■　PBP3日本語マニュアル目次　■

1．PICBASIC PRO Compilerのインストール ―――― 2
1-1、PICBASIC PRO Compiler とは・・・ …… 2
1-2、PICBASIC PRO Compiler をインストールしましょう …… 2
1-3、PICBASIC PRO Compiler 本体の種類 …… 7
2．PICBASIC PRO Compilerのアクティベートと設定 ―――― 7
2-1、PBP のアクティベート …… 7
2-2、MicroCode Studio を管理者として実行する設定 …… 9
2-3、PBP のデアクティベート（アクティベーションの解除） …… 10
2-4、MicroCode Studio の基本的な設定 …… 11
2-5、ユーザー登録 …… 15
～コラム～ ―――― 16
3．基本的な使い方と、簡単なチュートリアル ―――― 17
3-1、MicroCode Studio の基本画面 …… 17
3-2、はじめの一歩　～チュートリアル導入編～ …… 18
4．いろいろなチュートリアル ―――― 23
4-1、ポート単位での H-L 制御 …… 26
4-2、IF 文を使ったプログラム …… 28
4-3、SELECT CASE 文で比較演算子を使用する …… 31
4-4、RS-232C 経由で受信したデータを文字列として LCD に表示するプログラム …… 33
4-5、TMR0 の割込を用いたプログラム …… 39
4-6、AD コンバーターを使用したプログラム …… 41
4-7、32 ビット変数 long 型を使用するプログラム …… 43
4-8、PIC12F1822 のプログラム …… 45
［コラム］デバッグの基本とコードの移植 ―――― 47
5．PICBASIC PRO Compilerの変数と配列 ―――― 49
5-1、変数の種類 …… 49
5-2、変数や配列のメモリー配置について …… 49
5-3、32 ビット長変数の使い方 …… 50
5-4、変数に値を代入する …… 50
5-5、変数同士の算術演算 …… 51
5-6、変数の各ビットを参照する …… 51
5-7、変数のオーバーフローを検出する …… 51
5-8、変数のスケーラー …… 52
5-9、配列の使用 …… 53
5-10、定数の使用 …… 55
5-11、文字列にレジスタやレジスタの各ビットを割り当てる（アサイン） …… 55
6．PICBASIC PRO Compilerの演算機能 ―――― 56
6-1、各進法の表現 …… 56
6-2、算術演算子と論理演算子 …… 56
6-3、算術演算の方法 …… 59
6-4、ビット演算と論理演算 …… 60
…… 60
6-5、比較演算子 …… 60
7．PICBASIC PRO Compilerのその他の機能と使用上の設定 ―――― 62
7-1、動作周波数設定と DEFINE …… 62
7-2、AD コンバータ機能、コンパレータ機能を搭載した PIC マイコンを使用する場合の注意 …… 62
7-3、内蔵発振子搭載のデバイスを使用する場合 …… 63
7-4、ポートの入出力方向設定 …… 63
7-5、12 ビットコアの PIC や 8 ピンデバイスを使用する場合の注意 …… 64

7-6、周辺機能のピンが PPS（Peripheral Pin Select）によって選択できるデバイスを使用する場合 ……64
7-7、マルチステートメント ……65
7-8、ラベルとコメント行 ……65
8. ディレクティブ ――――――

9-40、OWOUT（ワンワイヤインターフェイスでデータを送信）……103
9-41、ON INTERRUPT（各種割込を有効にする）……103
9-42、PAUSE（指定した時間プログラムを一時停止）……105
9-43、PAUSEUS（指定した時間プログラムを一時停止）……105
9-44、PEEKCODE（指定した番地のプログラムメモリーの値を取得）……106
9-45、POKECODE（指定した番地のプログラムメモリーに値を書き込む）……106
9-46、POT（ポテンショメーターの値を取得）……107
9-47、PULSIN（パルス幅を計測）……108
9-48、PULSOUT（パルスを出力）……108
9-49、PWM（ソフトウエア PWM 出力）……109
9-50、RANDOM（疑似乱数の発生）……109
9-51、READ（EEPROM からデータを読み込む）……110
9-52、READCODE（コードスペースからデータを読み込む）……110
9-53、REPEAT....UNTIL（条件式に従い繰り返し処理）……110
9-54、RESUME（割込先サブルーチンからメインルーチンへ復帰）……111
9-55、RETURN（サブルーチンからメインルーチンへ復帰）……111
9-56、REVERSE（ピンの入出力方向の反転）……111
9-57、SELECT CASE（条件式による複数の分岐）……111
9-58、SERIN2（ソフトウエア非同期式シリアル通信 UART でデータを受信）……112
9-59、SEROUT2（ソフトウエア非同期式シリアル通信 UART でデータを送信）……117
9-60、SHIFTIN（クロック信号に同期してデータを受信）……120
9-61、SHIFTOUT（クロック信号に同期してデータを送信）……121
9-62、SLEEP（デバイスを指定時間一時休止モードにする）……122
9-63、SOUND（トーン又はホワイトノイズの発生）……122
9-64、STOP（プログラムの強制停止）……123
9-65、SWAP（2 変数間でデータの交換）……123
9-66、TOGGLE（ピンの状態を反転）……123
9-67、USBIN（USB2.0 機能内蔵のデバイスにおいて USB 経由でデータを受信）……123
9-68、USBINIT（USB2.0 機能内蔵のデバイスにおいて USB モジュールの初期化）……123
9-69、USBOUT（USB2.0 機能内蔵のデバイスにおいて USB 経由でデータを送信）……124
9-70、USBSERVICE（内蔵 USB2.0 ユニットをアクティブ化）……124
9-71、WHILE....WEND（条件ループ）……130
9-72、WRITE（内蔵 EEPROM に値を書き込む）……130
9-73、WRITECODE（コードスペースに値を書き込む）……131
10. 製品のサポートについて ―――――――

(C)2023 マイクロテクニカ
●不許複製（当方の許可無く本書の一部又は全部を複製及び転写をすることはできません）
●2023年2月版 （PBP Ver.3.1版）

●Copyrights and Tradmarks

PICBASIC PRO Compilerは米microEngineering Labs,Inc の商標です。
PICmicroは米Microchip Techbology Incの登録商標です。
Microsoft Windowsは米国Microsoft Corporation及びその国における登録商標です。
MS-DOSは米国Microsoft Corporationの登録商標です。

※PICBASIC PRO Compilerは米microEngineering Labs,Incの開発製品です。

1．PICBASIC PRO Compilerのインストール

1-1、PICBASIC PRO Compilerとは・・・

PICBASIC PRO Compiler（以下PBPと略して記載）はPIC10F、12F、16F、18Fシリーズの8ビットPICマイコンに対応したBASICコンパイラです。
PBPはBASICで書かれたソースプログラムからPICマイコンに書き込める形式であるHEXファイルを作ります。このHEXファイルをPICマイコンライターでPICマイコンへ書き込むことで動作します。
PBPにはPICマイコンのプログラミングをする上で便利な命令を数多く用意しています。PICマイコンで行う各動作はほとんどが命令化されており、ユーザーは複雑なプログラムを書くこと無く、簡単にPICマイコンにやらせたいプログラムを記述することができます。
PBPに搭載してある命令を使用すれば直感的にプログラムを記述することができPICマイコンを思い通りに動作させることができます。その他、レジスタに値を変数と同様の方法で直接値を代入したりレジスタの値を参照できたりするのでより高度なプログラミングを行うことができます。

C言語よりも簡単な言語体系でより動作内容をイメージしやすいのが特徴です。Windows上で動作する統合開発環境を搭載していますので操作性もよく初めての方でもすぐに開発を始めることができます。また対応しているデバイスが多いのも魅力の1つです。

1-2、PICBASIC PRO Compilerをインストールしましょう

PBPをパソコンにインストールしましょう。

PBPでは3つのソフトウエアをインストールします。1つはPBP本体、2つ目はPBPを動作させるための統合開発環境MicroCode Studio(以下、MCSと記載)のインストール、3つ目はPICマイコン用の純正開発環境MPLAB X IDEのインストールです。MPLAB X IDEは内蔵されているMPASM Xというソフトを利用するためにインストールが必要です。

下記のインストールの手順に従って指示通りにインストールしてください。

PBPでの開発はMicroCode Studioというソフトウエア上で行います！

PBPとMicroCode Studio(MCS)との関係ですが、PBPはコンパイラ本体でプログラムを書くときに使うエディタ画面などは持っていません。そこで、エディタ画面や各種ツールをまとめて搭載する、MCSというソフトを使います。MCSとPBPは連携して動きますので、開発はMCS上で行うことになり、PBPをユーザーが目にすることはほとんどありません。

では、MPLABとは何でしょうか？ MPLABはマイクロチップ社から配布されているPICマイコン用の開発環境です。アセンブラ言語やマイクロチップ社のリリースしているC言語などを使うときに使います。よって、PBPを使う場合にはほとんど目にしませんが実はPBPがコンパイルをする時、PBPはアセンブラファイルまでを作成します。このアセンブラファイルを最終的にPICマイコンに書き込めるHEXファイルに変換するのがMPLABの役目です。（実際にはMPLABの中にあるMPASMというソフトが担当しています。）

MCSはPBPやMPLABをまとめて動かしてくれますので開発をするユーザーはPBPやMPLABの動きを意識することなく使うことができます。しかし仕組みを知っておくと何かのトラブルの際役立つので参考までに記載しました。

① PICBASIC PRO Compiler本体のインストール

1 CD-ROMをパソコンのCD-ROMドライブに挿入してください。

2 CD-ROMの内容を開き、"PBP31_Setup.exe"をダブルクリックして実行します。
インストールウィザードが起動しますので、そのまま"Next>"をクリックして続行します。

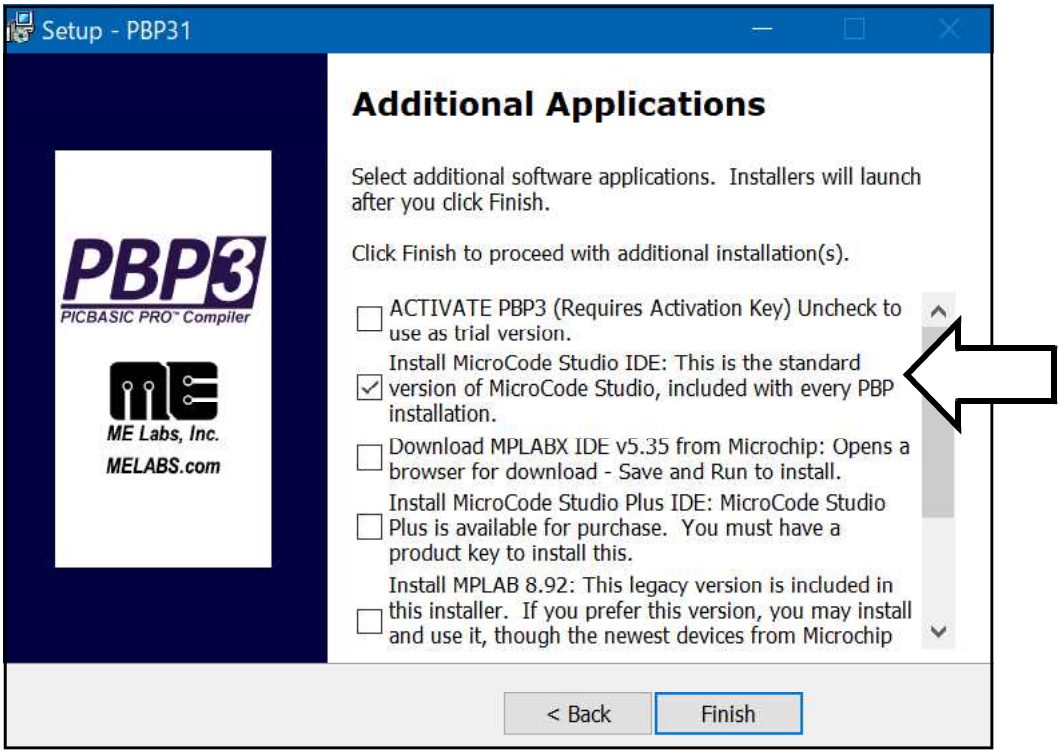
3 ライセンス許諾画面が表示されます。内容をお読みの上、同意される場合には
"I accept the agreemet"にチェックを入れて"Next>"をクリックします。
※同意されない場合、インストールはできません。

4 続いてインストールディレクトリを指定します。
デフォルトでは、C:\PBP3となっています。デフォルトの設定でインストールされることを強くお奨めします。フォルダ名を変更した場合には、後から一部のインクルードファイルのパスを変更するなど設定が面倒になります。また、日本語や2バイト文字を含むディレクトリにインストールすると、PBPが正しく起動しません。ディレクトリを指定したら、"Next>"を押して続行します。

5 次の"Select Start Manu Folder"はそのまま"Next"をクリックして続行します。

6 続いて、"Ready to Install"と表示されますので、そのまま"Install"ボタンを押してインストールを開始します。
インストールの進捗状況がプログレスバーによって表示されます。

7 インストールが完了すると下図のような画面が表示されます。
一番上の"Install MicroCode Studio IDE"にチェックが入っていることを確認します。入っていない場合にはチェックを入れます。その他のチェックはすべて外してください.



チェック項目は間違えないようにご注意ください。チェックは1つだけです！

"Finish"ボタンをクリックします。

② 統合開発環境、MicroCode Studio(MCS)のインストール

8 MicroCode Studio(MCS)をインストールします。MCSの"Setup"が起動します。
"Next>"をクリックして続行します。

9 "License Agreement"のダイアログが表示されます。内容をお読みいただき、同意される場合には、"I accept the agreement"にチェックを入れて、"Next>"ボタンを押して続行します。

10 インストールディレクトリを指定するダイアログが表示されます。
特に変更がなければ、そのまま"Next>"ボタンを押してください。インストールが開始されます。

11 完了したら、"Finish"ボタンを押して終了します。
続いて、MPLABのインストールを行います。"Install MPLAB IDE X.xx(required)"のチェックを入れて、MPLABをダウンロードした方は、下記より続いてインストールを行ってください。チェックを入れなかった方は、インストールはここで完了ですので続いて設定に進んでください。

③ MPLABX-IDEのインストール

12 MPLABXはマイクロチップ社が提供するPICマイコン用の開発環境です。
PICのプログラム開発に必要な色々なソフトウエアが詰め合わせになっています。
PBPでは直接MPLABは使用しませんが、コンパイルする時（BASIC言語で記述されたプログラムをマシン語のHEXファイルに変換することをコンパイルといいます。）にアセンブラの機能（MPLABXに含まれているMPASMというソフトウエア）を使用するため、必ずインストールが必要です。

MPLAB X IDEのバージョンに注意！

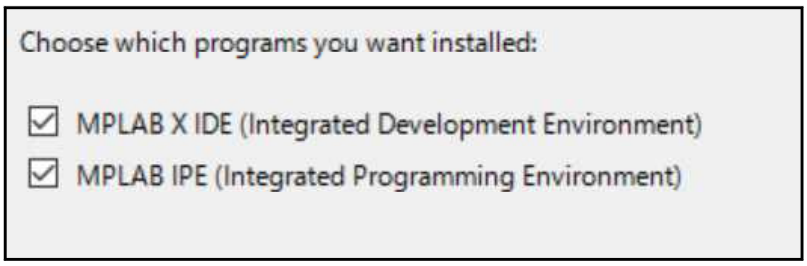
PBP3.1で利用できるMPLAB X IDEのバージョンは**Ver.5.35**までです。
最新版はマイクロチップ社のページで配布されていますが、PBP3.1を利用する場合には必ず5.35をご利用ください。(※付属のDVD-ROMに収録されているものを必ずご使用ください。)

MPLABXのインストールを開始します。
CD-ROM内の"MPLABX"フォルダを開いて，セットアップファイルを実行してください.

13　"License Agreement"の画面が表示されます。内容をお読みの上、同意される場合には"I accept the agreement"にチェックを入れて、"Next>"をクリックします。
なお、同意されない場合インストールは続行できません。

14　インストールディレクトリの指定するダイアログが表示されます．通常はデフォルトのままでインストールを続行してください．
なお，インストールしたディレクトリは後で設定の際，必要となりますのでどこかに控えておいてください．"Next>"をクリックして続行します。

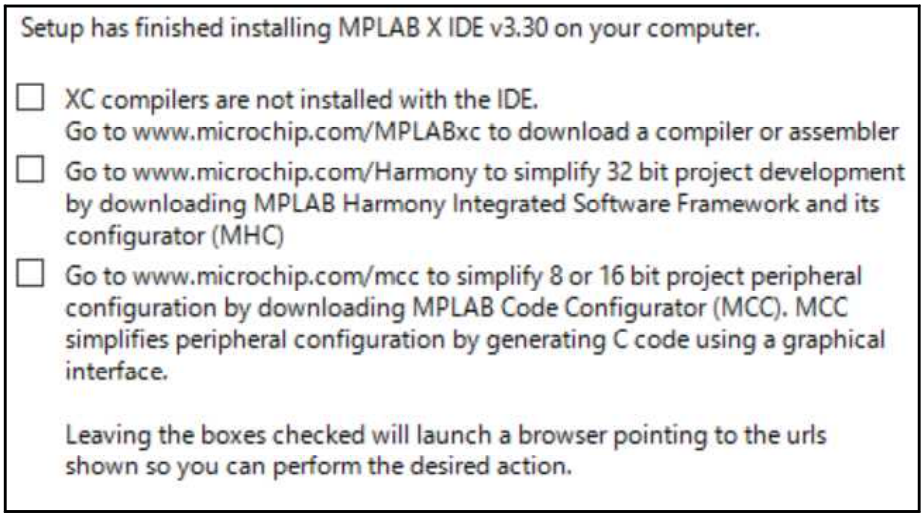
15　続いてインストールオプションを選択します．
下図のように2つにチェックを入れた状態で"Next>"をクリックして続行します．
さらに次ぎのウインドウでも"Next>"をクリックして続行します．
インストールが開始されますので，完了まで待ちます．



16　インストール途中で下図のようなドライバーインストールのダイアログが表示される場合があります．この場合には，"インストール"をクリックして続行してください．

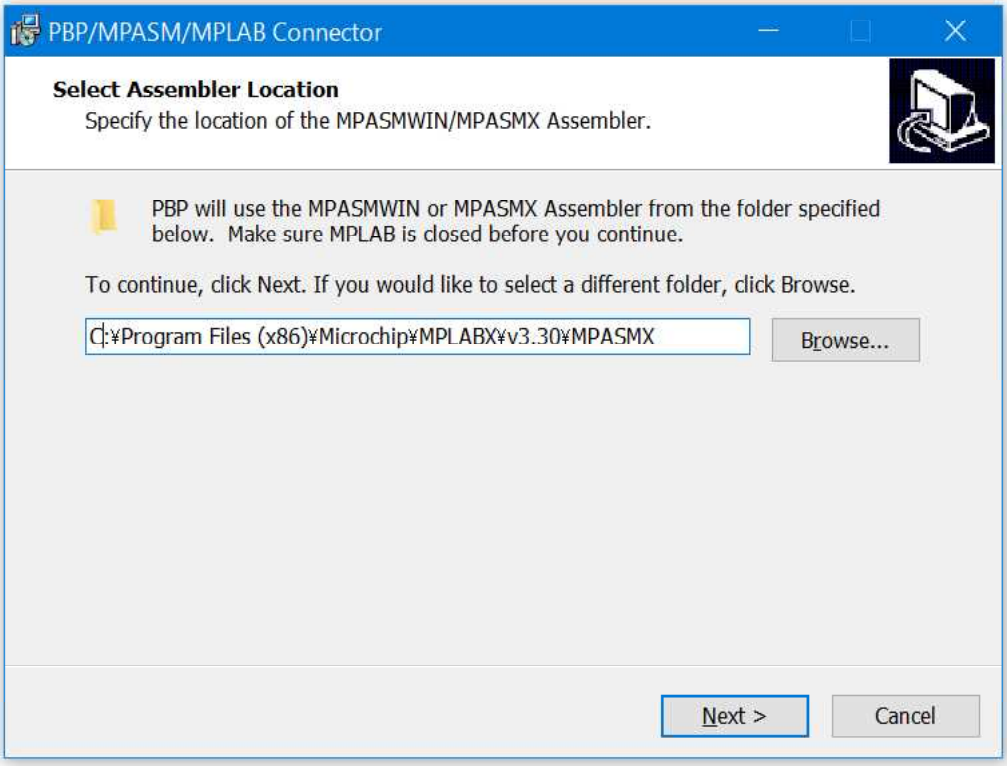
17　インストールが完了すると，下図のようなダイアログが表示されます．
いずれもMPLABX IDEに組み込むマイクロチップ社純正のCコンパイラ，XCコンパイラや32ビットPIC用のフレームワークといったツール類のインストールを続行するために，マイクロチップ社のサイトに接続するかどうかを聞く物です．

