione **-O**, sono effettivamente sostituite.

PBP -s filename

3.2.10. Opzione -V

L'opzione **-V** accende il verbose mode PBP, che presenta maggiori informazioni durante la compilazione del programma.

PBP -v filename

4. Fondamentali PicBasic Pro

4.1 Identificatori

Un identificatore è, molto semplicemente, un nome. Essi sono usati nel PBP per line labels e nomi variabili. Un identificatore rappresenta qualsiasi sequenza di lettere, cifre, sottolineature, anche se non deve cominciare con una cifra. Gli identificatori non sono dati sensibili, perciò label, LABEL, e Label sono tutte trattate come equivalenti. E, mentre le labels possono contenere un numero di caratteri qualsiasi in lunghezza, PBP riconosce solo le prime 32.

4.2 Line Labels

Per indicare che il programma preferisce far riferimento ai comandi GOTO o GOSUB, PBP usa le line labels. Diversamente da vecchi BASIC, PBP non concepisce line numbers e non richiede che ogni linea sia etichettata. Piuttosto, qualunque PBP line può iniziare con una line label, che è semplicemente un identificatore seguito dai due punti (:).

here: Serout 0,N2400, ["Hello, World!"],13,10]

Goto here

4.3 Variabili

Si trovano dove il dato temporaneo (temporary data) è immagazzinato in un programma PicBasic Pro. Sono create usando la parola chiave **VAR**. Le variabili possono essere bits, bytes o parole. Lo spazio per ogni variabile è allocato nella RAM del microcontrollori dal PBP. Il format per la creazione di una variabile è come segue:

```
Label VAR Size{.Modifiers}
```

Label è una qualsiasi identificatore, escludendo keywords, come descritto sopra. Size è **BIT**, **BYTE** o **WORD**. I Modifiers opzionali aggiungono un controllo su come vengono create le variabili standard

Alcuni esempi sono:

dog	var	byte
cat	var	bit
w0	var	word

Nel PicBasic Pro non ci sono utilizzatori predefiniti di variabili. Per questioni di compatibilità, sono stati forniti due files che creano le variabili

standard, usate con il Marchio BASIC: “bs1defs.bas” e “bs2defs.bas”. Per usare uno di questi files, aggiungi la linea:

```
Include "bs1defs.bas"
```

oppure

```
Include "bs2defs.bas"
```

vicino al top del programma PicBasic Pro. Questi files contengono molti rapporti **VAR**, che creano tutte le variabili e definizioni pin a Marchio BASIC.

Ad ogni modo, invece di usare questi files “in scatola”, ti raccomandiamo di creare variabili tue, usando nomi che per te abbiano significato.

Il numero di variabili disponibile, dipende dalla quantità di RAM su un particolare dispositivo. E dalla misura delle variabili e degli apparati. PBP si riserva approssimativamente “24 RAM locations per il suo uso. Può anche creare ulteriori variabili temporanee, per la separazione di equazioni complesse.

4.4 Pseudonimi (Alias)

VAR può essere usato per creare alias per una variabile. Questo è utilissimo per accedere alle parti interne di una variabile.

```
fido var dog      `fido è un alias di cane
b0   var w0.byte0 `b0 è il primo byte della
      `parola w0
b1   var w0.byte1 `b1 è il secondo byte
      `della parola w0
flea var dog.0    `flea è bit0 di cane
```

Modificatori	<u>Descrizione</u>
BIT0 oppure 0	Crea l'alias a bit0 di byte o parola
BIT1 oppure 1	Crea l'alias a bit1 di byte o parola
BIT2 oppure 2	Crea l'alias a bit2 di byte o parola
BIT3 oppure 3	Crea l'alias a bit3 di byte o parola
BIT4 oppure 4	Crea l'alias a bit4 di byte o parola
BIT5 oppure 5	Crea l'alias a bit5 di byte o parola
BIT6 oppure 6	Crea l'alias a bit6 di byte o parola
BIT7 oppure 7	Crea l'alias a bit7 di byte o parola
BIT8 oppure 8	Crea l'alias a bit8 di parola
BIT9 oppure 9	Crea l'alias a bit9 di parola
BIT10 oppure 10	Crea l'alias a bit10 di parola
BIT11 oppure 11	Crea l'alias a bit11 di parola
BIT12 oppure 12	Crea l'alias a bit12 di parola
BIT13 oppure 13	Crea l'alias a bit13 di parola
BIT14 oppure 14	Crea l'alias a bit14 di parola
BIT15 oppure 15	Crea l'alias a bit15 di parola
BYTE0 o LOWBYTE	Crea l'alias a low byte di parola
BYTE1 o HIGHBYTE	Crea l'alias a high byte di parola

4.5 Arrays

Possono essere creati schemi variabili in maniera simile a quanto succede per le variabili.

Label **VAR** Size [Number of elements]

Label è un identificatore, escludendo parole chiave, come descritto sopra. Size è **BIT**, **BYTE** o **WORD**. Number of elements (numero di elementi) rappresenta quante locazioni per schemi sono richieste. Alcuni esempi di creazione sono:

```
sharks var byte [10]
fish   var bit  [8]
```

Compilatore PicBasic Pro

La prima locazione è elemento 0. Nello schema `fish` definito sopra, gli elementi sono numerati `fish [0]` a `fish [7]`, raccogliendo in totale 8 elementi. Poiché gli schemi sono allocati in memoria, ci sono dei limiti, in termini di misura, per ogni tipo:

Size	Numero massimo di elementi
BIT	256
BYTE	96*
WORD	48*

*Dipende dal processore. Per maggiori informazioni, vedi la sezione riguardante l'allocazione in memoria.

Gli schemi possono essere interamente adattati in una RAM bank su molti μ PIC. Potrebbero non estendersi alle RAM banks dei dispositivi 14-bit o 17Cxxx. (Gli schemi possono estendersi a banks su dispositivi 18Cxxx. Schemi dimensionati in byte e word, sono limitati solo in lunghezza dalla quantità di memoria). Il compilatore garantisce che gli schemi si adatteranno alla memoria, prima che le compilazioni abbiano avuto esito positivo.

4.6 Costanti

Dette costanti possono essere create allo stesso modo delle variabili. Può essere più conveniente usare un nome costante piuttosto che un numero costante. Se il numero ha bisogno di essere cambiato, questo può essere fatto solo in un punto del programma: dove è definita la costante. Il dato variabile può essere immagazzinato in una costante.

```
Label  CON      Constant expression
```

Alcuni esempi di costanti sono:

```
mice  con      3
traps con      mice * 1000
```

4.7 Simboli

SYMBOL rappresenta un altro metodo ancora per creare alias (pseudonimi) di variabili e costanti. E' incluso BS1 per compatibilità. **SYMBOL** non può essere usato per creare una variabile. Per far questo, usa **VAR**.