PBP3ではいずれのソフトウエアも必要ありませんので，<u>すべてのチェックを外して</u>，"Finish"を押します．



18 パソコンを再起動してください
再起動しないで設定をすると正しく動作しないことがあります．
パソコンを再起動すると，下図のようなダイアログが表示されます．

"Select Assembler Location"ダイアログは，PBP3とアセンブラであるMPASMXをバインド（関連づける）ために設定をします．ディレクトリ表示に手順14でインストールした「MPLABX IDEのインストール先フォルダ+MPASMX」となっていることを確認してください．誤りがある場合にはMPASMXのディレクトリを手動で指定します．



通常は，そのまま"Next>"をクリックして続行します．
PBP3とMPASMをバインドするプログラムが自動的に実行されます．"Finish"ボタンが表示されたらクリックして完了します．

なお，ここまでの手順で上記の"Select Assembler Location"ダイアログが表示されなかった場合には，手動で実行する必要があります．次の手順を実行してください．

①Windowsのスタートボタンをクリックし，すべてのプログラムを表示します．

②"PBP3 from melabs"グループを展開して，その中にある"MPASM-MPLAB Setup"をクリックして実行します．上記の画面が表示されますので，"Next>"をクリックして続行します．

1-3、PICBASIC PRO Compiler本体の種類

コンパイラには、2つの種類があります。"PBPW"と"PBPL"があります。
主に対応デバイスによって使い分けますが、最も大きな違いは、PBPWは使用できる変数の最大値がWORD型（16ビット長）まで、PBPLは、最大値がLONG型（32ビット長）までとなっている点です。

■PBPW
全デバイスに対応するモードで、変数は3つの型が使えます。最も大きな変数の型はWORD型で16ビットの値を格納できます。符号には対応していません。

■PBPL
変数に符号付き32ビット変数のLONG型(-2147483648～2147483647)が使えるモードです。
PIC18シリーズでのみ使用することが可能で、PIC10F/12F/16Fでは使用できません。

どちらのモードを使用するかは、MicroCode Studio(MCS)で設定が可能です。
MCSの"Compile and Program Options"で設定します。

2. PICBASIC PRO Compilerのアクティベートと設定

2-1、PBPのアクティベート

PBPは、インストール直後は15日間のみ動作するトライアル版として動作しています。アクティベートすることで、製品版として使用できるようになります。
PBPのアクティベートには、使用するパソコンがインターネットに接続されていることが必要です。インターネットに接続されていないパソコンではアクティベートができません。

1ライセンス（1つのシリアルナンバー）でPBPは3回までインストールが可能です。
アンインストールする際に「デアクティベーション」をして登録を解除すると回数はリセットされます。アクティベーション回数を減らさないようにするため、ソフトウエアをアンインストールする場合には、アンインストールしてしまう前にデアクティベーションをしてください。

使用に際しては、次の手順で必ずアクティベートを行ってからご使用ください。

PBPをパソコンからアンインストールする際に、デアクティベーション（アクティベーションの解除）を行えばアクティベーションの回数を減らさずにパソコンからPBPをアンインストールできます。
パソコンを買い換えた時や、別のパソコンで使いたい時など元のパソコンからPBPをアンインストールする場合には必ずアンインストール前に、デアクティベーションの操作を行ってください。（デアクティベーションについては本書に記載しています。）

※デアクティベーションを忘れるなどしてアクティベーションの数を超過した場合
本ライセンスでは1つのシリアル番号で最大3台のパソコンのアクティベートが可能です。アンインストール時にデアクティベーションをすれば回数は減りませんが、デアクティベーションを忘れてPBPをアンインストールしてしまうと、アクティベーションできる回数が1減ってしまいます。これを繰り返して、アクティベーションを3回行ってしまうと、それ以降そのシリアル番号ではアクティベーションをすることができません。その場合にはシリアル番号のリセットが必要ですので、下記サイトでお手続きください。

http://www.microtechnica.tv/faq/faq.cgi?kate=PBP3&faq=4

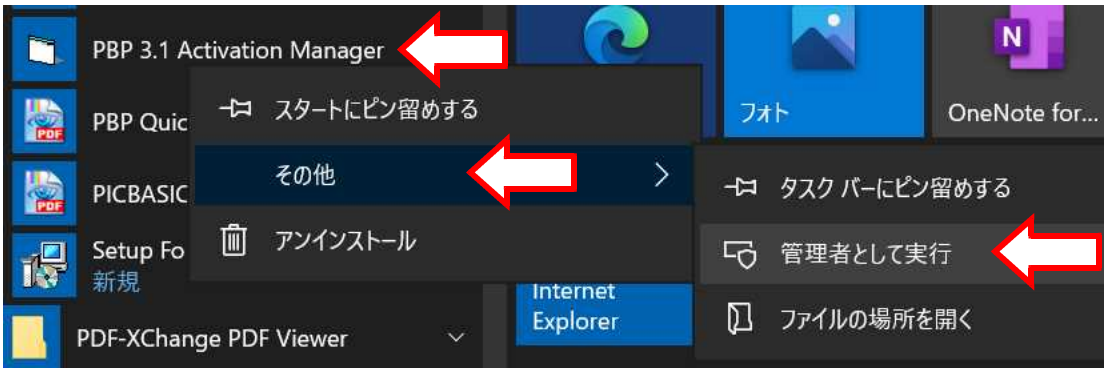
ウイルス対策ソフトウエアが有効の状態でアクティベーションを行うと、まれに正しくアクティベーションができない場合があります。念のためお手数ですが、アクティベーションの作業をする際には一時的にウイルス対策ソフトウエアの機能を無効にして頂けますようお願い致します。
アクティベーション完了後、ウイルス対策ソフトウエアを有効にしてください。
なお、アクティベーションがうまく実行できない（アクティベーションをしたにも関わらず体験版として15日限定使用版として動作してしまう場合）には、お手数ですが下記のFAQをお読みの上、お試しください。

http://www.microtechnica.tv/faq/faq.cgi?kate=PBP3&faq=3

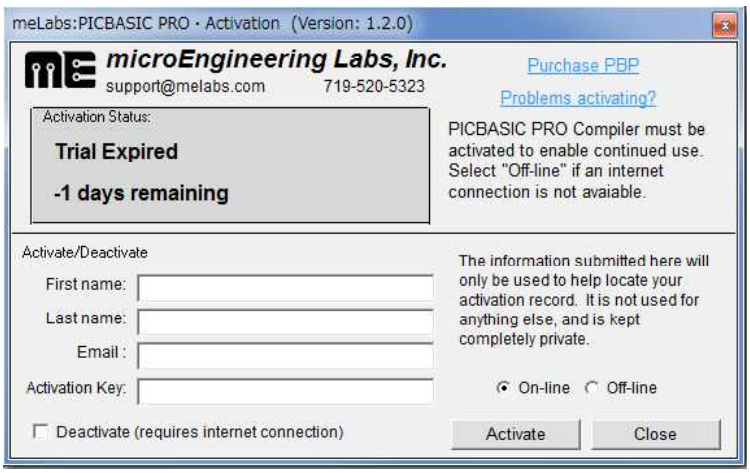
1 タスクバーのWindowsマークをクリックしてスタートメニューを表示します。

2 “P”グループ内の“PBP3.1 from Me Labs,Inc”をクリックし展開します。

3 その中の“PBP3.1 Activation Manager”を右クリックしてメニューを表示させ、“その他”→“管理者として実行”をクリックして起動させます。



4 起動すると同時に、次のようなダイアログが表示されます。

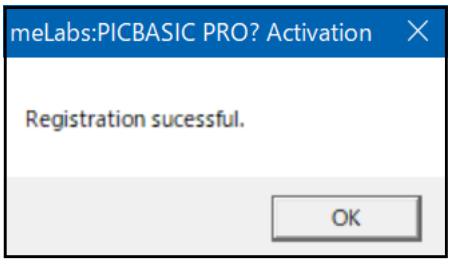


必要事項を入力します。

First name	名前（ローマ字表記）
Last name	名字（ローマ字表記）
Email	Eメールアドレス
Activation Key	製品マニュアルと同梱されていたカードに記載のアクティベーションキー

※すべて半角英数字で入力してください。

6 "On-line"にチェックが入っていることを確認して、"Activate"ボタンをクリックします。
正しくアクティベートが完了すると、"Registration Sucessful"というメッセージが表示されます。



"OK"を押して完了します。
この作業は、1つのパソコンで初回起動時の1回だけです。

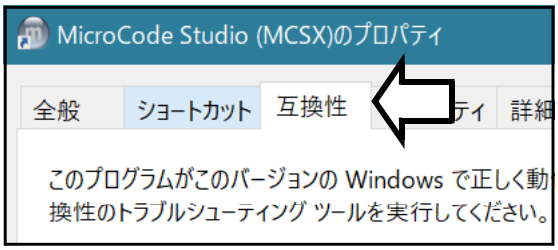
シリアル番号は今後、再インストールの際などに必要となりますので紛失されませんよう十分にご注意ください。

2-2、MicroCode Studioを管理者として実行する設定

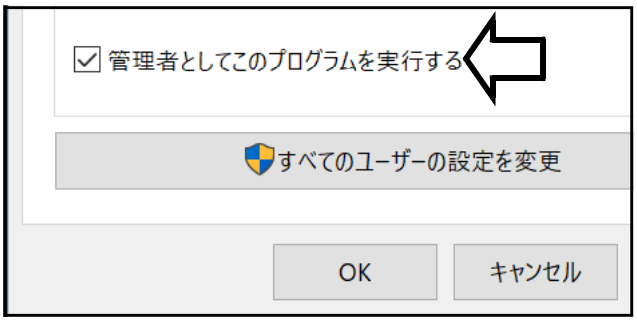
1 デスクトップにMicroCode Studioのショートカットアイコンが作成されていますので、これを右クリックしてメニューを表示して、「プロパティ」を選択してください。

←右クリック！

2 プロパティが表示されますので、「互換性」タブに移動します。



3 一番下にある「管理者としてこのプログラムを実行する」にチェックを入れてOKボタンを押して完了してください。



2-3、PBPのデアクティベート（アクティベーションの解除）

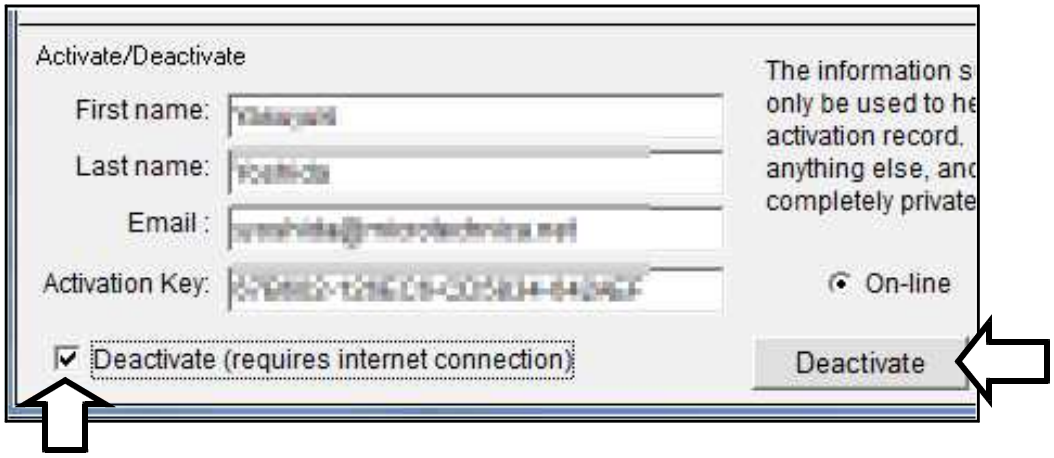
パソコンを変更する場合や、OSを再インストールする際、その他ハードウエアの大きな変更（CPUの変更やメモリーの増設など）を行う場合には、使用しなくなる予定のパソコンでアクティベーションを解除することで、そのシリアルキーのパソコンカウント数をリセットすることができます。

アクティベーションの解除を行わないで、使用パソコンを交換すると古いパソコンも1カウントされていますので、使用できるパソコンの台数が減ってしまいます。1つのシリアル番号で最大3台までのパソコンのアクティベーションができますが、パソコン買い換え時等に解除を忘れると、1つのシリアル番号で使えるパソコンの最大数が1つ減ってしまいますのでご注意ください。

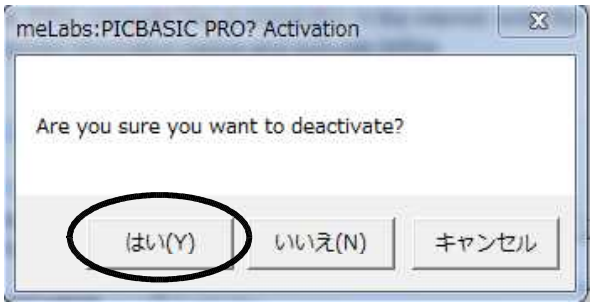
アクティベーションの解除は、PBP3を使わなくなったパソコン（PBPをアンインストールするパソコン）で下記の通り行います。

1 スタートボタンから、"すべてのプログラム"→"PBP3 from melabs"→"PBP Activation Manager"を起動します。アクティベーションマネージャが起動します。

2 すでにアクティベーションが行われている場合には、名前・メールアドレス・シリアル番号が入力された状態で起動します。
"Deactivate"にチェックを入れて、"Deactivate"ボタンをクリックします。



3 デアクティベートをしていいかどうか尋ねるメッセージが表示されますので、"はい"をクリックします。



4 これで解除が完了しました。そのパソコンではPBP3はご使用頂けません。
新しいパソコンで、再度アクティベーションを行って頂く必要があります。
なお、デアクティベーションをせずにアンインストールしてしまい、アクティベーション回数が減ってしまい上限の3回に達してしまった場合には下記サイトからアクティベーション回数のリセット申請ができます。

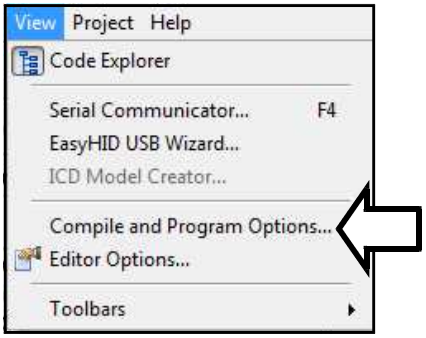
http://www.microtechnica.tv/script/deactivation/deactivation_app.html

2-4、MicroCode Studioの基本的な設定

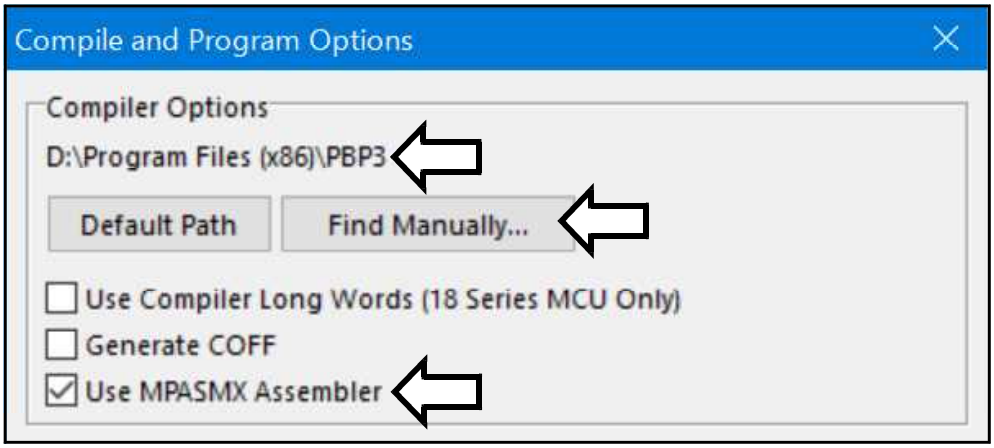
インストールが完了しましたので、MicroCode Studio(MCS)にて各種設定を行いましょう。
MCSでは、コード記述後にボタン1つでコンパイルができる他、当方のPICマイコンライターシリーズ(EPICシリーズ)及び、PICマイコン開発用統合評価ボード(PICD-700SX)をお持ちの場合には、コンパイルとプログラムの書き込みを連動させて動作させることができる便利な機能があります。
本章では、MCSが使用できるように基本的な設定を行います。
当方の製品EPICシリーズ及びPIC開発用統合評価ボードをお持ちの方は、本章をお読みの後、各製品に付属のマニュアルに記載されています"MicroCode Studioとの連動動作"の項目をお読みください。

1 MCSを起動します。"スタート"ボタンから"プログラム"を選択し、"MicroCode Studio(MCSX)"グループの中の"MicroCode Studio"をクリックします。
又はデスクトップのアイコンをクリックして起動します。

2 メニューバーの"View"から"Compiler and Program Options..."をクリックします。



3 ダイアログが開きます。PBPがインストールされているディレクトリを指定します。

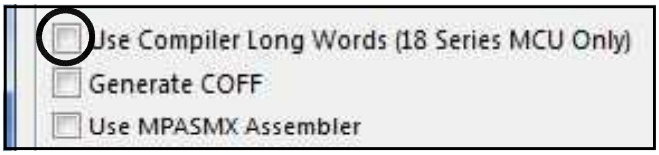


PBP3がインストールされているディレクトリを指定します．PBP3のインストールされているディレクトリが正しく表示されているか確認してください．

もし，インストールディレクトリが表示されていない場合や，誤ったディレクトリになっている場合には，"Find Manually"ボタンをクリックして，PBP3をインストールしたディレクトリを指定します．

また，"Use MPASMX Assembler"にチェックが入っていることを確認してください．
チェックが外れている場合にはチェックを入れます．

4 その他の項目はデフォルトの状態でかまいませんが、PIC18シリーズを使用する場合で、Long型変数を使用したい場合には、このウインドウの"Use Compiler Long Words(18 Series MCU Only)"にチェックを入れます。ここにチェックを入れることで、PIC18シリーズを使用する場合に限り、32ビットのLong型変数（-2147483648～2147483647）の変数が使用できます。