SYMBOL lion = cat `cat è stato creato precedentemente

Usando VAR

SYMBOL mouse = 1 `lo stesso come mouse con 1

4.8 Costanti numeriche

PBP permette di definire costanti numeriche su tre basi: decimale, binaria, ed esadecimale. I valori binari sono definiti usando il prefisso '%', e i valori esadecimali, usando il prefisso '\$'. I valori decimali non richiedono prefisso.

100	`	Valore decimale 100
%100	`	Valore binario per decimale 4
\$100	`	Valore esadecimale per decimale 256

Per facilitare la programmazione, i singoli caratteri sono convertiti ai loro equivalenti ASCII. I caratteri costanti devono essere citati usando le doppie virgolette e devono contenere solo un carattere (oltretutto, sono costanti in stringhe).

"A"	`	Valore ASCII per decimale 65
"d"	`	Valore ASCII per decimale 100

4.9 Costanti in stringhe

PBP non ha facoltà nel trattare stringhe, ma esse possono essere usate come comandi. Una stringa contiene uno o più caratteri ed è delimitata da doppie virgolette. Non sono supportate sequenze di fuga per caratteri non-ASCII (sebbene molti comandi PBP siano provvisti di questo trattamento).

Lcdout "Hello" ` Stringa output (corta per "H", "e", "l", "l", "o")

Generalmente, le stringhe vengono ritenute come una lista di valori a caratteri singoli.

4.10 Porte ed altri registri

Nel PicBasic Pro, si può avere accesso a tutti i registri del μ PIC, porte incluse, proprio come qualsiasi altra variabile byte-sized. Ciò significa che possono essere letti, scritti, oppure usati direttamente come equazioni:

Compilatore PicBasic Pro

```
PORTA = %01010101    `Scrivi il valore per PORTA
anyvar = PORTB & $0f  `Isola 4 bits di PORTB più in
basso e pone il risultato in anyvar
```

4.11 Pins

Si può accedere ai pins in un differente numero di modi. Il miglior modo per specificare un pin per un'operazione, è, semplicemente, usare il suo PORT name e numero bit:

```
PORTB.1 = 1          ` Imposta PORTB, bit 1 a 1
```

Per ricordare più facilmente a che scopo è usato un pin, bisognerà assegnargli un nome, utilizzando il comando **VAR**. In tale maniera, il nome può essere usato per qualsiasi operazione:

```
led var PORTA.0      ` Rinomina PORTA.0 come led
High led             ` Imposta led (PORTA.0) high (alto)
```

Per compatibilità con il Marchio BASIC, I pins usati nei comandi del Compilatore PicBasicPro possono anche essere riferiti da numeri, 0 - 15. Questi pins sono materialmente segnati su differenti porte hardware μ PIC, secondo quanti pins ha il microcontrollore.

Numero di pins μ PIC	<u>0 - 7</u>	<u>8 - 15</u>
8-pin	GPIO*	GPIO*
18-pin	PORTB	PORTA*
28-pin (eccetto 14C000)	PORTB	PORTC
28-pin (14C000)	PORTC	PORTD
40-pin	PORTB	PORTC

*GPIO e PORTA non hanno pins I/O.

Se una porta non ha 8 pins, come PORTA, possono essere usati solo i numeri pin esistenti, per esempio 8 - 12. Usando i numeri pin 13 - 15 avremo un effetto non visibile.

Questo numero pin, 0 - 15, non ha niente a che vedere con il numero pin fisico di un μ PIC. Secondo il particolare μ PIC, il numero pin 0 potrebbe essere pin fisico 6, 21, o 33, ma in ogni caso è segnato su PORTB.0 (oppure GPIO.0 per i dispositivi 8-pin, oppure PORTC.0 per un PIC14C000).

Compilatore PicBasic Pro

I pins possono essere menzionati con un numero (0 – 15), un nome (ad esempio **Pin0**, se uno dei files bsdefs.bas sono inclusi o se li hai definiti da te), oppure un nome full bit (per esempio PORTA.1). Si può accedere a qualsiasi pin o bit del microcontrollori usando l'ultimo metodo.

I nomi pin (ad es. **Pin0**) non sono automaticamente inclusi nel tuo programma. In molti casi dovrai definire nomi pin come li vedi più adatti, usando il comando **VAR**:

```
led var PORTB.3
```

Comunque, due definition files hanno provveduto ad accrescere la compatibilità con il Marchio BASIC. Il file "bs1defs.bas" oppure "bs2defs.bas", può essere incluso nel programma PicBasic Pro per fornire pin e bit names che si accordino con i nomi a Marchio BASIC.

```
Include "bs1defs.bas"
```

oppure

```
Include "bs2defs.bas"
```

BS1DEFS.BAS definisce **Pins, B0-B13, W0-W6** e molti degli altri pin BS1 e nomi variabili BS1.

BS2DEFS.BAS definisce **Ins, Outs, B0-B25, W0-W12** e molti degli altri pin e nomi variabili BS2.

Quando si dà energia ad un µPIC, tutti i pins sono selezionati a input. Per usare un pin come output, il pin o la port devono essere trasformati in output o deve essere usato un comando che automaticamente traduce un pin in output.

Per trasformare un pin o port in output (oppure input), seleziona il suo registro TRIS. In tal modo, un TRIS bit a 0, fa del suo pin un output. Selezionando un TRIS bit a 1, fa del suo pin un input. Per esempio:

```
TRISA = %00000000 'oppure TRISA = 0
```

trasforma tutti i pins della PORTA in outputs.

```
TRISB = %11111111 'oppure TRISB = 255
```

trasforma tutti i pins della PORTB in inputs.

Compilatore PicBasic Pro

```
TRISC = %10101010
```

trasforma tutti i pins pari sulla PORTC in outputs, e i pins dispari in inputs. Le direzioni individuali possono essere selezionate nella stessa maniera.

```
TRISA.0 = 0
```

trasforma PORTA, pin0 in output. Tutte le altre direzioni pin su PORTA sono invariate.

I nomi variabili Marchio BASIC, Dirs, Dirh, Dir1 e Dir0-Dir15, non sono definiti, e non dovranno essere usati con il Compilatore PicBasic Pro. Dovrà essere usato, invece, TRIS, ma all'opposto stato di Dirs.

Questo **NON** funziona nel PicBasic Pro:

```
Dir0 = 1    `Non trasforma il pin PORTB.0 in output
```

Invece:

```
TRISB.0 = 1    `trasforma il pin PORTB.0 in output  
Oppure, usa semplicemente un comando che, automaticamente, selezioni la  
direzioe del pin.
```

4.12 Commenti

Un commento PBP comincia sia con la parola chiave **REM**, che con la virgoletta ('). I caratteri che seguono su questa linea sono ignorati.

A differenza di molti BASIC, **REM** è un'unica parola chiave e non un'abbreviazione di REMark. Così, nomi variabili possono iniziare con **REM** (mentre **REM** da solo non è valido).

4.13 Linee di affermazioni multiple

Al fine di permettere programmi più compatti e raggruppamenti logici di comandi relazionati fra loro, PBP mantiene l'uso dei due punti (:) per separare affermazioni poste sulla stessa linea. Così, i due esempi che seguono sono equivalenti:

```
W2 = W0  
W0 = W1
```

W1 = W2

è lo stesso che:

W2 = W0 : W0 = W1 : W1 = W2

Ciò non cambia, comunque, la dimensione del codice generato.

4.14. Carattere line-extension

Il numero massimo di caratteri che può apparire su una linea PBP è 256. Rapporti più lunghi possono essere estesi alla linea successiva, usando il carattere di estensione linea (`_`) alla fine di ogni linea da continuare.

```
Branch B0, [label0, label1, label2, _  
Label3, label4]
```

4.15. INCLUDE

Al programma PBP possono essere aggiunti altri BASIC source files, usando **INCLUDE**. Potresti standardizzare subroutines, definizioni o altri files che desideri tener separati. Il Marchio i definition files a modalità seriale, sono esempi di questo. Tali files possono essere inclusi in programmi dove si ritiene sia necessario, ma tenuti fuori dei programmi dove non se ne richiede la presenza.

Le source code line del file incluso sono inserite nel programma, esattamente dove **INCLUDE** è situato.

```
INCLUDE "modedefs.bas"
```

4.16. DEFINE

Alcuni elementi, come il clock dell'oscillatore di frequenza e le locations di pin LCD, sono predefiniti nel PBP. **DEFINE** permette al programma PBP di cambiare queste definizioni, se desiderato.

DEFINE può essere usato per cambiare il valore predefinito dell'oscillatore i pins DEBUG e baud rate, e le locations pin LCD, tra le altre cose. Queste definizioni devono esistere nel caso superiore. Per specifiche informazioni su queste definizioni, vedi le sezioni appropriate.

Compilatore PicBasic Pro

```
DEFINE BUTTON_PAUSE      10      'Pulsante per debounce
ritardo in ms
DEFINE CHAR_PACING      1000     'Carattere serout misurazione
in us
DEFINE DEBUG_REG PORTB          'Porta debug pin
DEFINE DEBUG_BIT        0          'Bit debug pin
DEFINE DEBUG_BAUD      2400       'Debug baud rate
DEFINE DEBUG_MODE      1          'Debug mode: 0 = True, 1 =
Inverted
DEFINE DEBUG_PACING     1000     'Carattere debug misurazione
in us
DEFINE DEBUGIN_BIT     0          'Debugin pin bit
DEFINE DEBUGIN_MODE    1          'Deugin mode 0 = True, 1 =
Inverted
DEFINE HSER_BAUD        2400       'Hser baud rate
DEFINE HSER_SPBRG      25          'Hser spbrg init
DEFINE HSER_RCSTA      90h        'Hser riceve status init
DEFINE HSER_TXSTA      20h        'Hser trasmette status
init
DEFINE HSER_EVEN       1          'Usa solo se desideri
parità
DEFINE HSER_ODD        1          'Usa solo se desideri
disparità
DEFINE I2C_HOLD        1          'Pausa trasmissione I2C mentre clock
si mantiene low
DEFINE I2C_INTERNAL    1          'Usa per EEPROM interni su
16CExxx e 12CExxx
DEFINE I2C_SCLOUT      1          'Imposta il clock
bipolare invece di
open-collector
DEFINE I2C_SLOW        1          'Usa per OSC >8MHz
con dispositivi a
                                velocità standard
DEFINE LCD_DREG        PORTA      'Porta dati LCD
DEFINE LCD_DBIT        0          'LCD data starting
bit 0 oppure 4
DEFINE LCD_RSREG       PORTA      'Port seleziona il
registro LCD
DEFINE LCD_RSBIT       4          'Bit seleziona il
registro LCD
```

Compilatore PicBasic Pro

```
DEFINE LCD_EREG PORTB      `Port  abilita LCD
DEFINE LCD_EBIT  3         `Pin  abilita LCD
DEFINE LCD_RWREG PORTE     `Port  legge/scrive LCD
DEFINE LCD_RWBIT 2         `Bit   legge/scrive
LCD
DEFINE LCD_BITS  4         `Dimensione LCD bus
4 oppure 8
DEFINE LCD_LINES 2        `Numero linee su LCD
DEFINE LCD_COMMANDUS 2000 `Comando ritardo tempo in
us
DEFINE LCD_DATAUS   50     `Dato  ritardo tempo in us
DEFINE OSC 4             `Velocità
oscillatore in MHz: 3(3.58)
4 8 10 12 16 20 25
32 33 40
DEFINE OSCCAL_1K 1        `Imposta OSCCAL per
PIC12C671/CE673
DEFINE OSCCAL_2K 1        `Imposta OSCCAL per
PIC12C672/CE674
DEFINE SER2_BITS 8       `Imposta numero di
data bits per Serin2
e Serout2
DEFINE SHIFT_PAUSEUS 50  `Rallenta il clock Shiftin
e Shiftout
DEFINE USE_LFSR 1        `Usa  l'istruzione 18Cxxx
LFSR
```

4.17. Operatori matematici

Al contrario del Marchio BASIC, il Compilatore PicBasic Pro, compie tutte le operazioni matematiche in completo ordine gerarchico. Ciò significa che esiste una precedenza tra gli operatori. Moltiplicazioni e divisioni sono compiute prima delle addizioni e sottrazioni, ad esempio. Per garantire che le operazioni siano eseguite nell'ordine che tu desideri, usa le parentesi, per raggrupparle:

$$A = (B + C) * (D - E)$$

Tutte le operazioni matematiche non sono segnate, e compiute con precisione 16-bit.

Gli operatori supportati sono:

Compilatore PicBasic Pro

<u>Operatori matematici</u>	<u>Descrizione</u>
+	Addizione
-	Sottrazione
*	Moltiplicazione
**	Top 16 Bits di Moltiplicazione
*/	Media 16 Bits di Moltiplicazione
/	Divisione
//	Resto (Moduli)
<<	Shift Sinistro
>>	Shift Destro
ABS	Valore Assoluto
COS	Coseno
DCD	Decifra 2n
DIG	Numero
MAX	Massimo*
MIN	Minimo*
NCD	Encode
REV	Reverse Bits
SIN	Seno
SQR	Radica Quadrata
&	Bitwise AND
	Bitwise OR
^	Bitwise esclusivo OR
~	Bitwise NOT
&/	Bitwise NOT AND
/	Bitwise NOT OR
^/	Bitwise NOT esclusivo OR

*L'implementazione differisce dal Marchio BASIC.