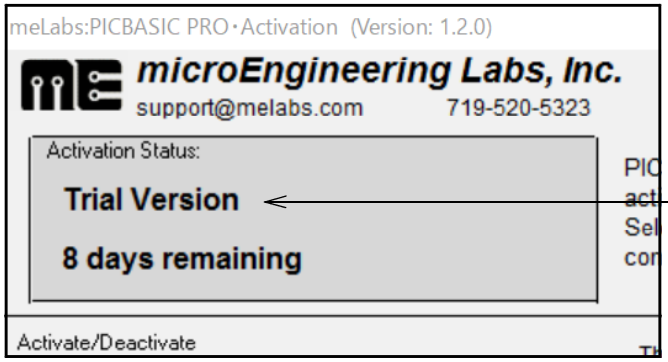


※PIC10F/12F/16Fシリーズを使用する場合には、Long型変数は使用できませんので、必ずここのチェックは外して下さい。チェックを入れた状態でPIC18以外のデバイスを開発しようとするとエラーが表示されます。Long型変数を使用する場合のみ、その都度チェックを入れるようにします。

これで基本的な設定は完了しました。

アクティベーションが完了しているはずなのにMicroCode Studioを起動した時に再度アクティベーションの画面が表示され、トライアル期間と表示される場合の対応方法

上記の手順4が完了して、正しくアクティベーションが完了しているはずにも関わらず、MicroCode Studioを起動すると手順2の、アクティベーションウインドウが表示されてしまうケースが報告されています。そのアクティベーションウインドウでは、トライアル期限であることが表示されており、正しくアクティベーションが出来ていないことが示されています。



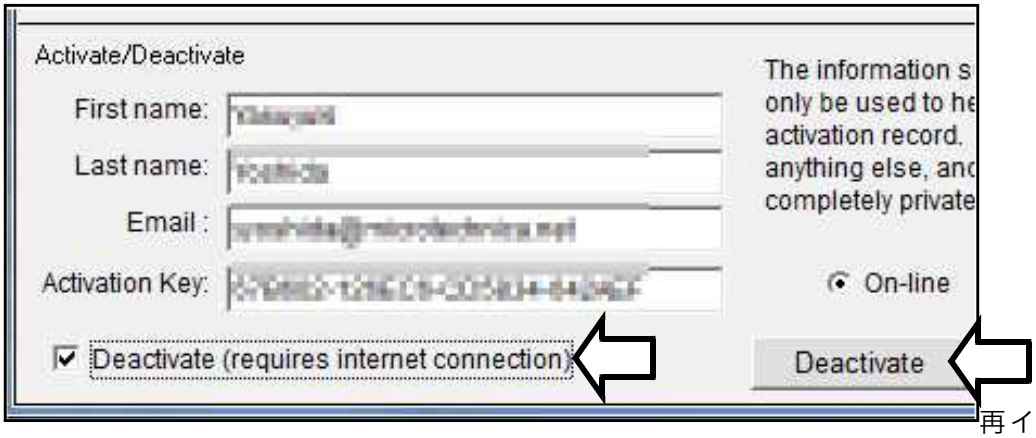
トライアルバージョンであることが表示されています。

本件は、主にパソコンのユーザー権限の問題や、ウイルス対策ソフトウエアによってWindowsのレジストリ変更が拒否された場合に発生します。
その場合には大変お手数ですがPBP3の再インストールを下記の手順にて行うと問題が解決します。

1.PBPの再インストールをお願い致します。これでほとんどの場合、問題が修復されます。
再インストールする場合には、現在インストールされているPBPをいったんアンインストールしますが、この時アクティベーションを解除する「デアクティベーション」が必要です。それをしないでアンインストールしてしまうと、アクティベーションできる回数が減ってしまいます。下記の方法で一度アクティベーションの解除を行ってください。

スタートボタンから、"すべてのプログラム"→"PBP3 from melabs"→"PBP Activation Manager"を起動します。アクティベーションマネージャが起動します。

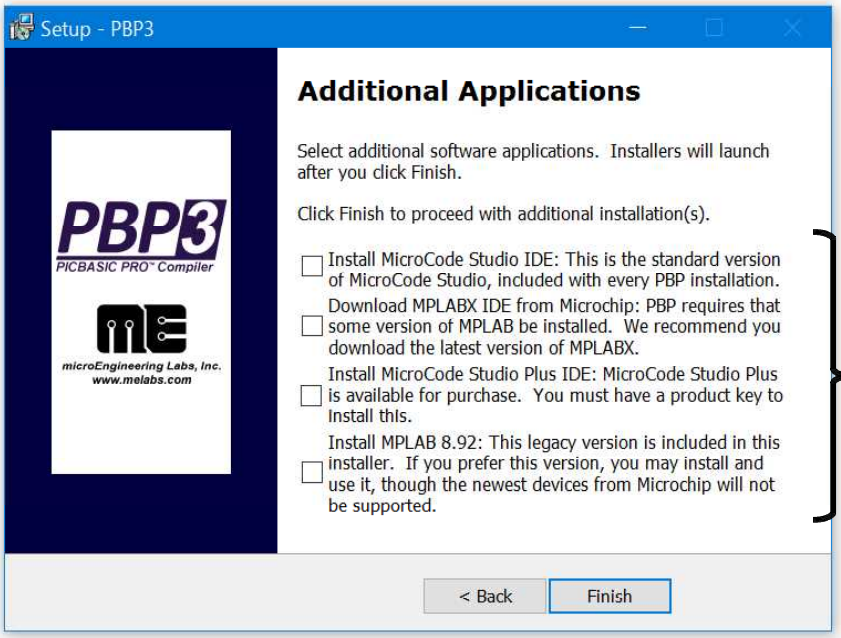
"Deactivate"にチェックを入れて、"Deactivate"ボタンをクリックします。



再インストールはPBPのみで結構です。MCSやMPLABの再インストールは必要ありません。
Windowsの「コントロールパネル」から「プログラムと機能」（又はプログラムのアンインストール）を起動して項目の中から「PICBASIC PRO(tm) Compiler 3.x.x.x」を選択して「アンインストール」ボタンを押してください。

アンインストールが完了したら、再度CD-ROMからPBP3のインストールを実行してください。

すでにMCSとMPLABXはインストール済みですのでインストール最後に表示される"Additional Aplications"の画面では4つともチェックを外して完了してください。



2.再インストール完了後、管理者権限でアクティベーションソフトを立ち上げます。
スタートボタン→すべてのプログラム→PBP from melabsをクリックします。

3.その中の“PBP3.1 Activation Manager”を右クリックしてメニューを表示させ、“その他”→“管理者として実行”をクリックして起動させます。

4.アクティベーションソフトが起動しますので必要事項を入力して"Activate"ボタンをクリックしてアクティベーションを行って下さい。これで完了します。

※上記方法でもMicroCode Studioを起動するごとにアクティベーションソフトが起動してしまう場合にはパソコンにインストールされているウイルス対策ソフトウエアによってアクティベーションソフトの通信が遮断されているために、インターネット経由で認証に不具合が生じていることが考えられます。その場合、ウイルス対策ソフトウエアを一時的に停止した上で実行すると正しくアクティベーションができます。

また、場合によってはPBP3のインストール時にウイルス対策ソフトウエアがPBP3の通信関係のソフトウエアのインストールをブロックしてしまう場合があり、その場合正しく登録ができないなどの問題が発生することがあります。もし上記方法でも正しくアクティベーションができない場合にはお手数ですが、一時的にウイルス対策ソフトウエアを終了（又は機能を停止）した上で再度、PBP3をインストールし直して頂くことで問題が解決する場合があります。

Windowsに常駐しているすべての常駐ソフトウエアとサービスを停止した状態でWindowsを起動し、アクティベーションを実行して頂くと問題は解決します。

http://www.microtechnica.tv/faq/faq.cgi?kate=PBP3&faq=3

2-5、ユーザー登録

設定が終わりましたらユーザー登録を行って頂けますようお願い致します。

■開発元サイト(microEnginerring Labs)のサイトに登録する

1 下記サイトにアクセスします。
http://store.melabs.com/LOGN.html

2 画面上部にある"Create New Account"のリンクをクリックします。

3 次のページで必要事項を半角英数字で、英語で記入して頂きます。次のように入力必要事項を入力します。
・Login: ログイン名を付けます。適当なIDを入力してください。
・Email Lost Passwords To: パスワードを忘れた場合に必要となるﾒｰﾙｱﾄﾞﾚｽを入力します。
・Password ログインするためのパスワードを決めて入力します。
・Confirm Password: 確認のためもう一度パスワードを入力します。

その下の部分に必要事項を入力します。入力例は下記の通りです。

Login:	HHANAKO
Email Lost Passwords To:	hanako@email.com
Password:	●●●●●●●●
Confirm Password:	●●●●●●●●
Ship To:	
First Name:	Yamada
Last Name:	Hanako
Email Address:	hanako@email.com
Phone Number:	+81-3-3700-3535
Fax Number:	
Company:	
Address:	1-3-10 Tamagawa
Address line-2:	
City:	Setagayaku
State/Province:	Outside US
Other State/Province:	Tokyo
Zip/Postal Code:	1580094
Country:	Japan

※Bill To (If Different): の部分には入力しないでもかまいません。

4 入力したら"Save"ボタンをクリックします。
次の画面が表示されれば登録完了です。

～コラム～

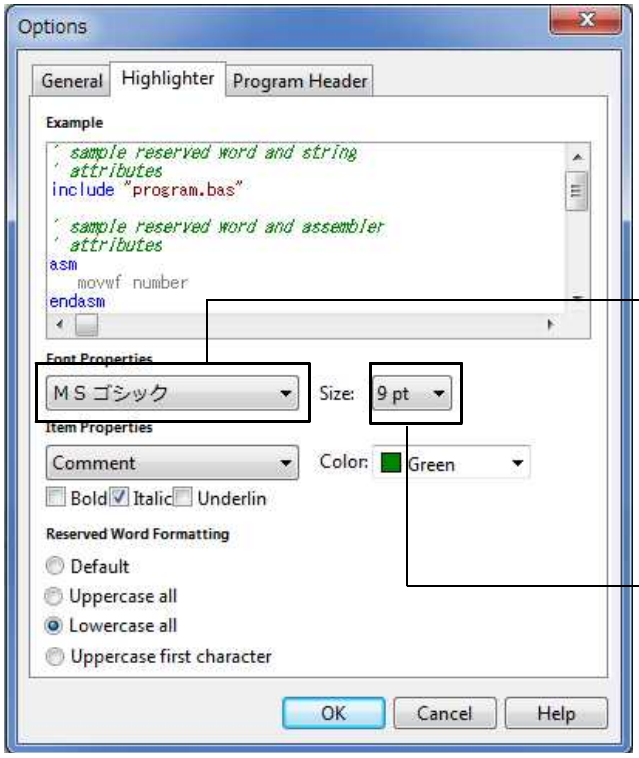
MicroCode Studioで日本語コメントを使用する場合の設定方法
及びエディターフォントのサイズを変更する方法

MicroCode Studio(MCS)ではプログラム中にコメントを挿入できますがデフォルト設定ではエディタのフォントが英語フォントになっているため、日本語や2バイト文字は入力ができません。
下記の設定をすることで、日本語のコメントを入力できます。またフォントサイズを変更することで、文字を大きくして見やすくすることができます。

【注意】

本設定によって日本語入力を可能にした際、アポストロフィー(')以降でない、すなわちコメント扱いされない部分に誤って日本語を入力してしまうと、場合によってはMCSがハングアップしたり、強制終了してしまう場合があります。安全に使用したい場合には上記の設定はデフォルト設定にして日本語は使わないようにしてください。また、プログラム中（コメント部分でない部分）では全角スペースは日本語として扱われます（2バイト文字）ので、スペースは必ず半角で入力する必要があります

■日本語のコメントが入力できるように設定する



MCSのメニューバーの"View"→"Editor Options..."をクリックします。ダイアログが表示されますので、"Highlighter"タブへ移動します。その中の"Font Properties"のプルダウンから"MSゴシック"を選択してください。

フォントサイズを変更することで文字が大きくなり、見やすくなります。文字のサイズはデフォルトが9ポイントですので、14ポイントくらいまでの範囲で設定してください。

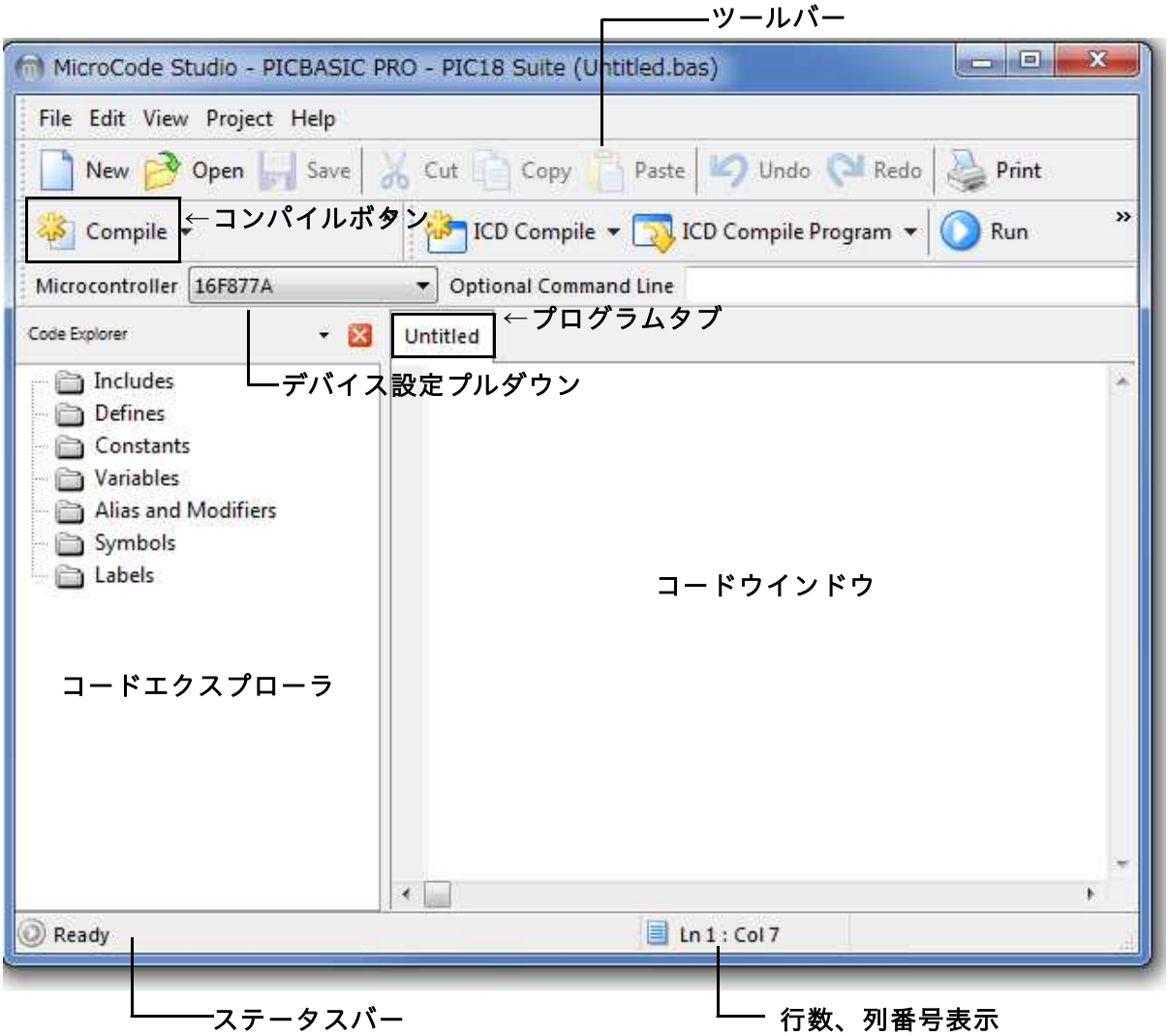
"MSゴシック"に設定すると、アポストロフィー(')以降のコメントに限り日本語入力が可能になります。なお、"Font Properties"のデフォルト設定は"Courier New"フォントです。

3. 基本的な使い方と、簡単なチュートリアル

PBPには色々な機能や命令、文法上の規則などがありますが、ここではとりあえずMCSの使い方を把握するためにサンプルのプログラムを作成してコンパイルしてみましょう。
MCSの使い方とプログラムの書き方、コンパイルの要領を確認します。

3-1、MicroCode Studioの基本画面

MCSのウインドウは下記のようになっています。



※ツールバーの状態は、選択されているデバイスや、インストール状態になどによって必ずしも上図のようになっていない場合があります。

※下図のようにツールバーに"Buy PICBasic"とツールバーに表示されている場合には、正しくPBPのアクティベーションができていないことを示しています。クリックするとオンラインストアーに接続されてしまいます。この状態では、PBPは限定されたデバイスしか使えず、製品版として登録されていません。再度、本書に記載の方法でアクティベーションを行ってMCSを再起動し、"Buy PICBasic"のボタンが消えていることをご確認下さい。

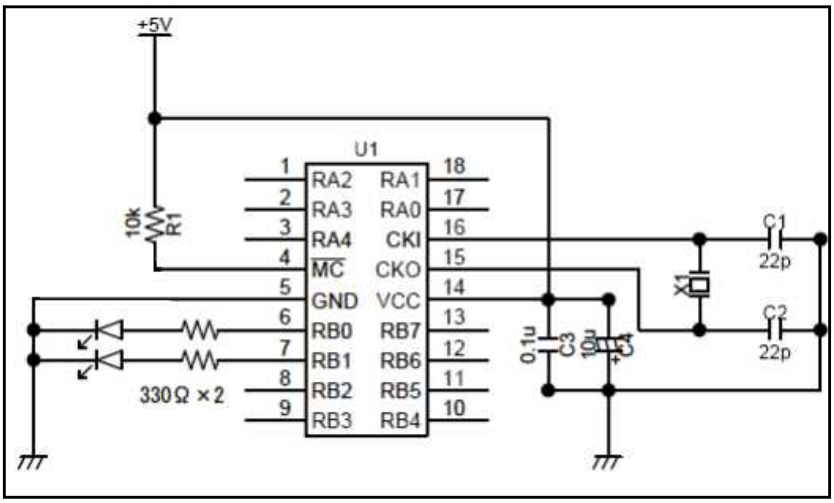
"Buy PICBasic"と表示されている場合には正しくアクティベーションができていません。
お手数ですが再度アクティベーションをお試しください。

3-2、はじめの一歩　～チュートリアル導入編～

では、実際にプログラムを記述してプログラム開発の流れを体験してみましょう。
ここで紹介するチュートリアルは、例として「１秒間隔でPORTBのビット0(RB0)とビット1(RB1)を交互に点滅（HとLを繰り返す）させるプログラム」を作ります。

※PICマイコンの場合、慣習としてPORTBをRB、PORTAをRAなどとRをつけてポート名を記述します。
PORTBのビット0という場合には、RB0と表現するのが一般的です。本マニュアルでもこの表現にて以降記載します。

RB0とRB1に下図のようにLEDを接続しておくと、動作が目で見え理解が簡単です。
参考回路図を下記に示します。X1はクロックですが例として8MHzとします。



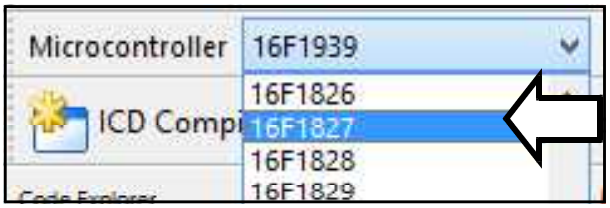
図：PIC16F1827(18ピンPIC)を用いた標準回路

PICマイコンのロジック電圧のH時の出力レベルは電源電圧とほぼ同一ですので、Hレベルの時は上図の場合には+5V、Lレベルの時はGNDレベルです。LEDの順方向電圧が2V、消費電流が10mAとした場合には300Ω程度の抵抗が必要です。上図では、330Ω程度の抵抗を取り付けています。
※上図は参考例ですので使用するマイコンの型式等によって接続方法が変わる場合があります。

なおこのチュートリアルで作るプログラムは下記の環境で動作させることを前提としています。

・ PIC16F1827を使用
・ 動作周波数は8MHz

1　MCSを起動します。
最初に開発するデバイスの種類を選択します。"Microcontroller"のプルダウンから使用するデバイスの種類を選択してください。ここでは"16F1827"を選択します。



※MCSでは、PBPがサポートするすべてのPICマイコンのデバイス名をプルダウンに表示します。もし、プルダウンに数種類のデバイスしか表示されない場合には、MCSにPBPの実行ファイルを正しく登録されていないか、正しくアクティベーションができていないことが原因です。再度設定を確認する必要があります。設定をし直してください。

2　プログラムを記述します。下記のように記述しましょう。
※プログラムウインドウには自動的にプログラムの概要を記述するコメント行が挿入されていることがあります。必要ない場合には削除してください。

1827_test

```
define OSC 8

ANSELB = 0

main:   high PORTB.0
        low PORTB.1

        pause 1000

        high PORTB.1
        low PORTB.0

        pause 1000

        goto main
```

```
define OSC 8

ANSELB = 0

main:   high PORTB.0
        low PORTB.1

        pause 1000

        high PORTB.1
        low PORTB.0

        pause 1000

        goto main
```

大文字と小文字は基本的に区別されませんが最初の行の"Define OSC"の"OSC"は必ず大文字で記述してください。スペースや改行、タブは無視されます。プログラムを見やすくするために適宜タブやスペースを入れることをお奨めします。

■DEFINE OSC 8
DEFINEはPBPにデフォルトで設定されている設定内容について、ユーザーが定義を変更する時に使用します。"DEFINE OSC"では動作周波数を指定します。単位はMHzです。
ここでは動作周波数は8MHzなので、"8"と記述します。
なおDEFINEの後ろに続く文字列は必ず大文字で記述します。よって"OSC"は大文字で記述します。

PLLで逓倍した時の周波数設定はどうする？

PLLが搭載されているデバイスではPLLを有効にした場合には逓倍された後の周波数を記述します。例えば外部周波数8MHzを使用していて4xPLL機能を有効にすると32MHz動作となります。この場合には "DEFINE OSC 32" と記述します。

内蔵発振子を使った場合の周波数設定はどうする？

内蔵発振子搭載のデバイスを使う場合コンフィギュレーションビットの設定で発振子の種類を"INTOSC"などの設定ににする他、プログラム内で"OSCCON"レジスタの設定をして動作周波数を指定します。PBPでは必ずOSCCONで設定した動作周波数に合わせて"DEFINE OSC"を記述してください。例えばOSCCONで8MHzとした場合には、"DEFINE OSC 8"とします。

■ANSELB = 0
ANSELBレジスタはPIC16F1827においてPORTBレジスタの各ビットについて、デジタルIOに設定するか、ADコンバータ用のアナログ電圧入力とするかを設定するレジスタです。
該当ビットが0でデジタルIOとなります。ここでは0を代入することでPORTBすべてのビットをデジタルIOとして設定しています。（この記述を忘れるとPORTBデジタルIOとして動きません。）

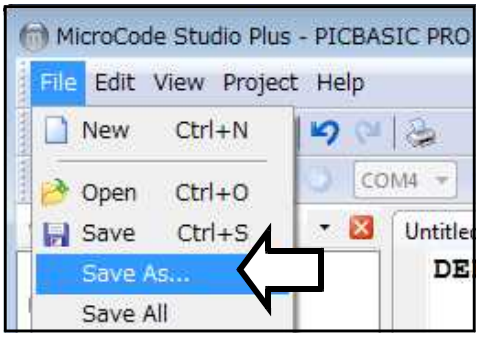
■main:
任意の文字列に：（コロン）を付けるとラベルになります。ラベルは、プログラムの中でどこにでも作れます。ラベルを作成するとGoto命令によって、任意のラベルに処理をジャンプさせることができます。但し、ラベル名に予約語は使用できません。予約語とは、PBPで用意されている命令や関数に使われる文字列です。予約語をラベル名として使用するとコンパイル時にエラーとなります。

■High PORTB.0
Highは、任意のポートをHighレベルにする命令です。ここではPORTB.0を指定しています。
LowはHigh命令の反対で任意のポートをLowレベルにします。
PBPでは、ピンの指定は、「ポート名．ビット名」で記述します。

■Pause 1000
任意の時間プログラムを一時停止します。単位はミリ秒です。この場合は1000ミリ秒、すなわち1秒プログラムが停止します。

■Goto main
指定したラベルへジャンプします。ここではmainラベルへジャンプしますのでルーチンが無限にループする構造になります（永久ループと呼びます）。

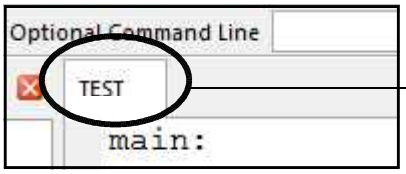
3 コンパイルを実行前に作成したソースプログラムに名前を付けて保存します。
メニューバーの"File"から"Sava As"をクリックします。



"名前を指定して保存"ダイアログが表示されます。
インストールディレクトリは任意の場所でかまいませんが、ディレクトリ名に日本語などの2バイト文字が入るディレクトリは使用できません。長いディレクトリ名（全部で16文字以上）は使用できません。ファイル名にも日本語、全角文字などの2バイト文字は使用できません。

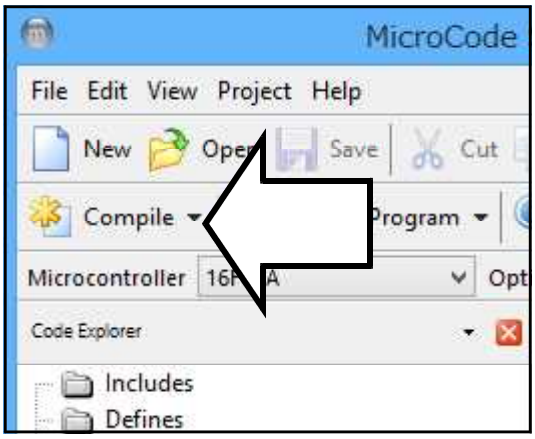
できる限り、 "C:¥PBP3¥Apps" などの簡単なディレクトリに保存してください。

※マイドキュメントやユーザーフォルダーへの保存は、ディレクトリ名に日本語が含まれていたり、パスが長くなることがあるためお奨めできません。また、最近各種ファイルをクラウドストレージに保存するサービスが増えています。マイクロソフト社が提供するOne Driveや、Dropboxなどがありますが、PBPではこれらのストレージサービスにソースファイルを保存することはできません。クラウドストレージでは生成されるファイルのシンボルと、ファイルのアップロードに時間差があり、コンパイル時に多くの中間ファイルを作るPBPではクラウドストレージにソースファイルを保存してコンパイルすると、コンパイルの途中でファイルが開けなくなり、コンパイルに失敗します。
PBPのソースファイルは必ずローカルストレージに保存してください。



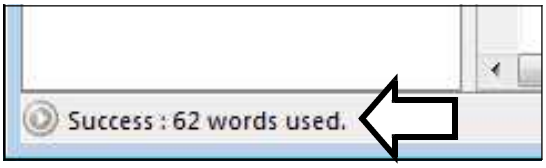
ファイル保存後は、タブにファイル名が表示されることを確認してください。

4 コンパイルはボタン一発で行えます。コンパイルを行うとPBPはBASIC言語で書かれたプログラムからHEXファイルを作ります。ツールバーの"Compile"ボタンをクリックします。
コンパイルが開始されます。キーボードの「F9」キーでもコンパイルができます。



コンパイル中は数秒で終わります。

5 コンパイルが正常に完了すると、下図のようにステータスバーに"success: ○○words used."と表示されます。これで問題なくHEXファイルが生成されました。HEXファイルは、ソースファイルが保存してあるディレクトリと同じディレクトリ内に作られます。
記述ミスがあったり構文に問題があると問題箇所が表示されますので、プログラムを再度確認してください。



6 できあがったHEXファイルを、PICマイコンライター等でPICマイコンへ書き込んでください。
プログラムの内容通りに動作することを確認します。

入力した予約後が勝手に小文字に変換されてしまう・・・

命令等の予約語を記述すると入力した文字が大文字・小文字に関係なくすべて小文字又は大文字に自動変換される場合があります。これは、予約語を自動的に大文字や小文字に変換するオートスタイラが搭載されているためです。設定は次の方法で変更できます。

1. メニューバーの"View"→"Editor Options"をクリックします。
2. "Highlighter"タブをクリックします。
3. "Reserved Word Formatting"の所にあるチェックボックスを次のよう選択します。
   - ・Default: 予約語は小文字になります。
   - ・Uppercase all: 予約語はすべて大文字になります
   - ・Lowercase all: 予約語は小文字になります。
   - ・Uppercase first character: 予約語の文字列先頭のみ大文字になります。

4.設定を変更したら"OK"を押して完了します。

プログラミングに際してはとりあえず次の点に注意してください！

①動作周波数は必ず「DEFINE OSC 周波数値」の書式で記述してください。
→20MHzの場合には、、、、　DEFINE　OSC　20
【重要】DEFINE OSCの「OSC」は必ず大文字で記述してください。

②多くのPICマイコンでは、ADコンバータ機能やコンパレーター機能を搭載しています。これらのデバイスは、レジスタの設定をしないとデフォルトの状態では、I/Oピンの設定がアナログ入力ピンに割り当てられています。よってデジタルI/Oピンとして使用するためには、レジスタを設定して該当ピンをデジタルモードにする必要があります。これを行わないと、プログラムは期待通り動作しません。

③命令に大文字と小文字の区別はありません。（ただしDEFINE文は大文字で！）
その他スペースやタブ、空白行についてもPBPでは無視されます。

次のページからはいくつかのチュートリアルを通して、PBPプログラムの書き方を練習します。プログラムを実際に記述することでPBPの各命令の記述方法や命令の種類などを把握することができます。最初にチュートリアルを行う前に重要な規則を確認します。PBPではプログラムを記載するにあたり主に次のような注意が必要です。

- プログラム先頭で動作周波数を設定する。設定は、［DEFINE OSC xx］で行う。
- その他デフォルトの設定を変更する場合はDEFINEを使う。DEFINEの内容は各命令の項を参照。
- 変数は好きな名前が使用できる。但し予約語は使用不可。
- 変数には4つの型があるので適切な型を選ぶ。宣言は、［変数名 var 型］である。
- 変数はピンアサインにも利用できる。［任意のピン名 var ポート］で指定できる。
- コメントは、'（アポストロフィー）以降の部分に書く。
- コメントはなるべく詳しく、後から見ても分かるように書く。
- ラベルは：（コロン）で記述する。任意のラベルへのジャンプはGOTO命令を使用する。
- ピンの状態を判断する場合には、IF文を利用する。例：IF PORTB.0 = 0 Then　など
- ボタン入力を判定したい場合には、IF文を用いる。必要に応じてGOTOを利用しループする。

なお、これ以降に記載されているサンプルプログラムは、一般的な例を示していますので使用するデバイス（ターゲットデバイス）や、動作させる環境（動作周波数等）に応じて、適切な設定に設定し直す必要があるかもしれません。

DEFINEに続く文字は必ず大文字で記述！！

PBP3には各種定義をするためのDEFINEというキーワードがあります。例えばクロック周波数などを定義する DEFINE OSC や、非同期シリアル通信(UART)の通信速度を定義する　DEFINE HSER_BAUD などです。

これらのDEFINEに続く文字は必ず大文字にするという決まりがあります。
大文字で記述していないとPBP3は正しくその定義を認識しません。

DEFINE hser_baud 9600　　←小文字で記述しているので認識しない！！

DEFINE **HSER_BAUD** 9600　　←大文字で記述しているので正しく認識する

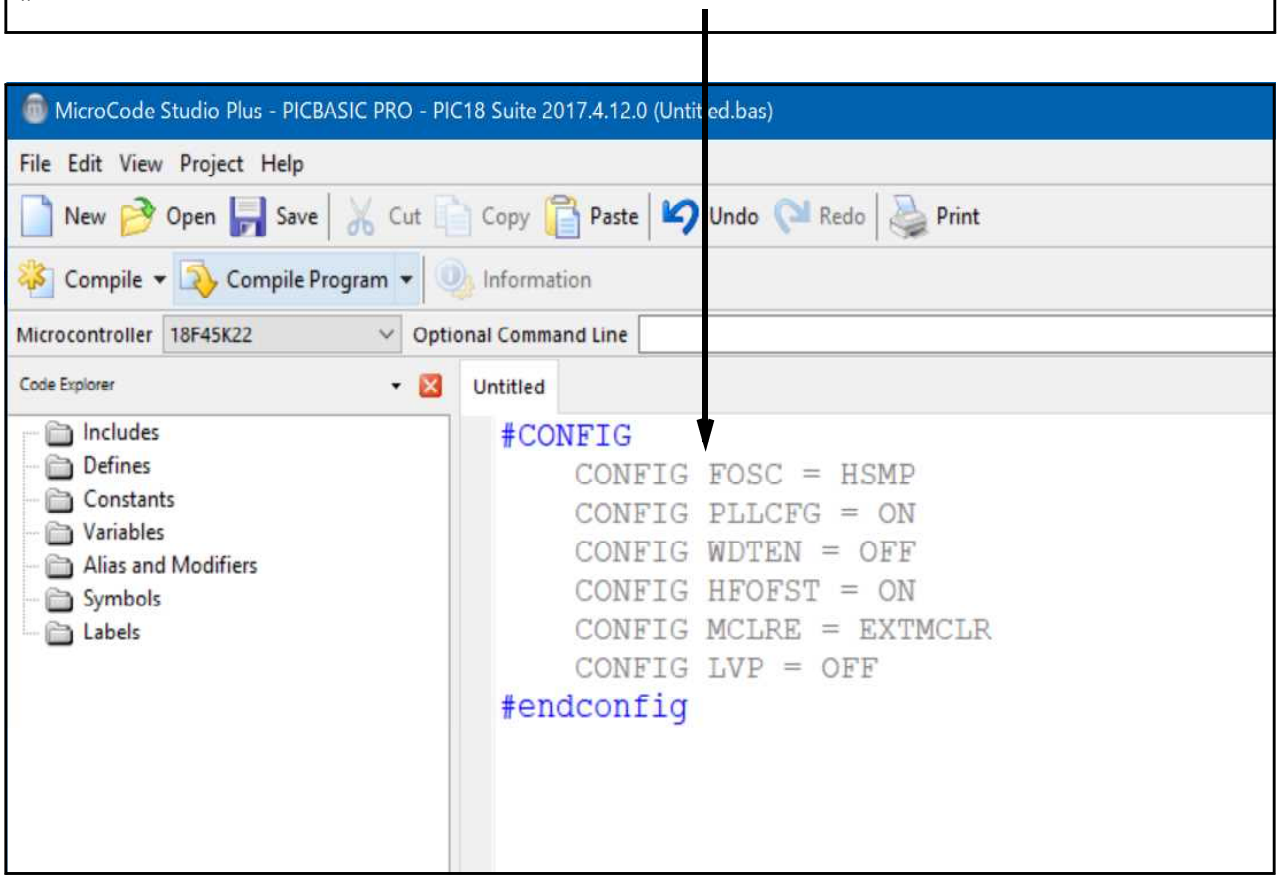
"DEFINE全文は必ず大文字"ということを忘れずにプログラミングをお願い致します。

4. いろいろなチュートリアル

4章ではチュートリアルとして色々なプログラムを作ってみます。
このチュートリアルでは**PIC18F45K22**を使用し、動作環境も同一の環境で動かすものとします。よっていずれのプログラムもコンフィギュレーションビットの設定内容は同じです。

よってどのプログラムでも共通なコンフィギュレーションビットの設定値についてのディレクティブの記述は省略してプログラムを掲載致しました。
各チュートリアルのプログラム記述では記述されていませんが、プログラムを記述する場合にはいずれのプログラムでも下記の通りコンフィギュレーションビットの設定を記述して、その下からプログラムを記述して頂けますようお願い致します。

```
#CONFIG
    CONFIG FOSC = HSMP
    CONFIG PLLCFG = ON
    CONFIG WDTEN = OFF
    CONFIG HFOFST = ON
    CONFIG MCLRE = EXTMCLR
    CONFIG LVP = OFF
#ENDCONFIG
```



#CONFIG～#endconfigは、PICマイコンのコンフィギュレーションビットの設定を記述するディレクティブです。ここにコンフィギュレーションビットの設定を記述することで、このプログラムをコンパイルして生成されたHEXファイルをPICマイコンライターのソフトで読み込むと自動的にコンフィギュレーションビットの設定が行われます。

プログラム内にコンフィギュレーションビットの設定を記述しておかなくてもコンパイルは可能です

がその場合にはHEXファイルをPICに書き込む際、書き込みソフトウエアにて都度コンフィギュレーションビットの設定を行う必要があります。都度設定するのは面倒ですし間違えてしまう場合もあるので基本的にはソースプログラムに#CONFIGディレクティブを使ってコンフィギュレーションの設定を記述することをおすすめします。

なお書式はデータシートにも記載されていますが、下記のディレクトリに定義ファイルがありますので、ご使用のデバイスの定義ファイルをメモ帳等のテキストエディタで開いてください。

C:¥Program Files (x86)¥PBP3_1¥DEVICE_REFERENCE

```
27 ;MPASM CONFIG Directive Options
28 ;
29 ;  This section is included for reference only.  The options below may be
30 ;  used with the MPASM CONFIG directive to specify configuration settings
31 ;  in the source program using the #CONFIG/#ENDCONFIG block.
32 ;
33 ;  The PBP default configuration for the PIC18F45K22 is:
34 ;
35 ;  #CONFIG
36 ;    CONFIG  FOSC = INTIO67        ; Internal oscillator block
37 ;    CONFIG  PLLCFG = OFF          ; Oscillator used directly
38 ;    CONFIG  PRICLKEN = OFF        ; Primary clock can be disabled by softwa
39 ;    CONFIG  FCMEN = OFF           ; Fail-Safe Clock Monitor disabled
40 ;    CONFIG  IESO = OFF            ; Oscillator Switchover mode disabled
41 ;    CONFIG  PWRTEN = OFF          ; Power up timer disabled
42 ;    CONFIG  BOREN = SBORDIS       ; Brown-out Reset enabled in hardware onl
43 ;    CONFIG  BORV = 190            ; VBOR set to 1.90 V nominal
44 ;    CONFIG  WDTEN = ON            ; WDT is always enabled. SWDTEN bit has n
45 ;    CONFIG  WDTPS = 32768         ; 1:32768
```

デフォルトで設定されているコンフィギュレーションビットの設定値が記述されています。
例えばPIC18F45K22の定義ファイル「PIC18F45K22.INFO」を開くと次のような記述があります。

このファイルの#CONFIG～#endconfigまでがデフォルトの設定値です。
この設定内容から設定を変更したい場合には、プログラム内に#CONFIGを使って設定値を記述してください。プログラム内の#CONFIGで記述した内容がデフォルトの設定値に上書きされてHEXファイルが作られます。設定値一覧については上記ファイルに記載されています。

このチュートリアルで記述した設定値は次のようになっています。

CONFIG FOSC = HSMP
　→発振子は8MHz以上の外部発振子

CONFIG PLLCFG = ON
　→PLLを有効に設定、(外部発振子8MHzの場合には32MHzになる)