4.17.1. Moltiplicazione

PBP esegue moltiplicazioni 16 \times 16. L'operatore '*' riporta i più bassi 16 bits del risultato 32-bit. Questa è la tipica moltiplicazione trovata nella maggior parte dei linguaggi di programmazione. L'operatore '**', riporta i più alti 16 bits del risultato 32-bit. Questi due operatori possono essere usati congiuntamente per eseguire moltiplicazioni 16 \times 16 che producono risultati 32-bit.

W1 = W0*1000 'moltiplica il valore W0 x 1000 e mette il risultato in W1

W2 = W0**1000 'moltiplica W0 x 1000 e mette l'alto ordine 16 bits (che può essere 0) in W2

L'operatore "*" riporta i 16 bits medi del risultato 32-bit.

W3 = W1*/W0 'moltiplica W1 x W2 e mette i 16 bits medi in W3

4.17.2. Divisioni

PBP esegue divisioni 16 x 16. L'operatore "/" riporta il risultato 16-bit. L'operatore "//" riporta il resto. Talvolta questo è riferito ai coefficienti del numero.

W1 = W0/1000 'divide il valore W0 x 1000 dando il risultato in W1

W2 = W0//1000 'divide il valore W0 x 1000 e pone il resto in W2

4.17.3. Shift

Gli operatori "<<" ">>" cambiano un valore rispettivamente sinistro o destro, da 0 a 15 volte. I bits cambiati sono selezionati a 0.

B0 = B << 3 'cambia B0 a sinistra di 3 posizioni (lo stesso che moltiplicare per 8)

W1 = W0 >> 1 'cambia W0 a destra di 1 posizione e mette il risultato in W1 (lo stesso che dividere per 2)

4.17.4. ABS

ABS converte il valore assoluto di un numero. Nel caso in cui un byte è maggiore di 127 (high bit set), ABS riporterà il valore 256. Con un termine maggiore di 32767 (high bit set), ABS riporterà il valore 65536.

B1 = ABS B0

4.17.5. COS

COS converte il coseno 8-bit in un valore. Il risultato è nella forma di complemento di due (ad es. da -127 a 127). Per trovare il risultato, esso usa una tabella di controllo ad 1/4 d'onda.

Coseno comincia con un valore in radiante binario, da 0 a 255, in opposizione ai soliti 0-359 gradi.

B1 = COS B0

4.17.6. DCD

DCD converte il valore decifrato di un numero bit. Cambia un valore bit (0-15) in un numero binario solo se esso è impostato a 1. Tutti gli altri sono impostati a 0.

B = DCD 2

4.17.7. DIG

DIG converte il valore di un numero decimale. Scegli semplicemente il numero primo (0 -4, essendo lo 0 la cifra massima) di cui cerchi il valore, e voilà.

B0 = 123

B1 = B0 DIG 1

4.17.8. MAX e MIN

MAX e **MIN** convertono, rispettivamente, il massimo e il minimo di due numeri. Normalmente, è utilizzato per limitare numeri di un valore.

B1 = B0 MAX 100 'imposta B1 maggiore di B0 e 100

(B1 sarà compreso tra 100 e 255)

B1 = B0 MIN 100 ‘imposta B1 minore di B0 e 100

(B1 non può essere maggiore di 100)

4.17.9. NCD

NCD converte la priorità del numero bit encoded (1 – 16) di un valore. E’ utilizzato per trovare il più alto bit set in un valore. Nel caso in cui nessun bit è impostato, NCD riporta 0.

B0= NCD %01001000 ‘imposta B0 a 7

4.17.10. REV

REV inverte l’ordine dei bits più bassi in un valore. Il numero di bits da invertire varia da 1 a 16.

B0= %10101100 REV`4 ‘imposta B0 a %1010001

4.17.11. SIN

SIN converte il seno 8-bit di un valore. Il risultato è in forma di complemento a due (per es. –127 a 127). Per trovare il risultato esso usa una tavola di controllo a ¼ d’onda. Seno comincia con un valore in radiante binario, da 0 a 255, al contrario dei normali 0-359 gradi.

B1 = SIN B0

4.17.12. SQR

SQR converte la radice quadrata di un valore. Visto che PicBasic Pro lavora solo con numeri interi, il risultato sarà sempre un valore 8-bit non più grande dell’effettivo risultato.

B0 = SQR W1 ‘seleziona B0 a radice quadrata di W1

4.17.13. Operatori bitwise

Gli operatori bitwise agiscono su ciascun bit di un valore in un modo Booleano Attraverso il loro utilizzo si possono isolare o aggiungere bits in un valore.

B0 = B0 & %00000001 ‘isola bit 0 di B0

Compilatore PicBasic Pro

B0 = B0 | %00000001 'seleziona bit 0 di B0
B0 = B0 ^ %00000001 'inverte lo stato di bit 0 su
B0

4.18. Operatori di paragone

Gli operatori di paragone sono usati in rapporti IF...THEN per mettere in relazione un'espressione con un'altra. Gli operatori supportati sono:

OPERATORI PARAGONE	DESCRIZIONE
= oppure ==	Uguale
<> o !=	Non uguale
<	Minore di
>	Maggiore di
<=	Minore di oppure uguale
>=	Maggiore di oppure uguale

IF i > 10 THEN LOOP

4.19. Operatori logici

Gli operatori logici differiscono dalle operazioni bitwise. Essi producono un risultato true/inverted, dalla loro operazione. Valori pari a 0 sono ritenuti inverted, qualsiasi altro è true. Sono usati per la maggior parte congiuntamente agli operatori di paragone, in un rapporto IF..THEN. Gli operatori supportati sono:

OPERATORE LOGICO	DESCRIZIONE
------------------	-------------

Compilatore PicBasic Pro

AND o &&	Logico AND
OR o	Logico OR
XOR o ^^	Logico Esclusivo
NOT AND	Logico NAND
NOT OR	Logico NOR
NOT XOR	Logico NXOR

```
IF (A = = grande) AND (B > medio) THEN RUN
```

Ricorda di usare le parentesi per comunicare al PBP l'esatto ordine che hai deciso per l'esecuzione delle operazioni.