CONFIG WDTEN = OFF
　→ウォッチドッグタイマーは無効

CONFIG HFOFST = ON
　→内部オシレータの高速起動を有効

CONFIG MCLRE = EXTMCLR
　→外部リセットピンMCLRをリセットピンに設定

CONFIG LVP = OFF
　→低電圧書き込み機能を無効

ここではオシレータの種類を外部発振子として、ウォッチドッグタイマーはOFF、MCLRピンによるリセット入力を有効、PLL逓倍は無効に設定しています。

以下のチュートリアルでは共通した内容なので記述を省略しています。

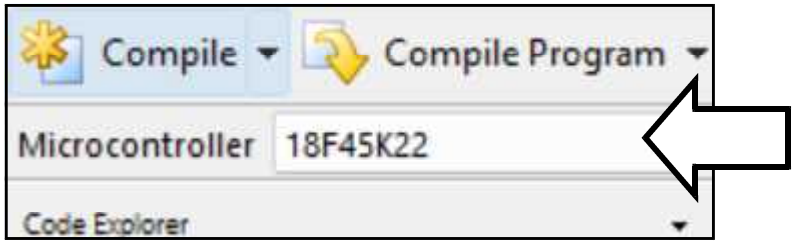
4-1、ポート単位でのH-L制御

ここでは、ポート単位で各ビットのH-Lを指定して制御する方法を紹介します。

■学習の内容
ポート単位でのHigh-Low制御を行い、PBPでのレジスタの直接操作の仕方を学びます。

■プログラムの内容
PORTBとPORTCを一定のパターンで交互に点滅動作させます。ビット単位ではなくポート単位での操作方法とポートの入出力設定の仕方を紹介します。

■デバイスの設定
デバイスをプルダウンから選びます。ここではPIC18F45K22を選択しました。
実際に使用されてるデバイスを選んでください。



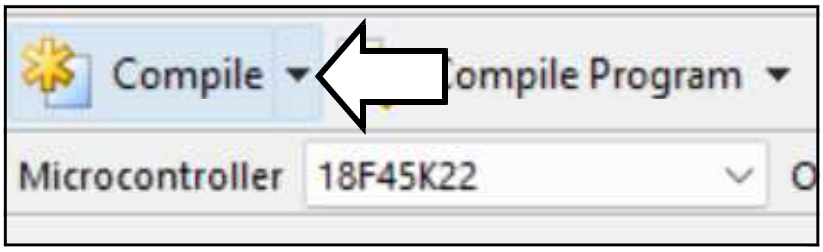
■プログラムを書いてみましょう
次のようにプログラムを書いてみましょう。
記述するのは、下の四角い枠の中です。右側は、プログラムの解説です。

```
DEFINE OSC 32

init:
    ANSELB = 0                      PORTB は全ビットデジタル IO ピン　①
    ANSELC = 0                      PORTC　　〃
    TRISB = 0                       PORTB は全ビット出力方向に設定　②
    TRISC = 0                       PORTC　　〃
    LATB = 0                        LATB レジスタに 0 を代入　③
    LATC = 0                        LATC　　〃

main:                               ラベルを定義
    LATB = %10101010                LATB レジスタに 2 進数 10101010 を代入　④
    LATC = 0                        LATC レジスタに 0 を代入
    pause 500                       500 ミリ秒待機
    LATC = %01010101                LATC レジスタに 2 進数 01010101 を代入
    LATB = 0
    pause 500

    goto main                       Loop ラベルへジャンプ　＝永久ループ
```

■コンパイルしましょう

プログラムを記述したらファイルを保存します。
"File"→"Save as"をクリックしてファイルに名前をつけて保存してください。

保存できたらコンパイルします。“Compile”ボタンを押してください。



生成されたHEXファイルは、ソースプログラムを保存したのと同じディレクトリに作られます。
そのファイルをPICライターで読み込んでPICに書き込んでください。

■プログラムの解説

① PIC18F45K22のようにADコンバータを搭載したデバイスでは電源投入時はADコンバータの入力ピンとして割り当てられているピンはすべてアナログ電圧入力になっています。
そのままでは通常のデジタルI/Oとして設定する必要があります。

このレジスタがデバイスによって異なるため少々面倒で、お使いのマイコンのデータシートを見て確認する必要があります。PIC18F45K22の場合にはANSELxレジスタで設定が可能です。
このレジスタの各ビットに1を代入するとアナログ入力、0でデジタルI/Oになります。
ここではPORTBとPORTCの全ビットをデジタルI/Oとして設定しています。
PBPでは、「レジスタ名=値」でレジスタに値を代入できます。

② TRISxレジスタはIOピンの入出力方向を設定するレジスタです。該当ビットに0を代入すると出力に、1を代入すると入力ピンになります。

③ LATxレジスタはIOピンの出力状態をコントロールするレジスタです。
該当ビットが0でLレベルに、1でHレベルになります。プログラムの初期化時に値を代入して初期化しておきます。

④ LATBレジスタに2進数で10101010を代入しています。PBPでは値を10進数・2進数・16進数で表現できます。進法の違いにより、値の前に次のような接頭辞を付けます。

・10進数の場合　　何も付けません
・2進数の場合　　数値の前に ％記号 を付けます
・16進数の場合　　数値の前に ＄記号 を付けます

よってこのプログラムでは、"LATB = $AA" としても同じことになります。

4-2、IF文を使ったプログラム

IF文を使用し状態や条件に応じて処理を分岐するプログラムです。また変数を使います。

■学習の内容

このプログラムでは任意のピンの論理状態（HかL）をIFを使って判定し、処理内容を分岐する方法について学習します。この方法は入力処理としてよく利用されます。
PBPではHの状態を1、Lの状態を0、で表現します。そのほかピンをエイリアスで宣言する方法を学習します。さらに変数を宣言してプログラム内で変数を利用します。

■プログラムの内容

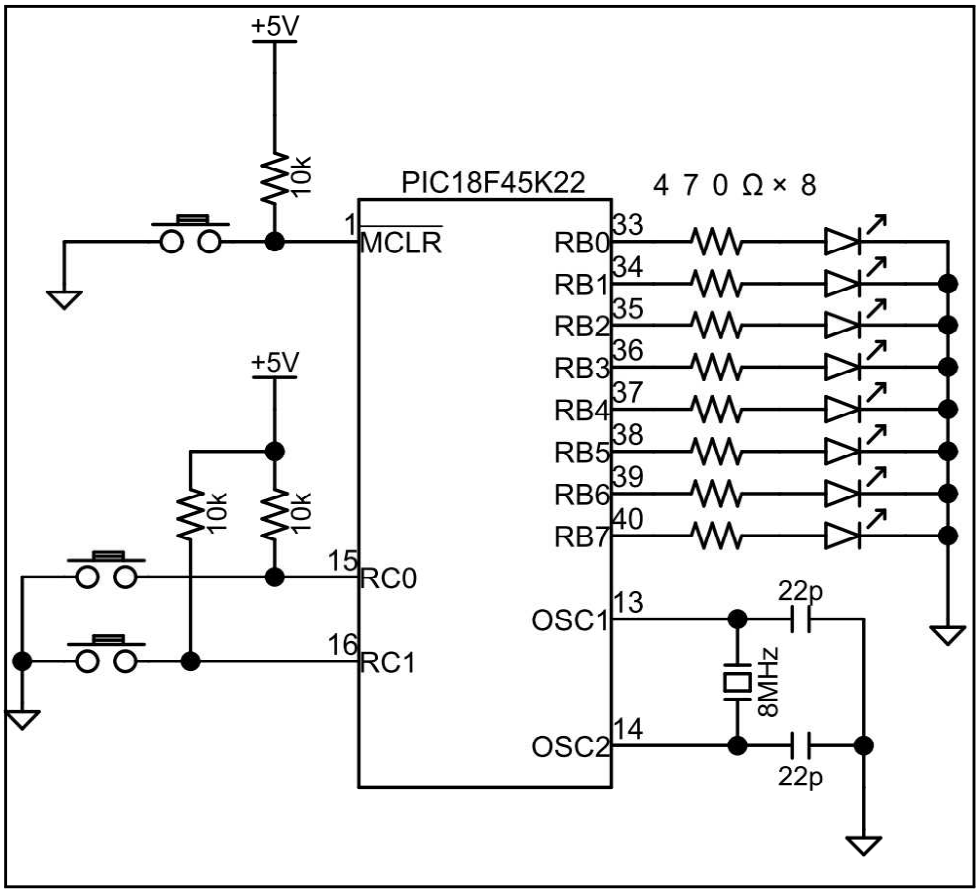
RC0とRC1をスイッチ入力として使います。
スイッチを押していない時はHレベル、スイッチを押すとLレベルになるようにアクティブローのスイッチにします。

RC0のスイッチを押すとPORTBの全LEDが500ミリ秒間隔で点滅します。
RC1のスイッチを押すとのPORTBの全LEDが100ミリ秒間隔で点滅します。
スイッチを押していない時はPORTBの全LEDはすべて消灯しています。

■プログラムを書いてみましょう

```
DEFINE OSC 32

SW1 var PORTC.0                     RC0 を SW1 という文字列にアサイン　①
SW2 var PORTC.1                     RC1 を SW2       〃

delay var word                      変数 delay を Word 型として宣言　②

init:
    ANSELB = 0
    ANSELC = 0
    TRISB = 0                       PORTB は全ビットデジタル IO ピン
    TRISC = 3                       RC0 と RC1 を入力ピンに設定、それ以外は出力　③
    LATB = 0

main:
    if SW1 = 0 then delay = 500     if 文で SW1 が 0 の時は delay 変数に 500 を代入　④
    if SW2 = 0 then delay = 100     SW2 が 0 の時は delay 変数に 100 を代入
    if (SW1=1 AND SW2=1) then       SW1 と SW2 両方が 1 の時・・・　⑤
        LATB = 0                      PORTB の全ビットは L レベル
        goto main                     ラベル main にジャンプ
    endif                             if 文終了

    LATB = $FF                      PORTB の全ビットを H にする　⑥
    pause delay                     delay 変数の値だけ遅延　⑦
    LATB = 0                        PORTB の全ビットを L にする
    pause delay

    goto main                       main に戻る
```

このプログラムの回路例を下図に記載します。



スイッチはアクティブLowにします。
LEDを接続する場合には電流制限用の抵抗を取り付けます。
この回路図では、VssとVddの処理は省略しています。

■プログラムの解説

① 任意の文字列にレジスタの任意のビットを割り当てるエイリアスと呼ばれる方法です。
ここでは文字列"SW1"にRC0を対応させています。宣言以降では"SW1"と記述することでRC0を示すことになります。同様にしてSW2というエイリアスも作りました。
エイリアスとして特定のピンを宣言しておけばプログラムを書く際に直感的に配線を意識しながらプログラムを書くことができます。

② delayという文字列を変数として使います。
ここではword型という16ビットの幅を持つ変数を宣言しました。変数の宣言は
「変数名 var 型」という書式になります。変数の型はPBP3では4つ利用できます。
2ビットのbit型、8ビットのbyte型、16ビットのword型、PIC18Fシリーズだけで使える32ビットのlong型です。

③ PORTCはRC0とRC1だけを入力にしたいので00000011となります。
これは10進数にすると3なのでTRISC=3として3を代入しています。

④ if文による判定です。if文には2つの書式がありこれは1つ目です。
「if 条件式 then 真の時に実行される文」という形式です。真の時に実行される文が1行だけの場合に使えます。
SW1、つまりRC0の物理的な状態がLレベルの時、すなわちスイッチが押された時、変数delayに500を代入しています。
同様にしてSW2がLレベルの時は変数delayに100を代入しています。
変数delayに異なる値を入れることで点滅周期を変更させています。

⑤ if文による判定の2つ目の形式です。
「if 条件式 then 真の時に実行される文・・・endif」という形式です。真の時に実行される文が2行以上の場合に使います。then～endifまでの間に書かれたプログラムが条件式が真の時実行されます。

ここでは論理演算子のAND(論理積)を使っています。このANDは複数の条件式が両方真の時に1となります。すなわち「SW1=1かつ、SW2=1」の時、両方のスイッチが押されていない時にPORTBの全ビットをLレベルにしてLEDを消灯させています。

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

左表は論理積の真理値表です。
AとBがともに真のみ結果が1となります。

論理演算をする場合には、その式を( )でくくっていることに注意してください。

消灯後は点滅する部分は実行させたくないのでラベルmainに戻ります。

if文での論理演算子はANDの論理積のほかにOR(論理和)、Ex-OR(排他的論理和)なども使えます。

⑥ $FFは255、2進数で%11111111です。
すなわちPORTBのLEDが全点灯します。

⑦ pause命令の引数に変数を指定しています。
delay変数には④のif文でスイッチの状態に応じた値が代入されています。SW1の時は500、SW2の時は100が代入されているためこの変数の値によって点滅周期が変わるようになります。
このように多くの命令では引数に適宜変数が利用できます。

4-3、SELECT CASE文で比較演算子を使用する

■学習の内容

分岐処理をする命令にはIF文の他にSELECT CASE文があります。IF文は1つの評価式の評価結果が真か偽により処理を分岐しますが、SELECT CASE文では1つの評価式に対し複数の条件を設定できます。本チュートリアルではSELECT CASE文を使用して条件式に比較演算子を使用する方法を解説します。

■プログラムの内容

RC0のスイッチを押した回数によって10回未満の時はRB0を点灯させ、10回の時はRB0とRB1を両方点灯、11回以上でRB0を消灯してRB1を点灯させます。

10回未満　○●
10回　●●
11回以上　●○

※左側がRB1、右側がRB0、●が点灯、○が消灯

■プログラムを書いてみましょう

```
DEFINE OSC 32

SW1 var PORTC.0
LED1 var LATB.0
LED2 var LATB.1
result var byte                          変数 result をバイト型で宣言

init:
    ANSELB = 0 : ANSELC = 0              複数行にわたる文は:（コロン）を使って 1 行に
    TRISB = 0 : TRISC = 1                記述することができる　①
    LATB = 1                             RB0 だけ初期値を H レベルにしておく
    result = 0                           変数 result を 0 で初期化

main:
    if SW1 = 0 then                      SW1 が押された時・・・
        result = result + 1                result の値を 1 インクリメント　②
        gosub judge                        サブルーチン jugde にジャンプ　③
        pause 300                          チャタリング防止の 300 ミリ秒遅延　④
    endif
    goto main                            main に戻る

judge:                                   サブルーチン judge ここから return まで
    select case result                   select case 文で変数 result の値を判定　⑤
        case is < 10                     result の値が 10 未満の時・・・
            high LED1                      LED1 を点灯
        case 10                          result の値が 10 の時・・・
            high LED1 : high LED2          LED1 と LED2 を点灯
        case is >= 11                    result の値が 11 以上の時・・・
            low LED1 : high LED2           LED1 を消灯、LED2 を点灯
    end select

    return                               メインルーチンに復帰　⑥
    end                                  終了
```

■プログラムの解説

① ：コロンはラベルを付ける時にも使えますが複数の行にわたる文を1行にまとめる時にも使えます。まとめて行う処理などは1文1行にするより短くプログラムを書けて便利です。

② プログラムではよく使うインクリメントの方法です。インクリメントとは変数の値を1ずつ増加させるという意味です。ここではスイッチ(RC0)が1回押されるたびに変数resultの値を1ずつ増加させています。なお変数は宣言した後、ランダムな値が入っているため使用する時は初期化する必要があります。

③ 処理を別のルーチンにまとめることをサブルーチンといいます。
gosub命令は引数に指定したサブルーチンに処理を移します。処理が終わった後はresult命令でもとのルーチンに戻ります。
goto命令は指定したラベルにジャンプしますがジャンプ元に戻ることはできません。あるまとまった処理をしたい場合や繰り返し行う処理などはサブルーチンにまとめると便利です。

④ スイッチのチャタリングによる誤動作を簡易的に防止するための遅延です。
機械スイッチは通常押したり話したりすると接点が振動し、指で1回押したつもりでも何回もスイッチが押されたり離れたりしたようになってしまいます。プログラムでチャタリングの対策をしていないと1回スイッチを押しただけで何回もスイッチが押されたようになって誤作動の原因となります。
ここでは簡易的に300ミリ秒の遅延を入れることで機械接点の振動による影響を低減させています。

⑤ select case文です。select caseの後ろに評価する変数など評価式を記述します。
その評価式の値が"case"で指定された条件式を満たす場合にはそのcase内の文が実行されます。
ここでは、変数resultを評価式として指定しており、変数resultの値によって処理が分岐されます。
以下はcaseによって判断される内容です。

case isは評価式の内容を条件式と比較する時に使用します。比較演算子<は～未満です。
変数resultの値が10未満であれば、次のcaseまでの間のルーチンを実行します。
条件式に比較演算子を用いる場合にはcase isを用います。

caseは単に変数の値がその値であった時に実行されます。
ここでは変数resultの値が10の時実行されます。

case isで11以上を判定する場合には比較演算子　>=　を使います。これは「以上」で記述した値を含みます。

select case文の最後はend selectで締めくくります。

⑥ gosub命令でジャンプしてきたサブルーチンからはretun命令でジャンプ元に処理を復帰させることができます。

4-4、RS-232C経由で受信したデータを文字列としてLCDに表示するプログラム

パソコンのRS-232Cシリアルポートと接続して非同期式シリアル通信(UART)を行うプログラムです。PICのハードウエアUART機能を使用しデータ受信時に割込を発生させてデータを受信するプログラムを作成します。

■学習の内容

RS-232C経由でパソコンから送られてきたデータをLCDに表示する方法を学習します。ハードウエアUART機能を使ってデータを送受信します。シリアル通信割込の使い方を学習します。

■プログラムの内容

通信速度9600bps、データ長8ビット、ストップビット1、パリティなし、の設定でUART通信を行います。通常時はRA0のLEDを100ミリ秒間隔で点滅させておき、UARTにてデータを受信したら割込を発生させ割込ルーチン先でデータを受信します。

パソコンから送られてきた1バイトのデータ+キャリッジリターン($0D)の計2バイトを受信して、受信したデータを16進数でLCDに表示します。
またデータを受信したら受信した値をそのままパソコン側に送信します。(エコーバック)

■プログラムを書いてみよう

```
DEFINE OSC 32

On Interrupt Goto UART_Rec          割込発生時のジャンプ先ラベルの設定　①

DEFINE LCD_DREG PORTB               ┐
DEFINE LCD_DBIT 0                   │
DEFINE LCD_RSREG PORTB              │ LCD の結線情報を定義する DEFINE
DEFINE LCD_RSBIT 4                  │ LCD の各ピンが PIC のどのピンと接続されて
DEFINE LCD_EREG PORTB               │ いるかの定義　②
DEFINE LCD_EBIT 5                   ┘

DEFINE  HSER_RCSTA      90h         受信レジスタを有効に設定
DEFINE  HSER_TXSTA      20h         送信レジスタを有効に設定
DEFINE  HSER_CLROERR    1           オーバーフローエラーを自動的にクリアー

BAUDCON1 = %00001000                通信速度設定レジスタ   BRG16 = 1
TXSTA1 = %00100100                          〃             BRGH = 1

SPBRG1 = $40                                〃
SPBRGH1 = $03                               〃             9600bps に設定　③

Rec_Data VAR byte[2]                UART の受信データバッファ配列　④

init:
    INTCON = $C0                    ペリフェラル（周辺機能）の割込を有効
    PIR1.5 = 0                      UART1 受信フラグを 0 にクリア　　⑤
    PIE1.5 = 1                      UART1 のデータ受信割込を有効に設定　⑥
    ANSELA = 0 : ANSELB = 0         PORTA,PORTB はすべてデジタル IO
    ANSELC = 0                      PORTC はすべてデジタル IO
    TRISA = 0                       PORTA はすべて出力ピン
    TRISC = %10000000               RC7 のみ入力、それ以外は出力ピン
```

```
    LATA = 0
    LATC = 0

    Lcdout $fe,1
                                        LCD クリアー　⑦
main:
    high LATA.0
    pause 100
    low LATA.0
    pause 100

    goto main

disable                                 通常ルーチン終了　⑧

UART_Rec:                               割込ルーチン（UART 受信割込発生時）

    hserin[str Rec_Data¥2¥$0D]          UART から指定した方法でデータを受信し配列に格納⑨
    lcdout $fe, 1, hex2 Rec_Data[0]     LCD に指定した形式で表示　⑩
    hserout [Rec_Data]                  UART からデータを出力　⑪

    INTCON = $C0                        INTCON レジスタを初期化
    PIR1.5 = 0                          UART 受信割込フラグを 0 にクリア　⑫

    resume                              割込ルーチンから復帰　⑬

enable                                  通常ルーチン有効　⑭
```