5. Cenni sul rapporto PicBasic Pro

Compilatore PicBasic Pro

@	Inserisce una linea d'assembly language code
ADCIN	Convertitore on-chip da analogico a digitale
ASM.ENDASM	Inserisce la sezione assembly language code
BRANCH	GOTO valutato (equivalente a ON..GOTO)
BRANCHL	BRANCH fuori pagina (long BRANCH)
BUTTON	Ferma e ripete automaticamente input su pins specifici
CALL	Chiama subroutines d'assembly language
CLEAR	Azzera tutte le variabili
CLEARWDT	Cancella il Watchdog Timer
COUNT	Definisce argomenti iniziali EEPROM on-chip
DEBUG	Output seriale asincrono pin e baud fissi
DEBUGIN	Input seriale asincrono pin e baud fissi
DISABLE	Disattiva i procedimenti ON DEBUG e ON INTERRUPT
DISABLE DEBUG	Disattiva il procedimento ON DEBUG
DISABLE INTERRUPT	Disattiva il procedimento ON INTERRUPT
DTMFOUT	Produce toni su un pin
EEPROM	Definisce argomenti iniziali su EEPROM on-chip
ENABLE	Abilita i procedimenti ON DEBUG e ON INTERRUPT
ENABLE DEBUG	Abilita il procedimento ON DEBUG
ENABLE INTERRUPT	Abilita il procedimento ON INTERRUPT
END	Ferma l'esecuzione e inserisce il low power mode
FOR..NEXT	Esegue relazioni in modo ripetitivo
FREQOUT	Produce fino a 2 frequenze su un pin
GOSUB	Chiama subroutines BASIC e specifiche labels
GOTO	Continua l'esecuzione sulla specifica label
HIGH	Rende alto un output pin
HSERIN	Input hardware seriale asincrono
HSEROUT	Output hardware seriale asincrono
I2CREAD	Legge bytes da dispositivi I2C
I2CWRITE	Scrive bytes a dispositivi I2C
IF..THEN..ELSE..ENDIF	Esegue rapporti in modo condizionato
INPUT	Trasforma il pin in input
LCDIN	Legge da LCDRAM
LCDOUT	Mostra caratteri su LCD
{LET}	Assegna il risultato di un'espressione ad una variabile
LOOKDOWN	Ricerca tavola delle costanti per valore
LOOKDOWN2	Ricerca tavola delle costanti/variabili per valore
LOOKUP	Estrae valori costanti dalla tavola
LOOKUP2	Estrae valori costanti/variabili dalla tavola

Compilatore PicBasic Pro

LOW	Rende basso un output pin
NAP	Rallenta il processore per un piccolo periodo
ON DEBUG	Esegue dispositivo di controllo BASIC
ON INTERRUPT	Esegue subroutine BASIC su un'interruzione
OUTPUT	Trasforma un pin in output
PAUSE	Pausa (1msec resolution)
PAUSEUS	Pausa (1µsec resolution)
PEEK	Legge byte dal registro (non usare)
POKE	Scrive byte sul registro (non usare)
POT	Legge potenziometro su pin specifico
PULSIN	Misura l'ampiezza d'impulso su un pin
PULSOUT	Genera impulso su un pin
PWM	Ampiezza impulso output e sviluppo modulato al pin
RANDOM	Genera numeri pseudo-random (casuali)
RCTIME	Misura l'ampiezza d'impulso su un pin
READ	Legge byte da EEPROM on-chip
READCODE	Legge caratteri da code memory
RESUME	Continua l'esecuzione dopo aver trattato un'interrupt
RETURN	Continua la relazione che segue l'ultima GOSUB
REVERSE	Trasforma un output pin in input e viceversa
SERIN	Input seriale asincrono (BS1 style)
SERIN2	Input seriale asincrono (BS2 style)
SEROUT	Output seriale asincrono (BS1 style)
SEROUT2	Output seriale asincrono (BS2 style)
SHIFTIN	Input seriale sincrono
SHIFTOUT	Output seriale sincrono
SLEEP	Rallenta il processore per un periodo
SOUND	Genera toni o rumori bianchi su uno specifico pin
STOP	Ferma l'esecuzione del programma
SWAP	Scambia i valori di due variabili
TOGGLE	Trasforma output pins e toggle state
WHILE..WEND	Esegue rapporti se la condizione è true
WRITE	Scrive byte su EEPROM on-chip
WRITECODE	Scrive caratteri verso la code memory
XIN	X-10 input
XOUT	X-10 output

5.1. @

Usato all'inizio di una linea, @ fornisce una scorciatoia per l'inserimento di un assembly language nel tuo programma PBP. Puoi usare questa via per adattare liberamente l'assembly language code ai rapporti del Pic Basic Pro.

```
    i   var byte
      rollme var byte
      For i = 1 to 4
@ rlf _ rollme, F ; 'ruota una volta il byte sinistro
      Next i ;
```

La scorciatoia @ può trovare impiego anche per includere assembly language routines in un altro file. Per esempio:

```
@ Include "fp.asm"
```

La funzione @ imposta il register page a 0, prima di eseguire l'assembly language instruction. Il register page non sarà alterato usando @.

Vedi la sezione riguardante la programmazione assembly language, per maggiori informazioni.

5.2. ADCIN

ADCIN Channel, var

Converte il Channel on-chip da analogico a digitale e immagazzina il risultato in **Var**. Mentre si può avere un accesso diretto ai registri ADC, **ADCIN** rende il procedimento un po' più semplice.

Prima che **ADCIN** sia usato, deve essere impostato l'appropriato registro TRIS, per inserire i pins desiderati (input). Anche ADCON1 deve essere impostato, per assegnare i pins desiderati ad inputs analogici e, in alcuni casi, selezionare i result format e clock source. Vedi Microchip data sheets, per maggiori informazioni su questi registri e su come impostarli per lo specifico dispositivo.

Si possono utilizzare anche diverse **DEFINES**:

```
DEFINE ADC__BITS 8      'imposta il numero di bits in risult.
DEFINE ADC__CLOCK 3     'imposta clock source (rc=3)
DEFINE ADC__SAMPLEUS 50 'imposta il tempo campione in
                        μsec
TRISA = 255             'seleziona PORTA per tutti gli input
ADCON1 = 2              'PORTA è analogica
ADCIN 0, B0             'legge channel 0 a B0
```

5.3. ASM..ENDASM

ASM

ENDASM

Secondo il PBP, il codice tra le istruzioni **ASM** ed **ENDASM** è in assembly language, e tali linee non saranno interpretate come statements PicBasic Pro. Puoi usare queste due istruzioni per adattare liberamente il codice assembly language alle affermazioni PicBasic Pro.

La misura massima per una sezione di assembler text è 8K. Questa è la misura massima per l'effettiva source, che include commenti, non il codice generato. Se il text block è ampio, frammentalo in sezioni multiple di **ASM..ENDASM** o includilo semplicemente in un file separato.

ASM reimposta il register page a 0. Se l'assembly language ha alterato il register page, assicurati che esso sia reimpostato a 0, prima di passare a **ENDASM**.

Per maggiori informazioni, vedi la sezione dedicata alla programmazione assembly language.

ASM

`bsf PORTA , 0` 'seleziona bit 0 su PORTA

`bcf PORTB , 0` 'cancella bit 0 su PORTB

ENDASM

5.4. BRANCH

BRANCH Index, [Label {, Label...}]

BRANCH permette al programma di saltare verso una differente locazione basata su un esponente variabile. E' simile ad On..Goto in altri BASICs.

Index seleziona una label all'interno di una lista . L'esecuzione ricomincia dalla label indicata. Per esempio, se Index è 0, il programma passa alla prima label specificata nella lista. Se Index è uno, il programma passa alla seconda label, e così via. Se Index è maggiore od uguale al numero di labels, nessun'azione è compiuta e l'esecuzione prosegue con lo statement successivo a **BRANCH**. In **BRANCH** può essere usato un numero di labels superiore a 255 (256 per 18Cxxx)

Label deve stare nella stessa code page dell'istruzione **BRANCH**. Se non ne sei sicuro, usa **BRANCHL** qui di seguito.

```
BRANCH B4, [dog, cat, fish]
```

```
`Same as
```

```
`If B4 = 0 then dog (Goto dog)
```

```
`If B4 = 1 then cat (Goto cat)
```

```
`If B4 = 2 then fish (Goto fish)
```

5.5. **BRANCHL**

BRANCHL Index, [Label {,Label...}]

BRANCHL (BRANCH Long) lavora in modo molto simile a **BRANCH**, in quanto permette al programma di andare verso una diversa locazione basata su un esponente (indice) variabile. Le principali differenze si basano sul fatto che il programma può andare verso una label che si trova in una code page diversa dall'istruzione **BRANCHL**, e che genera code circa due volte la misura del codice generato con l'istruzione **BRANCH**. Se sei sicuro che le labels si trovano nella stessa pagina del **BRANCH**, o se il microcontrollore non ha più di una code page (2K o meno di ROM), usa **BRANCH** invece di **BRANCHL**, e minimizzerai l'utilizzo di memoria.

Index seleziona una label in una lista. L'esecuzione ricomincia da quella indicata. Per esempio, se **Index** è zero, il programma passa alla prima label specificata nella lista, se **Index** è uno, il programma passa alla seconda label, e così via.

Se **Index** è maggiore o uguale al numero di labels, non è compiuta nessun'azione, e l'esecuzione prosegue con lo statement successivo a **BRANCHL**. In un **BRANCHL**, possono essere usati fino a 127 labels (256 per i dispositivi 18Cxxx).

```
BRANCHL B4 , [dog cat fish]
`If B4 = 0 then dog (Goto dog)
`If B4 = 1 then cat (Goto cat)
`If B4 = 2 then fish (Goto fish)
```

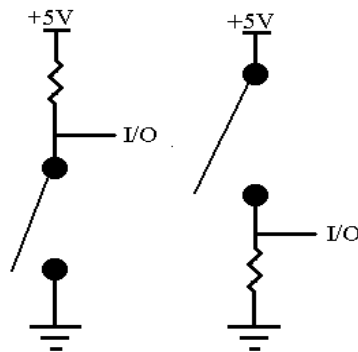
5.6. BUTTON

BUTTON Pin, Down, Delay, Rate, BVar, Action, Label

Compilatore PicBasic Pro

Legge Pin e facoltativamente esegue debounce e auto-repeat. Pin è automaticamente un input (entrata). Pin può essere una costante, 0-15, o una variabile che contiene un numero 0-15 (ad es. B0) o un nome pin (ad es. PORTA.O).

Down	Stato del pin quando il bottone è premuto (0..1)
Delay	Conteggio ciclico, prima che cominci l'auto-repeat (0..255). Se 0, non è eseguito né debounce, né auto-repeat. Se 255, sarà eseguito debounce, ma non auto-repeat (0..255)
Rate	Stima auto-repeat (0..255)
Bvar	Variabile byte-sized usata internamente per fermare/ripetere il conto alla rovescia. Deve essere inizializzato a 0 prima di usarlo e non utilizzarlo altrove nel programma.
Action	Stato del bottone per funzionare 0 se non premuto, 1 se premuto)
Label	Se Action è true, l'esecuzione ripartirà da questa label.



‘Goto notpressed se button a 0 su Pin2

```
BUTTON_PORTB, 2, 0, 100, 10, B2, 0, notpressed
```

Per lavorare correttamente, **BUTTON** ha bisogno di essere usato dentro un loop per auto-repeat. Button compie debounce attraverso l'esecuzione d'arresto programma, per un periodo di millisecondi, affinché i contatti si stabilizzino.

L'arresto di debounce è di 10ms. Per cambiare il debounce in un altro valore, usa

DEFINE:

```
‘ Setta il ritardo a 50 mS
```

```
DEFINE BUTTON_PAUSE 50 'imposta il bottone  
arresto debounce a 50ms
```

Assicurati che **BUTTON_PAUSE** sia, in tutti i casi, maiuscolo.

In generale, è più facile leggere lo stato del pin in un **IF..THEN** piuttosto che attraverso l'uso di **BUTTON**, come:

```
IF PORTB2 = 1 THEN notpressed
```

5.7. CALL

CALL Label

Esegue la subroutine assembly language della Label.

Normalmente si usa **GOSUB** per eseguire una subroutine PicBasic Pro. La principale differenza tra **GOSUB** e **CALL** è che con **CALL** l'esistenza di Label non è controllata fino all'assembly time. Usando **CALL** si può entrare in una Label in una sezione assembly language, che in PBP è altrimenti inaccessibile.

Per maggiori informazioni, vedi la sezione riguardante la programmazione d'assembly language su **CALL**.

CALL pass 'esegue subroutine assembly language
chiamata pass.

5.8. CLEAR

CLEAR ‘Imposta tutti i registri RAM a 0

CLEAR azzerà tutti i registri RAM in ogni banco. Questo imposterà tutte le variabili, includendo quelle interne al sistema, a 0.

Ciò non avviene automaticamente, quando un programma PBP comincia come uno a Marchio BASIC. In generale, le variabili dovrebbero essere impostate nel programma in uno stato iniziale appropriato, piuttosto che usare **CLEAR**.

CLEAR ‘porta tutte le var a 0

5.9. CLEARWDT

Compilatore PicBasic Pro

CLEARWDT 'azzerà il Watchdog Timer

Il Watchdog Timer è usato congiuntamente alle istruzioni **SLEEP** e **NAP**, per attivare il μ Pic dopo un certo tempo. Le istruzioni dell'assembler (clrwdt), necessarie per impedire che il Watchdog Timer vada fuori tempo sotto normali circostanze, e reimpostare il μ Pic, sono automaticamente inserite nell'esatto spazio all'interno del programma.

CLEARWDT permette il posizionamento di altre istruzioni clrwdt nel programma.

CLEARWDT 'azzerà Watchdog Timer

5.10. COUNT

COUNT Pin, Period, Var
50

Conta il numero d'impulsi che ricorrono sul Pin durante il Period immagazzinando il risultato in Var. Pin è reso automaticamente in input. Pin può essere una costante, 0-15, o una variabile contenente un numero 0-15 (ad es. B0) oppure un nome pin (ad es. PORTA.0)

La risoluzione di Period è in millisecondi. Segue la frequenza dell'oscillatore basata sull'OSC definito.