※上記プログラム中の￥マークはMicroCode Studio Plus上では＼（バックスラッシュ）として表示されます。これはMCSPが英語フォント環境のためです。￥マークと＼ は同じ意味ですので読み替えてください。

■プログラムの動作を確認しましょう

このプログラムの動作確認するにはPICのハードウエアUARTポートがUSB-UART変換IC(FT232など)を介してパソコンと接続され、Windows上で仮想COMポートが設定されている必要があります。COMポート番号はWindowsのデバイスマネージャで確認しておきます。

RS232C通信を行うには、パソコン側にはターミナルソフトが必要です。
有名なターミナルソフトにTeraTerm(テラターム)という定番ソフトがありますが、MCSPには標準で簡易シリアルターミナルが搭載されているのでここではそれを利用します。

MCSPのメニューバーから"View"→"Serial Communicator"をクリックします。
ターミナルソフトが起動します。

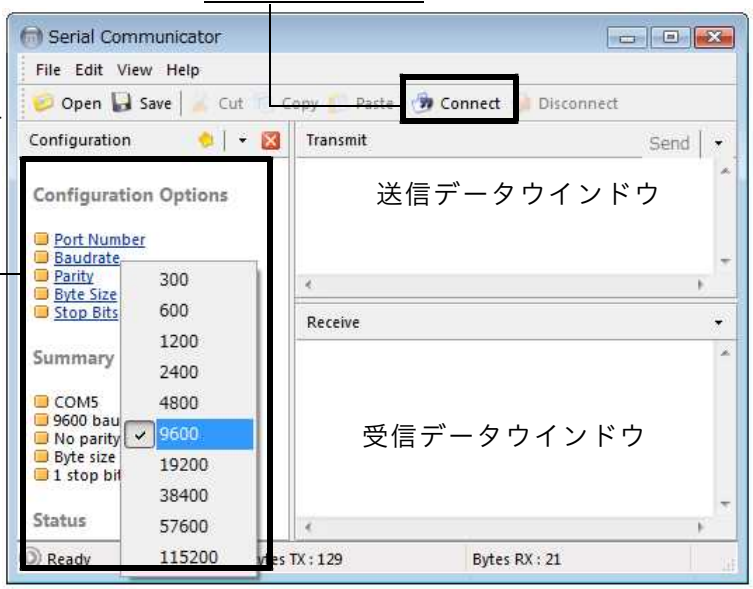
"Configuration Options"でポート番号や通信速度、パリティの有無、データ長、ストップビットなどの通信に関する設定を行います。

ここでは、Baudrateは9600bps、ParityはNone、Byte Sizeは8、Stop Bitsは1に設定します。
Configuration Optionsのところにある青い文字をクリックしてそれぞれ設定してください。
“Port Number”は仮想COMポートの番号です。パソコンで利用可能なCOMポートが表示されます。

COMポート番号はお客様の環境に合わせて設定します。

通信を開始するには、ツールバーの"Connect"ボタン、又はF9キーを押します。

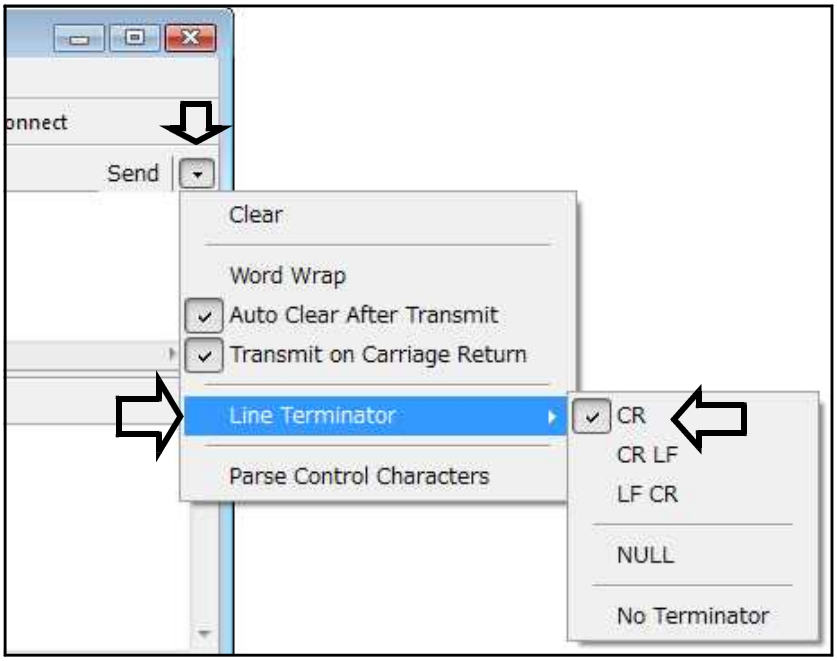
通信の設定をします



"Connect"ボタンを押すとConfiguration Optionsの画面が消えて、データウインドウが広がります。

ここではエンターキーを押した時に送信ウインドウに入力したデータが送信されるように設定します。また送信されるデータはキャリッジリターン(16進数で0D)で終端されるようにします。

"Send"の隣の▼ボタンを押してメニューを表示します。



上図の通り、"Auto Clear After Transmit"と"Transmit on Carrige Return"にチェックを入れます。

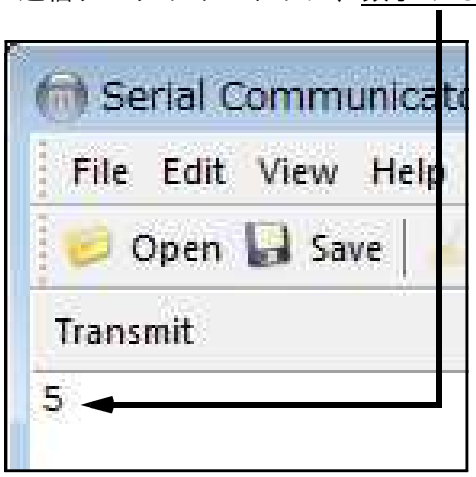
次に"Line Terminator"にマウスを当てると横にメニューが表示されるので上図の通り"CR"にチェックを入れます。

この設定でデータの終端にキャリッジリターンが自動的に挿入されるようになりました。

通信をして動作を確認しましょう。
プログラムをコンパイルして書き込んでください。

プログラムが走るとRA0が100ミリ秒間隔で点滅していることを確認します。

送信データウインドウに、数字の 5 を入力してエンターキーを押します。



LCDに　35　と表示されていれば成功です。(文字5はASCIIコード16進数で$35です)

さらに受信ウインドウに　5　と表示されれば成功です。
なおRA0のLEDはデータ受信時に若干点滅速度が変わりますがまたもとに戻ることを確認します。

その他例えば文字 A　を送信するとLCDにはASCIIコードで　41　と表示されます。他のいろいろな文字を送信してみてください。その文字のASCIIコードがLCDに表示されます。

※複数バイト送った場合には、受信した最初の文字の値を表示します。それ以外のデータは破棄されます。

■プログラムの解説

① on interrupt gotoはPBP3において割込を有効にした時、割込発生時にジャンプするラベル名を指定するキーワードです。gotoの後ろには割込発生時にジャンプするジャンプ先のラベル名を記述します。
PICでは様々な割込が用意されていますがどの割込が発生してもすべてここで指定したラベルに飛んできます。よって複数の割込を有効にしている場合には、どの割込で飛んできたのかを割込先ルーチンでif分などで判定して処理を分岐します。⑧の"Disable"とペアで使用します。

② lcdout命令では、あらかじめPICとLCDがどのように配線されているのかをDEFINEを用いて定義しておく必要があります。PBP3のlcdout命令は日立のLCDコントローラID、HD44780U及びその互換コントローラのLCDを制御できます。
ここではLCDを4ビットモードで駆動します。4ビットモードで駆動する場合はデータ線上位4ビットとRSビット、Eビットの計6線を使用します。

この例ではデータ線はRB0～RB3に、RSビットはRB4に、EビットはRB5に接続されているものとして結線情報をDEFINEで定義しています。

③ 通信速度の設定を行います。通信速度は複数の要因によって計算式が変わります。下の表はデータシートからの抜粋です。特に設定を要するのはBRG16ビット(BAUDCON1レジスタのビット3)と、BRGHビット(TXSTA1レジスタのビット2)です。これらが0か1かによって計算式が変わります。

通信速度の設定値は16ビットあり、下位バイトがSPBRG1レジスタ、上位バイトがSPBRGH1レジスタです。計算の結果8ビットを超える設定値となった場合には上位バイトと下位バイトにそれぞれ値を分けて代入します。

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n+1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

ここでは次のように計算します。
マイコンのクロック周波数は32MHz、設定したい通信速度は9600bpsです。
ここではBRG16ビット、BRGHビットにともに1をセットします。そうすると先の表より次の計算式となります。

$$設定値 = \frac{\frac{Fosc}{baud\ rate}}{4} - 1$$

※Foscはクロック周波数(Hz)、baud rateは通信速度(bps)

よってここでは次のような計算結果となります。

$$\frac{\frac{32000000}{9600}}{4} - 1 = 832.33...$$

計算結果は832.33..なので832とします。
これは8ビットの範囲を超えているので上位バイトと下位バイトに分けます。832は16進数で$0340です。よって上位バイトのSPBRGH1レジスタには$03、下位のSPBRG1レジスタには$40を代入しています。これで32MHzクロックで動作している時に9600bpsの通信ができます。
上記の式で計算すると115.2kbpsの時は68となるのでSPBRGH1=0、SPBRG1=68となります。

④ UARTで受信するデータについてはバイト型変数または配列を指定します。
1回の送受信でデータの長さが分かっている場合にはそのバイト数分の配列を用意します。

配列は、"配列名 var 型[要素数]"で宣言します。
ここではバイト型の配列を4要素作っています。それぞれの配列には　配列名[要素番号] = 値で値を代入できます。配列は0番から作られますので、4を指定すると、num[0]、num[1]、num[2]、num[3]の4つが作られます。

⑤ UART受信割込が発生するとPIR1のビット5(RCIFビット)が1になります。これはソフトウエアで0にリセットする必要があります。通常はプログラム初期化時と割込先ルーチンでリセットします。

⑥ PIE1のビット5(RCIEビット)はUART1の受信割込を有効にするビットです。
設定値1で有効になります。

⑦ lcdout命令には引数としてコマンドが必要です。
コマンド一覧は下表の通りです。
"$fe, 1"はLCD画面をクリアする制御コマンドです。クリアーするとカーソル位置は1行目1文字目に移動します。

Command	Operation
$FE, 1	Clear display
$FE, 2	Return home (beginning of first line)
$FE, $0C	Cursor off
$FE, $0E	Underline cursor on
$FE, $0F	Blinking cursor on
$FE, $10	Move cursor left one position
$FE, $14	Move cursor right one position
$FE, $C0	Move cursor to beginning of second line
$FE, $94	Move cursor to beginning of third line
$FE, $D4	Move cursor to beginning of fourth line

⑧ 割込ルーチンを開始する位置にdisableを記述します。
これは「ここ以下は通常ルーチンを無効にする」という意味になります。
割込ルーチン終了時にはenableを記述して通常ルーチンを有効にします。

⑨ hserin命令はPICに搭載されているハードウエアUARTでデータを受信する命令です。
引数に指定した配列または変数に受信したデータを格納します。データは8ビット長なのでbyte型の変数もしくは配列を使用します。
str修正子は配列と組み合わせて使います。受信するデータのサイズ、受信を終了するデータを指定できます。例えば固定長のデータであればその長さを指定します。またそのデータの終端が分かっていればその終端データを指定することでそのデータを受信すれば指定したデータ長に達しなくても受信を終了します。「str 配列 \データサイズ \受信終了データ」 の書式で記述します。
なお終端データとして指定したデータは配列に格納されません。
この例では2バイトのデータを受信したら受信ルーチンを終了します。
またキャリッジリターン$0Dを受信したらすぐに受信を終了します。

hserin命令ではデータを受信するまでここでプログラムは待機状態となります。
str修正子を利用した場合指定したサイズもしくは終端データを受信するまでずっとプログラムは待機状態となります。もし一定期間データを受信できない場合にはタイムアウト時間を指定するオプションもあります。

⑩ LCDに受信したデータを表示します。
hex2修正子で変数に格納されている値を16進数2桁で表示します。表示するデータは配列の要素0なのでそれを指定します。

⑪ hserout命令はハードウエアUARTにてデータを出力する命令です。
配列を指定した場合、配列の内容すべてが出力されます。

⑫ UART受信割込のフラグPIR1レジスタのビット5はソフトウエアで0にクリアする必要があります。割込ルーチン最後で必ずクリアします。

⑬ 割込ルーチンから通常ルーチンへの復帰はresume命令を使用します。

⑭ 通常ルーチンを有効にするため最後にenableを記述します。

4-5、TMR0の割込を用いたプログラム

■学習の内容

PICに内蔵のタイマー、TMR0を利用して正確な1秒を作り出すタイマーを作成します。
TMR0はPICに搭載されている16ビットのカウンタでこれがアップカウンタとして動作します。フルカウントまで進むと次のパルス入力で0に戻りますが、この時に割込みが発生します。割込が発生するとTMR0IFビットが1にセットされます。

■プログラムの内容

LCDに4桁で秒数を表示します。電源を入れると1秒毎にカウントを開始します。
TMR0の利用方法、レジスタの設定方法などを学習してください。

■プログラムを書いてみよう

```
DEFINE OSC 32

On Interrupt Goto TMR0_OVF                  割込発生時の飛び先ラベルの設定

DEFINE LCD_DREG PORTB                       ┐
DEFINE LCD_DBIT 0                           │
DEFINE LCD_RSREG PORTB                      │ LCD の結線情報の定義
DEFINE LCD_RSBIT 4                          │
DEFINE LCD_EREG PORTB                       │
DEFINE LCD_EBIT 5                           ┘

init:
    second  var  word                       秒をカウントする変数
    update  var  bit                        秒アップデートのフラグ用変数
    second = 0                              変数の初期化
    update = 1                                  〃

    ANSELB = 0                              PORTB は全ビットデジタル IO
    T0CON = $86                             T0CON 変数に 134 を代入      ①
    TMR0H = $0B                             TMR0H 変数に 11 を代入
    TMR0L = $DC                             TMR0L 変数に 220 を代入
    INTCON = $A0                            割込を有効化、TMR0 割込を有効化  ②

    lcdout $fe, 1                           LCD を初期化

main:
    if update = 1 then                      もし秒がアップデートされた場合  ③
        lcdout $fe, 1, dec4 second            10 進数 4 桁で変数 second の内容を表示
        update = 0                            update フラグを 0 に
    endif
    goto main                               main に戻る

disable                                     通常ルーチンここまで

TMR0_OVF :                                  TMR0 割込発生でここに飛んでくる
    second = second + 1                     second 変数を 1 インクリメント  ④
    update = 1                              アップデートフラグを 1 にセット

    TMR0H = $0B                             TMR0H 変数に 11 を代入
```

```
    TMR0L = $DC                          TMR0L 変数に 220 を代入
    INTCON = $A0                         TMR0 割込フラグを 0 にクリア　⑤

    resume                               メインルーチンに戻る

enable                                   割込処理ルーチンを終了
```

■プログラムの解説

① TMR0で一定間隔の割込を発生されるためには割込タイミングの設定を各レジスタで設定する必要があります。TMR0は8ビットモードと16ビットモードが選択でき、T0CONレジスタのビット6で設定できます。このチュートリアルでは16ビットモードとして使います。

16ビットなので上位バイトと下位バイトをそれぞれTMR0HレジスタとTMR0Lレジスタに分けています。値を書き込む場合には必ずTMR0Hに値を書き込み、それからTMR0Lに書き込みます。
マイコンが32MHz動作時に1秒間隔で割込が発生するように数値を計算していきます。
PICではクロック周波数に対し実際に内部で動作しているクロックは1/4になります。よって32MHzクロックの時内部では8MHzでの動作となります。よって1サイクルは125ナノ秒です。
1秒ということは　1s / 125ns = 8000000 となり8000000回カウントすればよいことになります。しかしTMR0は16ビットモードで最大65536カウントなのでこのままではカウントできる上限を超えています。

そこでプリスケーラーを使います。プリスケーラーは周波数分周器といいます。
TMR0のプリスケーラー比はT0CONレジスタのビット0～2で設定できます。ここでは1:128にしました。これで1サイクルは125ns×128= 16マイクロ秒となります。
よって 1s / 16μs = 62500回カウントすれば1秒で割込が発生します。
62500回のカウントでオーバーフローさせるためには 65536 - 62500 = 3036　をあらかじめTMR0HとTMR0Lレジスタにセットしておけばよいことになります。

3036は16進数にすると$0BDCですので、上位のTMR0Hには$0Bを、下位のTMR0Lには$DCを代入しておけばよいことになります。

② 割り込み処理はINTCONレジスタを操作することで実現します。
INTCONレジスタの詳細はPICのデータシートをご参照ください。ここでは、TMR0の割込を有効に設定しその他の割込は無効に設定しています。また①で解説したとおりプリスケーラ分周比を1:128としています。
INTCONレジスタのビット2はTMR0オーバーフロー時、すなわち割込発生後に1になります。この値が1になることを検出して割込発生を処理しています。この値は、ソフトウエア側で割込ルーチン終了時に必ず0に初期化しなければなりません（⑤の位置）。

③ update変数を更新フラグとして使用しています。
LCDの表示を常に更新しているとLCD表示がちらつきます。よって1秒ごとsecond変数がインクリメントされた時だけLCDの表示を更新するようにしています。

④ second変数の値を1つインクリメントしています。
インクリメントとは増加させることをいいます。

※PAUSE命令は使用できないの？
1秒を作るならば、PAUSE命令を使用して"PAUSE 1000"と記述した方が簡単にタイマーが作れそうです。しかしPAUSE命令はTMR0を使用しているわけではなく、アセンブラ言語レベルで小休止の命令をたくさん組み合わせて擬似的に時間を作り出しています。よって、PAUSE命令が作る遅延時間は精度の高い時間ではないのです。PAUSE命令で時計を作るとすぐに狂ってしまいます。

4-6、ADコンバーターを使用したプログラム

■学習の内容

PICには逐次変換型でサンプルホールド機能を搭載した分解能10ビットのADコンバーターが搭載されています。ADコンバーターの使用方法と変数の内容をLCDに表示する方法を学習します。

■プログラムの内容

RA3(AN3)に印加された電圧をADコンバーターで取得してLCDに0000～1024までの数値を表示するプログラムを作成します。

■プログラムを書いてみよう

```
DEFINE OSC 32

Define  LCD_DREG       PORTB
Define  LCD_DBIT       0
Define  LCD_RSREG      PORTB
Define  LCD_RSBIT      4
Define  LCD_EREG       PORTB
Define  LCD_EBIT       5

DEFINE  ADC_BITS       10              ADC の分解能を 10 ビットに設定　①
DEFINE  ADC_CLOCK      3               ADC のｻﾝﾌﾟﾘﾝｸﾞｸﾛｯｸを内蔵 RC 使用に設定　②
DEFINE  ADC_SAMPLE     50              ｻﾝﾌﾟﾘﾝｸﾞ時間を 50 マイクロ秒に設定　③

ADResult var word                      ADC の結果を格納する word 型変数
NowData  var word                      データー時退避用変数
adresult = 0                           変数を初期化
NowData = 0                                 〃

init:
    ANSELA.3 = 1 : ANSELB = 0          RA3 だけアナログ入力ピンに設定　④
    TRISA.3 = 1 : TRISB = 0            RA3 だけ入力ピンに設定

    ADCON0 = %00001101                 ADC の入力ピンを AN3 に設定　⑤
    ADCON1 = 0                         Vref+を Vdd に、Vref-を Vss に設定　⑥
    ADCON2 = %10000000                 ADC の結果を右詰めで格納　⑦

main:
    if NowData != ADResult then        ADC の値が更新されたかを確認　⑧
        lcdout $fe, 1, dec4 ADResult     ADResult の値を 10 進数 4 桁で LCD に表示　⑨
        NowData = ADResult               現在の ADC の値を待避用変数に代入
        pause 300                        LCD 表示安定用のディレイ
    endif

    adcin 3, ADResult                  AN3(RA3)の印加電圧を ADC で取得 ⑩

    goto main
```

■プログラムの解説

① ADコンバーターの命令adcinには3つのDEFINEがあり基本動作を設定します。
DEFINE ADC_BITSによりADコンバーターの分解能を指定します。
分解能10ビットですので10を指定します。

② ADコンバーターのサンプリング周波数の時間を生成する発振源を選択します。
内蔵RC発振子の場合には3を指定します。

③ サンプリング時間をマイクロ秒単位で指定します。ここで設定した時間が取得時間となります。
50マイクロ秒と設定しました。

④ ANSELAレジスタのビット3を1に設定しRA3をアナログ入力に設定します。
なお同時にTRISAレジスタの設定も忘れないようにする必要があります。

⑤ ADCON0レジスタではどのアナログチャンネルのデータを取得するか設定します。
ここではAN3を選択しています。

⑥ ADCON1レジスタではVref（基準電圧）の値をどのように設定するのかを指定します。
ここでは、Vref-はVssに、Vref+はVddにそれぞれ設定しています。
よって、ADコンバーターのスケールはVss～Vddまでを10ビットで計測します。

⑦ ADコンバーターの結果は10ビットで返されますがPIC18F45K22の結果データ格納用レジスタの幅は8ビットです。そのため結果は2つのレジスタに分けて格納されます。
このとき上位2ビットを別のレジスタにするのか、下位2ビットを別のレジスタにするをADCON2レジスタのビット7で設定します。PBP3は必ず上位2ビットを上位レジスタに格納する指定"右詰指定"としなければなりません。ここでは右詰指定にしています。

⑤ ADCIN命令を使用する場合には、必ず該当ポートの入出力方向設定用のレジスタTRISxレジスタの設定をしなくてはなりません。本チュートリアルでは、RA3だけが入力でいいので%00001000と設定しています。

⑧ AD変換の結果が新しい値に更新された時だけLCD表示を更新させるために工夫が必要です。
常にLCDの表示を更新してしまうとLCDが高速に明滅して文字が読みにくくなってしまいます。
取得した結果を以前の値と比較し、値が異なっている時のみLCDの表示を更新させます。

!=　演算子は、「～ではない」という否定を表す比較演算子です。2つの変数の値が不一致の場合にif文が実行されLCD表示を更新します。

⑨ lcdout命令を使用してLCDにADCで取得した値を表示させます。
LCDに変数の値を文字列として表示させる場合には、プリフィックスと呼ばれる修正子を使用します。ここではdec修正子を使用して変数の値を10進数で表示させるように指定しています。
さらに dec4 と修正子の後ろに桁数指定を記述することで、指定した桁数の重みを付けられます。dec4 としているので仮に変数ADResultの値が0であってもLCDには　0000　と4桁で表示されます。
なお、lcdout命令では複数の文字列や変数を ,（カンマ）で区切ることで複数の引数を連結させて表示させることができます。例えば、LCDに　"RESULT = xxxx"（xxxxはAD値）と表示させたい場合には下記のように記述します。

```
LCDOUT $fe、 1、 "RESULT = "、 DEC4 ADResult
```

⑩ adcin命令は指定したチャンネルの電圧値を変数に取り込みます。
「adcin チャンネル, 変数」の書式で記述します。
返されるADCの結果は10ビットなので変数にはword型変数を使います。

4-7、32ビット変数long型を使用するプログラム

■学習の内容

PIC18Fシリーズでは32ビット幅のlong型変数が使えます。
ここでは簡単にlong型変数を使用した演算を行ってその結果をLCDに表示させます。

■プログラムの内容

ADコンバータのAN0(RA0)とAN12(RB0)の2チャンネル使用し、両方の値を取得してその値をLCDの1行目に表示します。その2値を乗算し乗算結果をLCDの2行目に表示させます。

■プログラムを書いてみよう

```
DEFINE OSC 32

Define  LCD_DREG       PORTB
Define  LCD_DBIT       0
Define  LCD_RSREG      PORTB
Define  LCD_RSBIT      4
Define  LCD_EREG       PORTB
Define  LCD_EBIT       5

DEFINE  ADC_BITS       10
DEFINE  ADC_CLOCK      3
DEFINE  ADC_SAMPLE     50

AN0RES  var word                          AN0 の ADC の値を入れる変数
AN12RES var word                          AN12 の ADC の値を入れる変数
Total   var long                          乗算結果を入れる long 型変数　①

init:
    ANSELA = 1 : ANSELB = 1               RA0 と RB0 をそれぞれアナログ入力
    TRISA = 1 : TRISB = 1                 RA0 と RB0 を入力ピンに設定
    ADCON1 = 0
    ADCON2 = %10000000

    LCDOUT $fe, 1

main:
    ADCON0 = %00000001                    ADC の入力チャンネルを AN0 に設定
    adcin 0, AN0RES                       AN0 の値を取得して変数に代入

    ADCON0 = %00110001                    ADC の入力チャンネルを AN12 に設定
    adcin 12, AN12RES                     AN12 の値を取得して変数に代入

    lcdout $fe,1
    lcdout dec4 AN0RES, "*", dec4 AN12RES AN0 と AN12 の値をそれぞれ表示　②

    Total = AN0RES * AN12RES              AN0 と AN12 の値を乗算して long 型
    lcdout $fe,$c0, dec Total             変数に代入して LCD に表示
```

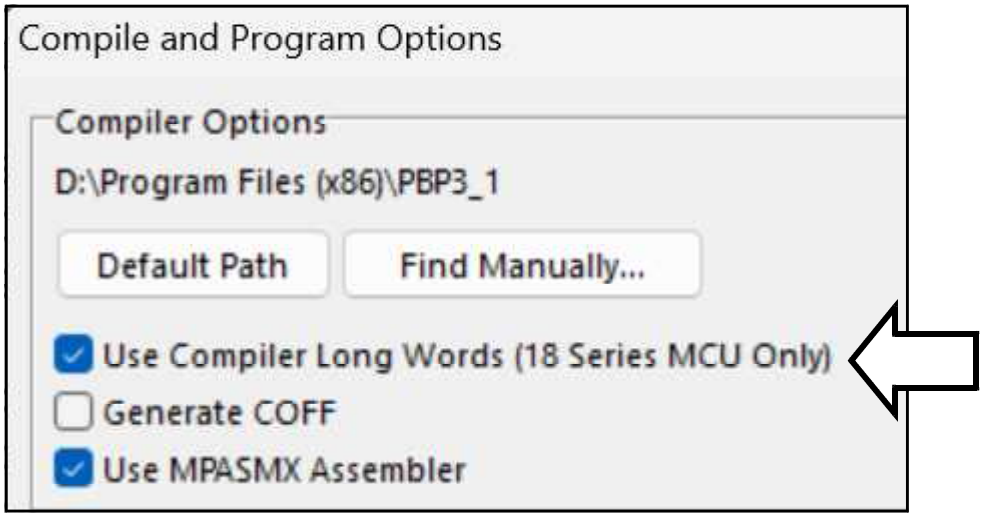
```
pause 100                                   ちらつき防止用のディレイ

goto main
```

■PIC18Fシリーズでlong型変数が使用できるように設定しましょう

PIC18Fシリーズに限り、32ビット長の値を格納できるlong型変数が使えます。
long型変数を使用したプログラムをコンパイルするにはPBPLという拡張されたPBPを使用してコンパイルする必要があります。コンパイルを実行する前にPBPLを使用できるように設定します。

MCSのメニューバーの"View"→"Compile and Program Options"をクリックします。

ダイアログが表示されますので"Use Compiler Long Words"にチェックを入れてOKボタンを押します。



これでlong型変数を使用したプログラムをコンパイルできるようになりました。
コンパイルなどの操作方法はこれまでと同様です。

■プログラムの解説

① long型はPIC18Fシリーズを使用した場合に限り使用できる32ビットサイズの型です。
0～2147483647の値を扱うことができます。
long型変数を使用したプログラムをコンパイルする場合にはPBPLを使用する必要があります。

PBPでは型変換は暗黙のうちに自動的に行われます。規則としては大きいサイズの型に合わせられます。ここでは、ワード型×ワード型というワード型同士の乗算ですが、結果がlong型になるため、結果はlong型変数に代入しています。
このような型の異なる変数同士の演算でもPBPが自動的に型変換をします。

② lcdout命令では複数の引数を指定することで複数の文字列を連結して表示させられます。
ここでは10進数4桁の変数の値と、文字 * を表示しています。

PIC18シリーズ以外のPIC使用時はlong変数使用のチェックを外してください！

long変数が利用できるのはPIC18シリーズだけです。PIC18シリーズ以外のデバイスを使う場合には必ずCompile and Program Optionsウインドウ内の"Use Compiler Long Words"のチェックを外してください。

4-8、PIC12F1822のプログラム

■学習の内容

8ピンのデバイスPIC12F1822のプログラムを作ります。
内蔵発振子を利用したプログラムの作成方法を解説します。

■プログラムの内容

PIC12F1822を内蔵発振子で8MHzにて動作させます。
ADコンバータのAN0に印加された電圧値を読み取りその電圧値に応じてRA4の点滅速度を変更するプログラムを作ります。

■プログラムを書いてみよう

```
define OSC 8

DEFINE ADC_BITS 10              ADC の分解能を 10 ビットとして定義
DEFINE ADC_CLOCK 3              ADC のクロック源は内蔵 RC として設定

OSCCON = %01110010              内蔵発振子 8MHz に設定　①
ANSELA = 1                      RA0 をアナログ電圧の入力として設定
TRISA = 1                       RA0 を入力ピンとして設定

adc  var  word                  ADC の値格納用変数
i  var  word                    ループ変数

main:
    adcin 0, adc                AN0 の ADC の値を変数 adc に代入

    LATA.4 = 1                  LATA.4 を 1 に設定
    gosub time                  サブルーチンで time ルーチンにジャンプ　②
    LATA.4 = 0
    gosub time

    goto main                   繰り返し

time:                           time サブルーチン
    for i=0 to adc              for 文で変数 i が 0 から adc の値までループ　③
        @nop                    アセンブラの nop 命令を挿入　④
    next i                      for 文の終わり

    resume                      サブルーチンを終了して飛び元に戻る　⑤
```

■プログラムの解説

① 内蔵発振子の周波数設定はOSCCONレジスタで行います。
ここでは8MHzに設定しています。

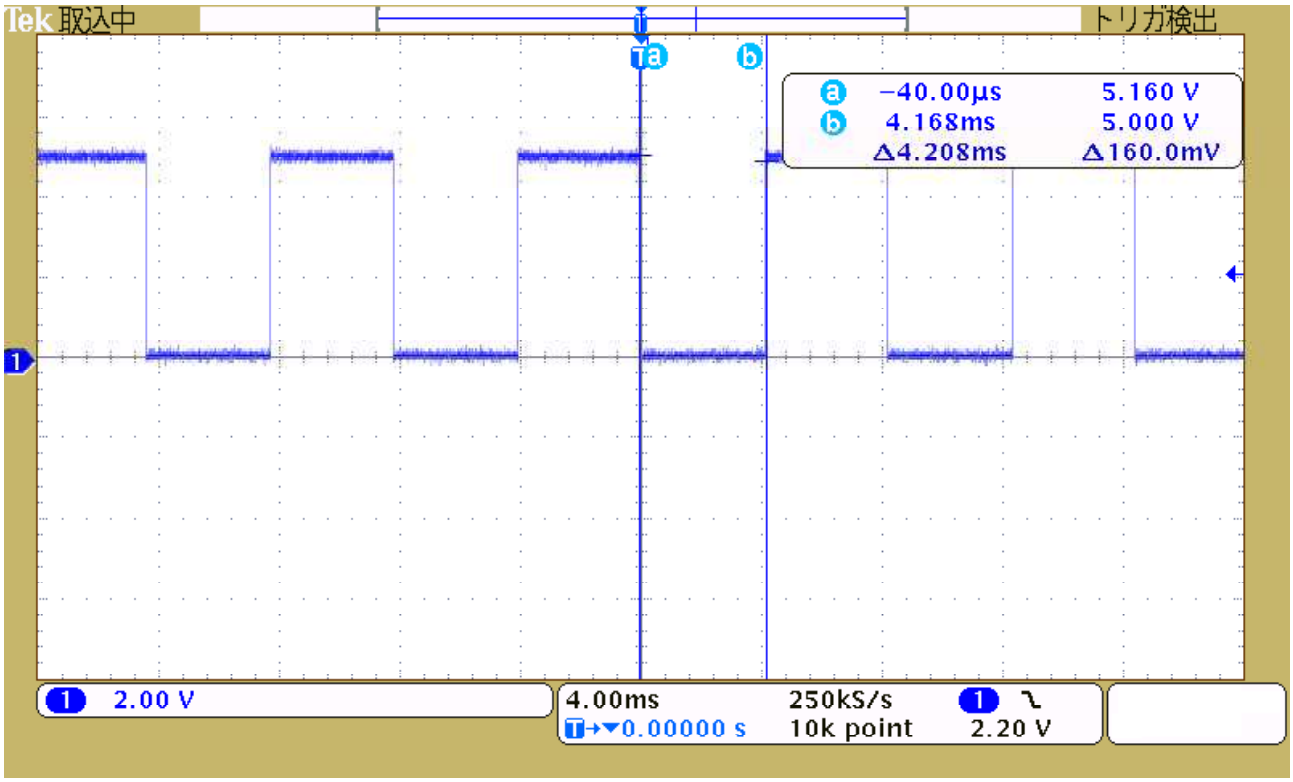
② gosub命令はサブルーチンへ飛ぶ命令です。サブルーチンはgoto文と異なりサブルーチン先でresume命令があると飛び元へ処理が返ります。

③ for文でループを作りディレイする時間を変更できるようにしています。
変数iは0からスタートし、変数adcの値までインクリメント(増加)してループを終了します。
adcの値が大きくなればループ回数が増えるためディレイ時間が増加できるという構造になっています。

④ @はインラインアセンブラを挿入するための命令です。
nopはアセンブラ言語でNo Operationすなわち「何もしない」命令です。このnop命令は命令実行サイクルの1サイクル何もしないという命令ですので、発振周波数によってその時間が変わります。

⑤ resume命令はサブルーチン先で実行することでサブルーチンを終了し処理を飛び先に戻します。

このプログラムを実行した時の点滅サイクルのオシロ画像を下図に示します。
adcの値を0とした時、下図のように4.2msサイクルとなります。



コンフィギュレーションビットの設定にてoscillatorの種類は必ずINTOSC(内蔵発振子)の設定にしてください。

［コラム］デバッグの基本とコードの移植

プログラムを作って実行してみると思ったように動作しないことや、不可思議な動きをして困惑するケースが出てきます。
また「ネットからサンプルプログラムをそのままコピーしてみたけど動かない」、「PIC16F○○用に作ったプログラムをPIC18F△△に移植したら動かない」という事例もよく見かけます。共通しているのは全体をゴソッとコピーしてきたり、移植したりするケースです。細かな部分をよくチェックできていないことや、PICの場合デバイスによって同じ設定内容でもレジスタの名前が違っていたりすることがあり、デバイスの型式が変わっているのに、レジスタ名を修正していないことなどけが原因になります。
まずプログラムが思うように動作しない・・という場合には次のことを試してください。

[1]プログラムを最小限短くする

まず最初にプログラムを「○○が動作しない」という部分だけに削り、その部分だけをデバッグしてみます。そこが動作しない結果、△△の部分にも影響を与えている場合があります。基本はまずプログラムは最小限の小さくしながら、各機能を試しながらそれを組み合わせる感じで長くしていくことを基本としてください。

多くの問題はほとんどこれで解決します。例えば「IOピンが思ったように動作しない・・・」という場合、当然その「IOピンを使って処理を分岐するように書いたプログラム」であれば、その分岐処理も期待通りには動作しないでしょう。このような場合、まずはIOピンだけのプログラムを書いてみます。例えばPORTAの入力値をそのままPORTBに出力するプログラムを書いてみたりします。・・・

```
TRISA = $FF        'PORTAは全ビット入力ピン
TRISB = $0         'PORTBは全ビット出力ピン
PORTB = PORTA      'PORTAの値をPORTBに代入
```

これで動作するでしょうか。もし動作しなければ原因を探すことになります。
一番多いのが次の[2]でも記載していますが、ポートがデジタルIOになっていないことです。PICは電源投入時やリセット後、ピンがアナログ入力モードになっていることが多くデジタルIOとして使う場合にはレジスタによる設定が必要です。PIC16F1939を使っていたのならば・・・

```
ANSELA = 0
ANSELB = 0
```

この2つをプログラムの先頭部分記述しないといけません。これで再度実行すればプログラムは動くようになったはずです。もしこれでも動かなければコンフィギュレーションビットの設定が誤っているのかもしれません。

最小限簡単にしたプログラムが動作したら、次のステップに進みましょう。少し機能を追加して思い通りに動作するかどうか確認します。そこで詰まったら再度その部分だけを切り取ってデバッグしたり、データシートを読み返してみましょう。

長くなったプログラムのデバッグはとにかく大変です。疑われる場所が複数あると、どこが悪くてどこが正しく動いているのか分からなくなり、行き詰まってしまいます。少しずつプログラムを確認し、動作確認ができたら次のプログラムを継ぎ足して・・・とすこしずつ大きくしていくように作ればデバッグが楽になります。

[2]サンプルプログラムを全コピーして使ったり、別のPIC用に作られたプログラムを移植しない

よくあるのがこの問題です。サンプルプログラムはプログラミングの際とても便利ですが、それをそのまま使ったり、そっくりコピーして思ったように動作させせよう・・というのは難しい話です。サンプルプログラムはあくまでも一例でありその内容がすべてのケースで使えるわけではありません。また、別のPIC用のプログラムを移植させている場合にもトラブルが多く発生します。

これは先に書いたとおりこれらは「プログラムを徐々に長くながらデバッグ」したわけではないからです。少しずつ、1つずつ動作を確認しながらプログラムを長くしてきた場合、どのあたりに問題があるのか検討が付きやすいです。一方長いプログラムのそのままコピーしたり別のPIC用に移植すると、いきなり長いプログラムを実行することになり、もしそのプログラムが思ったように動作しないと、どこに問題があるのか切り分けがすごく難しくなるということです。