COUNT controlla lo stato di Pin in uno stretto loop e conta le transizioni basse e alte. Con un oscillatore 4MHz, controlla lo stato del pin ogni 20µs. Con un oscillatore 20 MHz, controlla lo stato del pin ogni 4µs. Da ciò si calcola che gli impulsi più alti saranno di 25KHz con un oscillatore 4MHz, e di 125KHz con 20MHz, se la frequenza ha un ciclo d'uso pari al 50% (high time è lo stesso che low time).

```
COUNT PORTB.1, 100, W1
'conteggia # d'impulsi su Pin1 in 100 millisecondi

COUNT PORTA.2, 1000, W1      'determina la
frequenza su pin

Serout PORTB.2, N2400, [W1]    'trasmette
per 1 secondo
```

5.11. DATA

```
DATA { @ Location, } Constant { ,Constant...}
```

Immagazzina costanti in **EEPROM** on-chip non-volatile, quando il dispositivo è programmato. Se il valore opzionale Location è omissso, il primo statement **DATA** comincia ad immagazzinare ad indirizzo 0 e le successive affermazioni immagazzinano alle locations seguenti. Se tale valore è specificato, denota la Location d'inizio dove questi valori sono immagazzinati. Una label facoltativa può essere assegnata all'indirizzo **EEPROM** per riferimenti attraverso il programma.

Constant può essere una costante numerica o una stringa. E' utilizzato solo il più piccolo byte di valori numerici che sono immagazzinati senza il modificatore **WORD**. Le stringhe sono immagazzinate come bytes consecutivi di valori **ASCII**. Non sono aggiunti automaticamente né estensione, né limite.

DATA lavora solo con microcontrollori EEPROM on-chip, come il PIC16F84 e PIC16C84. Non lavorerà su dispositivi con EEPROM on-chip seriale interfacciato I2C, come le parti del 12CE67x e 16CE62x. Poiché EEPROM è memoria non-volatile, i dati rimarranno inalterati, anche in assenza di corrente elettrica.

Il dato è inserito nello spazio EEPROM solo una volta, nel momento in cui è programmato il microcontrollore, e non ogni volta che il programma è in funzione.

WRITE può essere usato per selezionare i valori dell'EEPROM on-chip, durante il funzionamento.

DATA @ 5,10, 20, 30 'immagazzina 10, 20, e 30 inizi a location 5

dlabel **DATA** word \$1234 'asigna una label ad un carattere nella location successiva 'immagazzina \$34,\$12

DATA (4), 0 (10) 'salta 4 locations e immagazzina 10 0

5.12. DEBUG

DEBUG Item { , Item...}

Compilatore PicBasic Pro

Invia uno o più Items ad un pin predefinito ad un baud rate in formato asincrono standard, usando 8 data bits, no parity e 1 stop bit (8N1). Il pin diventa automaticamente un output.

Se un cancelletto (#) precede un Item, la rappresentazione **ASCII** è inviata in modo seriale per ogni cifra. **DEBUG** supporta anche i modificatori dati come **SEROUT2**. Per questa informazione, fai riferimento alla sezione relativa a **SEROUT2**.

MODIFICATORE	OPERAZIONE
{I}{S}BIN{1..16}	Invia cifre binarie
{I}{S}DEC{1..5}	Invia cifre decimali
{I}{S}HEX{1..4}	Invia cifre esadecimali
REPCn	Invia carattere c ripetuto n volte
STR ArrayVar{\n}	Invia stringa di n caratteri

DEBUG è una delle diverse funzioni seriali asincrone create. E' la più piccola tra le routines seriali del software. Può essere usata per mandare informazioni debugging (variabili, contrassegni di posizione, ecc.) ad un programma terminale come Hyperterm. Può essere usata qualsiasi volta si desidera un output seriale su un pin predefinito ad un baud rate.

Il pin seriale e il baud rate sono specificati usando **DEFINEs**:

```
DEFINE DEBUG_REG PORTB    'seleziona Debug pin port
DEFINE DEBUG_BIT 0        'seleziona Debug pin bit
DEFINE DEBUG_BAUD 2400    'seleziona Debug baud rate
DEFINE DEBUG_MODE 1      'seleziona Debug mode: 0 = true,
                          1 = inverted
```

DEBUG assume un oscillatore 4MHz quando si generano dei bit. Al fine di mantenere il baud rate timing più appropriato con altri valori oscillatori, assicurati di definire l'impostazione **OSC** per ciascun termine. In alcuni

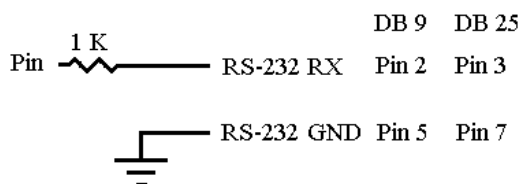
casi, gli ordini di trasmissione istruzioni **DEBUG**, possono presentare caratteri ad un dispositivo ricevente in modo troppo veloce.

Una **DEFINE** aggiunge pacing dei caratteri alla trasmissione output seriale. Ciò concede più tempo tra i caratteri, come essi sono trasmessi. Il pacing dei caratteri **DEFINE**, permette una pausa che va da 1 a 65,535 μ secondi (da .001 a 65.535 millisecondi) tra ogni carattere trasmesso.

Per esempio, per mettere una pausa di un millisecondo tra la trasmissione di ogni carattere:

```
DEFINE DEBUG_PACING 100
```

Mentre i convertitori single-chip a livello RS-232 sono comuni e non costosi, grazie all'impianto corrente RS-232, e le eccellenti spiegazioni dettagliate I/O del μ Pic, la maggior parte delle applicazioni non richiedono convertitori di livello. Invece, può essere usato TTL invertito (**DEBUG_MODE** = 1). Una resistenza per la limitazione di corrente è suggerita. (RS-232 si suppone essere short-tolerant).



‘invia il testo “B0 =” seguito dal valore decimale di B0 e un linefeed out in modo seriale

```
DEBUG “B0 =”, dec B0, 10
```

5.13. DEBUGIN

```
DEBUGIN { Timeout, Label, } [ Item {, Item...} ]
```

Compilatore PicBasic Pro

Riceve uno o più Items da un pin predefinito ad un baud prestabilito in formato asincrono standard, usando 8 data bits, no parity e 1 stop bit (8N1). Il pin diventa automaticamente un input. Può essere incluso un facoltativo Timeout e Label per permettere al programma di continuare se un carattere non è ricevuto in un certo lasso di tempo. Timeout è specificato in unità di 1 millisecondo.

DEBUGIN supporta gli stessi modificatori data di **SERIN2**.

Per questa informazione fai riferimento alla sezione relativa a **SERIN2**

MODIFICATORE	OPERAZIONE
BIN {1..16}	Riceve cifre binarie
DEC {1..5}	Riceve cifre decimali
HEX {1..4}	Riceve cifre hexadecimali
SKIP n	Salta n caratteri ricevuti
STR ArrayVar \n{C}	Riceve stringa di n caratteri opzionali terminati in car. c
WAIT ()	Aspetta sequenza di caratteri
WAITSTR ArrayVar {n}	Aspetta carattere a stringa

DEBUGIN è una funzione seriale asincrona. E' la più piccola tra le routines seriali generate del software. Può essere utilizzato per ricevere informazioni debugging da un programma terminale come Hyperterm. Può anche essere usato in qualunque momento si desidera l'input seriale su un dato pin ad uno specifico baud rate.

Usando **DEFINES** si specificano i pin seriali e baud rate:

```
DEFINE DEBUGIN_REG PORTB 'seleziona Debugin pin port
```

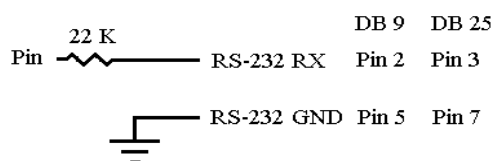
```
DEFINE DEBUGIN_BIT 0 'seleziona Debugin pin bit
```

```
DEFINE DEBUGIN_MODE 1 'seleziona Debugin mode: 0 = true, 1 = inverted
```

Se una qualsiasi **DEFINE** non è inclusa in un programma, la port **DEBUG**, pin o mode, sono impostati agli stessi valori come per **DEBUG**. Il **DEBUG** baud rate è sempre lo stesso del **DEBUG**. Non può essere definito diversamente.

DEBUG assume un oscillatore 4MHz quando sta generando il suo conteggio di bit. Per mantenere il proprio calcolo di baud rate con altri valori oscillatori, assicurati di definire l'impostazione **OSC** a qualsiasi valore oscillatore.

Poiché i convertitori di livello RS-232 single-chip sono comuni e non costosi, grazie all'impiantazione di corrente RS-232 e l'eccellente specificazione I/O del μ Pic, il maggior numero d'applicazioni non richiede convertitori di livello. Piuttosto, può essere usato TTL (**DEBUG_MODE** = 1) invertito. Si consiglia una resistenza di limitazione corrente (RS-232 si



suppone essere short-tolerant).

```
DEBUGIN [wait("A"), B0] 'attende finche il carattere "A" sia  
ricevuto in modalità seriale e mette il carattere successivo in B0
```

```
DEBUGIN [skip 2, dec 4 B0] 'salta 2 caratteri e blocca un  
numero a 4 cifre decimali .
```

5.14. DISABLE

DISABLE

DISABLE corregge e interrompe il procedimento seguendo quest'istruzione. Interruzioni possono ancora accadere, ma l'Interrupt handler e il monitor debug non saranno eseguiti nel programma PicBasic Pro, finché non sia affrontato un **ENABLE**.

DISABLE ed **ENABLE** sono molto simili a pseudo.ops per quel che riguarda ciò che essi danno alla guida del compilatore, piuttosto che generare effettivamente un codice. Per maggiori informazioni, vedi **ON DEBUG** e **ON INTERRUPT**.

DISABLE 'disabilita interrupt handler

myint : led = 1 'accende il **LED** quando interrotto

Resume 'ritorna al programma principale

Enable 'abilita interrupt handler

5.15. DISABLE DEBUG

DISABLE DEBUG

DISABLE DEBUG procede seguendo quest'istruzione. Il debug monitor non sarà chiamato tra le istruzioni finché un **ENABLE** oppure **ENABLE DEBUG** non sarà incontrato.

DISABLE DEBUG e **ENABLE DEBUG** sono più simili a pseudo-ops, in ciò che danno alle direzioni del compilatore, piuttosto che generare effettivamente un codice. Per maggiori informazioni, vedi **ON DEBUG**.

DISABLE DEBUG 'disabilita le chiamate di debug monitor

5.16. DISABLE INTERRUPT

DISABLE INTERRUPT

DISABLE INTERRUPT procede seguendo quest'istruzione. Possono ancora verificarsi interruzioni, ma il BASIC handler, nel programma PicBasic Pro, sarà eseguito quando sarà incontrato un **ENABLE** oppure un **ENABLE INTERRUPT**.

DISABLE INTERRUPT ed **ENABLE INTERRUPT** sono più simili a pseudo-ops in ciò che danno alle direzioni del compilatore, piuttosto che generare effettivamente un codice. Per maggiori informazioni vedi **ON INTERRUPT**.

DISABLE INTERRUPT 'disabilita interrupt handler

mynt: led = 1 'accende **LED** quando interrotto

Resume 'Ritorna al programma principale

Enable interrupt 'abilita interrupt handler

5.17. DTMFOUT

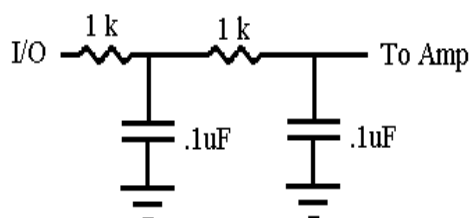
DTMFOUT Pin, {Onms, Offms,}[Tone {Tone,...}]

Il tasto **DTMF** produce una sequenza Tone su Pin.

Pin diventa automaticamente un output. Esso può essere una costante, 0-15, oppure una variabile contenente un numero 0 – 15 (ad es. B0) o un nome pin (ad es. PORTA.0). Onms è il numero di millisecondi necessario a produrre ogni tono, e Offms rappresenta il numero di millisecondi per stabilire una pausa tra ogni tono. Se non sono specificati, Onms conta 200ms e Offms 50ms

Tones sono numerati da 0 – 15. Tones da 0 – 9 sono gli stessi che si trovano su una tastiera telefonica. Tone 10 è la chiave *; Tone 11 la chiave #, Tones 12 – 15 corrispondono all'estensione delle chiavi A – D.

DTMFOUT usa **FREQOUT** per generare dual tones. FREQOUT genera toni usando una forma di modulazione dell'ampiezza d'impulso. Il dato grezzo che risulta dal pin sembra pauroso. Normalmente si rende necessario un certo tipo di filtro, al fine di attenuare il segnale ad un'onda seno, che si liberi da alcune armoniche generate



DTMFOUT funziona meglio con un oscillatore da 20MHz. Può lavorare anche con un apparecchio da 10Mhz e persino 4MHz, sebbene comincerà a filtrare difficilmente, e ad essere di un'ampiezza piuttosto bassa. Qualunque altra frequenza porterà DTMFOUT a generarne una che sarà una proporzione tra l'effettivo oscillatore utilizzato e 20MHz, non molto utile per l'invio dei toni.

```
DTMFOUT PORTB.1, [2, 1, 2] 'invia toni DTMF per 212 su Pin1
```