「PIC○○で動いたプログラムをPIC△△用にコンパイルして書き込んだけど動かない！」ということがあります。PICはデバイス毎にレジスタの設定内容が違ったり名前が異なったりしています。あるデバイスで動作したプログラムをそのままコピーして別のPIC用にコンパイルしてうまく動作するケースはごくまれです。PICが変わったのであれば、できれば1から作り直すことをお薦めします。

例えばTMR0を使ったプログラムの場合、PIC16F1939で作ったプログラムがあったとします。TMR0のプリスケーラ分周比はPIC16F1939ではOPTION_REGレジスタのビット0～2で設定します。このプログラムを同じピン数のPIC18F45K22に移植したとします。しかしPIC18F45K22ではプリスケーラの分周比設定はT0CONレジスタのビット0～2で設定します。そのためそのまま移植してもプリスケーラの分周比を設定するレジスタが違うので正しく設定の記述を変更しなければ動作しないのは明白です。1からプログラムを作るとこのあたりは1つ1つ確認しながら作り込むため違いに気づきますが、そうでないと見落としてしまいがちです。

[3]データシートを確認する

データシートは必ずご確認ください。特に重要なのはPICの場合電源投入時多くのデバイスで、ピンがアナログ入力モードになっている点です。
アナログ入力とはADコンバータの入力ピンです。この設定になっているピンはデジタルIOとしてプログラムから出力をしたりピンの状態をプログラムで検出することはできません。プログラムの先頭部分で必ずデジタルIOピンとしての設定をレジスタで行う必要があります。これをしない場合プログラムが全く思い通りに動かないということになります。
設定はある程度デバイスファミリー間では同じですが、異なるファミリーのデバイスでは設定するレジスタが異なります。例えば、PIC16F1939などエンハンスドPIC16シリーズ、PIC18F(K, J)などの場合には、ANSELAやANSELBといったレジスタ、PIC16F887などややレガシーなデバイスではANSELH、ANSELレジスタで設定します。さらに古いタイプのPIC16F877AなどではADCON1レジスタで行います。

ご使用のデバイスがどのレジスタを設定すればデジタルIOにできるか、などはそのデバイスのデータシートを見る以外方法はありません。
「うまく動作しない！」という場合には必ずデータシートをお読みください。

[4]コンフィギュレーションビットの設定を確認する

PICを使う上でコンフィギュレーションビットの設定は欠かせません。適当な設定値で使うとプログラムは正しくても全く動作しなかったり、勝手に動作周波数が4逓倍されていたり、Watchdog Timerでリセットがかかってタイマの動作がおかしくなったり・・・と原因を複雑にする様々な問題を引き起こします。
コンフィギュレーションビットの設定もデバイスにより「まちまち」です。ある程度の共通した項目はありますが、デバイスによって設定内容はことなるため、データシートの参照が不可欠です。
またインターネットなどでも情報を手に入れられますので、使っているデバイスの型式で検索をしてみると解決策があるかもしれません。

5. PICBASIC PRO Compilerの変数と配列

5-1、変数の種類

変数とはプログラム内で一時的に数値を格納しておく箱のようなものです。
実体はPICの中にあるRAMの領域を確保し、そこに値を一時的に記憶させます。
変数を宣言するとPBPはRAMに変数用のメモリー領域を確保します。

変数は使用する前に必ず宣言が必要です。PBPでは下記の書式で変数を宣言します。

変数名 var 型

型は4種類から選択できます。long型はPIC18シリーズでしか使用できません。
PIC10,12,16シリーズはbit型、byte型、word型の3つから選択します。

型名	ビット長	最大値（10進）
BIT	1	0～1
BYTE	8	0～255
WORD	16	0～65535
LONG	32	-2147483647～2147483647

PICマイコンは内部のRAMの容量が少ないですので適切な型を選ぶ必要があります。
PIC18シリーズでlong型変数を使う場合にはPBPLを使う必要がありMicroCode Studioで設定が必要です。詳細は44ページ又は50ページを参照してください。

5-2、変数や配列のメモリー配置について

PBPでは何も指定しないで変数を宣言するとPBPが自動的にRAMに変数のためのスペースを割り当てます。通常配置する場所はユーザーが意識しなくてもいいですが明示的に配置する場所を指定したい場所には変数宣言時に次のキーワードを使用します。

address　　RAMのアドレスを直接指定します。必ずデータシートで配置可能なアドレス値を確認しておく必要があります。

BANKx　　PIC16Fシリーズでのみ有効です。指定したバンクに配置します。
PIC18Fシリーズにはバンクの概念がありませんので使用できません。

```
ticker VAR BYTE BANK0      'ticker変数はバンク0にアサインされます。
wsave VAR BYTE $70         'wsave変数はRAMアドレス0x70につくられます。
```

※通常変数、配列を宣言する場合明示的にアドレスやバンクを指定する必要はありません。しかし、次のような場合には明示的に指定する必要があります。

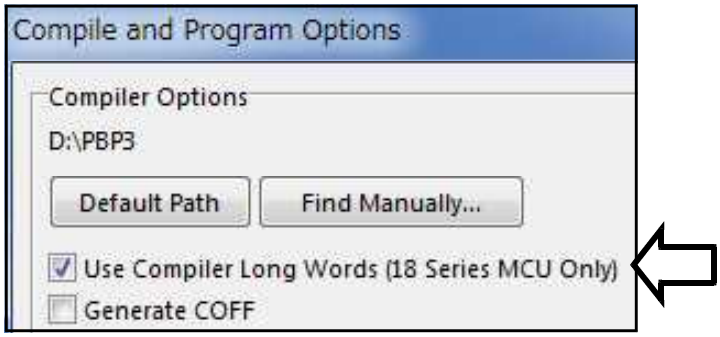
・USB通信機能付きPICの場合、バッファ用配列を宣言する時はUSBバッファのアドレスを指定する必要があります。

・PIC18F以外のPICを使用する場合で配列を使う場合、ページ(BANK)をまたがってRAMが割り当てられる場合があります。その場合例えば連続でアドレス指定してRAMから値を取り出そうとするとバンクの切り替えが必要になります。バンクの切り替えは手順が煩雑になるためそういう場合には明示的にバンクやアドレスを指定することで切り替えをなくすることができます。

5-3、32ビット長変数の使い方

32ビットの値を格納できるlONG型は、PIC18シリーズのプログラムにのみ使用できる型ですが、この型を使用したプログラムをコンパイルする場合には、PBPLという拡張されたPBPを使用する必要があります。
long型を使用する場合はMCSにて下記の手順で設定してください。

MCSのメニューバーの"View"→"Compile and Program Options"をクリックします。
ダイアログが表示されますので、"Use Compiler Long Words"にチェックを入れて、OKボタンを押します。



また、LONG型では符号付き演算が可能です。(LONG型以外の変数は符号なし演算のみです。)

5-4、変数に値を代入する

変数に値を代入する場合には他のプログラミング言語同様、代入演算子(=)の右辺から左辺へ値が代入されます。

variable = 350

数値に何も記号をつけないと、PBPは10進数として処理します。2進数、16進数を表す場合には下記のように数値の前にプレフィックスを付けます。

進数	プレフィックス	表記例
2進数	%	%11111111
10進数		255
16進数	$	$FF

文字を"(ダブルクォーテーション)で囲んで変数に代入するとASCIIコードの値が代入されます。

variable = "A"

variable変数には65が入ります。なお文字を代入する場合にはASCIIコードとなり8ビット長ですからbyte型変数を使用します。

PBPでは文字列型はありません。文字はASCIIコードとして8ビット長のbyte型変数を使います。
文字列の場合にはbyte配列を使います。

5-5、変数同士の算術演算

変数同士で算術演算ができます。型が異なる変数同士の場合、より大きな型に暗黙のうちに型変換が行われます。

```
a  var  byte
b  var  byte
c  var  word

a  = 200
b  = 150

c  = (a + b) * 5
```

上記の例では変数cには1750が演算結果として格納されます。結果を代入する変数の型が小さく、型の最大値を超えてしまった場合には、演算結果は信頼できないものとなります。

PBPの演算は算数同様（　）内が優先され次に乗除が優先されて計算されます。

5-6、変数の各ビットを参照する

変数はBIT型からLONG型まで宣言して使用することができますが、PBPでは変数の各ビットの値を参照したり、各ビットに値（0か1）を代入することができます。

```
variable  var  byte

variable.0  =  1
variable.1  =  0
variable.2  =  0
variable.3  =  1
variable.4  =  0
variable.5  =  0
variable.6  =  1
variable.7  =  0
```

変数名の後ろに.（ピリオド）を付けてビット数を記述することで変数の任意のビットを指定できます。
上記の場合変数variableの値は、 01001001 となります。
また同様の方法で任意のビットの値を取得することも可能です。

5-7、変数のオーバーフローを検出する

PBPでは宣言した変数がその最大値を超えオーバーフローすると変数の値は0に戻ります。
例えばWord型で宣言した変数の場合10進数で"65535"の次は"0"になります。またPBPでは負の値を扱えませんので0から-1を加算する（0から1を減算する）と型の最大値になります。
例えば、Word型変数の場合
"1　→　0　→　65535　→　・・・・　"　となります。

変数がオーバーフローしたかどうかを判定するには、下記にあるようなプログラムを簡易的に組んで判定する方法があります。PBPでは、宣言した変数を特殊処理していますので、STATUSレジスタ等でオーバーフローを判定しようとしても正しく行えません。下記のようなオーバーフロールーチンを記述する必要があります。

※この方法は、すべての演算結果におけるオーバーフローを検出できるものではありません。あくまでも簡易的な方法として利用する方法ですので使用に際しては十分にプログラムの検証を行ってください。

```
X var WORD   : Y var WORD  :  Z var WORD

Z = X + Y            '例: X=65530, Y=100  などで z がオーバーフローした場合

IF Z < X OR Z < Y THEN  Overflow        'z のオーバーフローを判定

'z がオーバーフローしていると overflow ラベルにジャンプします
```

5-8、変数のスケーラー

PBPに搭載のBIT型、BYTE型、WORD型、LONG型は、それぞれ次の表のように細かくスケーリングされています。これを変数スケーラと呼びます。
例えば、次のような使い方ができます。

```
long_val    var    LONG
long_val.BYTE2   =   255
```

Variable Modifiers				
LONG	WORD1	BYTE3	BIT31	most-significant-bit (MSB)
			BIT30	
			BIT29	
			BIT28	
			BIT27	
			BIT26	
			BIT25	
			BIT24	
		BYTE2	BIT23	
			BIT22	
			BIT21	
			BIT20	
			BIT19	
			BIT18	
			BIT17	
			BIT16	
	WORD0	BYTE1	BIT15	
			BIT14	
			BIT13	
			BIT12	
			BIT11	
			BIT10	
			BIT9	
			BIT8	
		BYTE0	BIT7	
			BIT6	
			BIT5	
			BIT4	
			BIT3	
			BIT2	
			BIT1	
			BIT0	least-significant-bit (LSB)

上記のようにした場合、上の表から分かるとおり、Long型変数long_valのBit16～Bit23すべてに値、1が入ります。（255は8ビットで%11111111です）

よって、変数long_valの値を参照すると次のようになります。

2進数の場合：　00000000111111110000000000000000
10進数の場合：16711680

1つの変数がそれぞれスケーリングできますので例えば次のような応用もできます。

```
word_val var WORD

word_val.BYTE1 = PORTC
word_val.BYTE2 = PORTB
```

Word型変数は16ビット長なので、8ビット長のI/OポートであるPORTCとPORTBをそれぞれ1つの変数にまとめてスケーリングすることができます。word_val変数に%FF00(%1111111100000000)を代入すれば、PORTCがすべてHで、PORTBがすべてLになります。
PBPでは下記のような使い方はすべて正しい使い方です。

```
long_val var LONG
word_val var WORD
byte_val var BYTE

long_val.WORD1 = 0
long_val.BYTE3 = 255
long_val.31 = 1

word_val.BYTE1 = 0
word_val.15 = 0

byte_val.7 = 0
```

5-9、配列の使用

配列は連続したメモリー領域を確保して値を一時的に記憶する仕組みです。
変数は宣言するごとにPBPが適当にメモリー領域を割り当てるため変数同士の関係性、連続性はありません。配列を使用すると、PBPはメモリー領域に連続した空間を確保するため、連続性のあるデータを格納する時などに使用します。配列は下記の書式で宣言します。

配列名　Var　型[要素数]

型は4種類から選択できますが、LONG型はPIC18シリーズの時だけ使用できます。

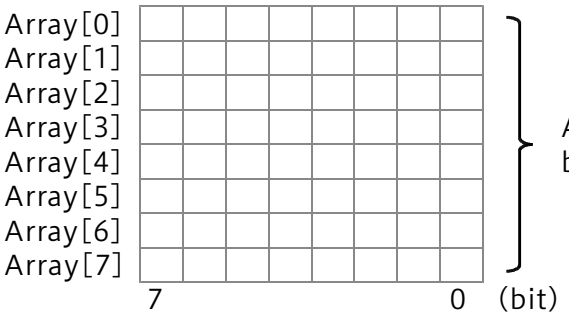
型	最大要素数(PIC18以外)
BIT	256
BYTE	96
WORD	48
LONG	制限なし

PIC18シリーズを使用する場合で、byte型、word型、long型の配列の場合のみ要素数の最大値は上記の値ではなくメモリーの空きがある限り使用できます。例えば、array var BYTE[2000]というような記述も可能です。コンパイル時にエラーが表示されなければ配列が正しく作成されています。PIC18シリーズ以外の場合には上表の最大要素数の制限をうけます。

エラーが表示された場合にはメモリーサイズが足りないですので要素数を適切に見直します。
メモリーはユーザーが宣言した以外にも各命令などが使用しますので、出来る限り小さい最適なサイズを指定するようにします。不必要に大きなサイズの配列は宣言しないようにします。

long型配列はPIC18Fシリーズ用のプログラムでPBPLを使用する場合にのみ利用できます。

たとえば、 **Array var byte[8]** と宣言すると、 Array という配列変数を作成します。
この場合配列は0～7までで確保されますから8個の配列要素が作成できます。配列は0番から用意されます。1番からではありませんので注意してください。記のような配列が作られるとイメージできます。



Array[0]～[7]の各配列が用意されます。
byte型配列ですので各要素も8ビット長で用意されます。

なおそれぞれの配列に値を個別に格納したり特定の配列の値を参照することが可能です。
例えば、次のような使い方ができます。

```
ANS  var  byte[3]        '変数 ANS を配列で宣言

ANS[0] = "A"             'ANS の配列 0 番目に,文字 A を格納
ANS[1] = "B"             'ANS の配列 1 番名に,文字 B を格納
ANS[2] = "C"             'ANS の配列 2 番目に,文字 C を格納

LCDOUT STR ANS¥3         'LCD に ANS の配列 3 つの内容を表示

IF ANS[0]="A"  AND  ANS[1]="B"  AND  ANS[2]="C" THEN
       LCDOUT $fe, 1, "ANS = "ABC" "
ENDIF
```

宣言した配列は各要素にそれぞれ下記の書式で値を代入したり、値を参照することができます。

・配列に値を代入する場合　　配列名[要素数] = 値
・配列の値を別の変数に代入する場合　　変数 = 配列名［要素数］

上のプログラム例ではIF文にてAND演算子を用いて、各配列の内容を参照してすべての条件が真の時のみIF文内を実行するようになっています。

配列をよく使用するのはデータの連続性が要求される場面です。例えば、シリアル通信で次々と受信したデータは連続したデータですので配列に格納していきます。
その他文字列も連続したデータですので配列に格納します。

ARRAYREAD命令、ARRAYWRITE命令を使うことで複数の値を変数に代入できます。詳しくは各命令の項目をご覧ください。

5-10、定数の使用

PBPでは文字列に定数を割り当てることができます。定数は変数と異なり一度宣言すると代入演算子で値を代入することはできません。

定数名 CON　値

割り当てられる値は、変数と同様で正の整数値のみです。
例：　APRIL CON 4

上記の例では文字列APRILは以降、値の4を示すことになります。

5-11、文字列にレジスタやレジスタの各ビットを割り当てる（アサイン）

PBPでは、文字列にPICマイコンのレジスタや、レジスタ内の任意のビットを割り当てることができます。書式は下記の通りです。

文字列　VAR　レジスタ名．ビット

例えば、RB0（PORTBのビット0）にLEDが接続されており、このピンをプログラム中でLEDと表記したい場合には次のように記述します。

LED　var　PORTB.0

プログラム中ではLEDと記述するとRB0を示すようになります。
同様にして任意のレジスタの特定のビットを文字列に割り当てて、プログラム内で参照できます。
例えば"PIR1レジスタ"のビット5を文字列"RCIF"にアサインする場合には、次のように記述します。

RCIF　var　PIE.5

任意のレジスタのビットを文字列にアサインできるため、各ビットの名前などを文字列としてアサインすればレジスタの処理が簡単になります。

変数や配列の宣言しすぎに注意！

変数や配列はいくらでも用意できるわけではありません。PICにはRAMの要領があり少ないRAMの中にPBPは適切に変数や配列の領域を確保しています。よって必要以上に変数や配列を宣言するとRAMに収まりきらずにエラーになります。

もう1つ注意が必要なのがPIC12F/16F系のデバイスを使う場合です。これらのデバイスはメモリーが内部で物理的に複数のページに別れています。そのため配列や変数を多用すると複数のページにまたがってメモリー領域を確保する必要がでてきます。すると本来連続したアドレスに配置されなければならない配列の連続性が失われ予期しない動作をしてしまうことがあります。このようにページをまたいでメモリーエリアが確保された場合PBPはコンパイル時に次のような警告メッセージを表示します。

Message[306]c\pbp\pbppic14.lib xxx : Crossing page boundary・・・

この警告が出た場合には配列等が複数のページをまたいだ領域に確保されていますので、動作をよく確認してください。動作に問題がなければこの警告が出てもそのプログラムは使えます。
上記の警告が出た状態で、より確実なプログラムを作りたい場合にはPIC18シリーズへの乗り換えをご検討ください。

6. PICBASIC PRO Compilerの演算機能

6-1、各進法の表現

PBPでは10進数、2進数、16進数の3種類の進数形式を使えます。
どの表現なのかを明確にするため数値の前にプレフィックスをつけて表します。

進数	プレフィックス	表記例
2進数	%	%00010001
10進数		1234
16進数	$	$7F

文字を"(ダブルクォーテーション)で囲んで変数に代入するとASCIIコードの値が代入されます。

6-2、算術演算子と論理演算子

■算術演算

演算子	詳細
+	加算
-	減算
*	乗算
/	除算
//	あまりを返す除算
<<	左シフト
>>	右シフト
ABS	絶対値
ATN	逆正接
COS	余弦
DCD	指定ビットを1にする　(例 VAL = DCD 2 と記述するとVALの値は%00000100)
DIG	10進数の指定桁の値を取り出す
DIV32	31ビット×15ビットの割り算
HYP	(直角三角形の)斜辺
MAX	大きい方の値を取り出す (例: VAL=A MAX 10 の場合Aが10より大きな値の時、Aの値がVALに代入、それ以外は10が代入される)
MIN	小さい方の値を取り出す　(MAXと同様)
NCD	最上位の1のビットの値を取り出す (例： VAL = NCD %01001000 の場合VALには7が代入される)
REV	指定したビットより下位のビットの値を反転
SIN	正弦
SQR	平方根

左シフト(<<)の例

```
cont   CON      %11101100          '定数contを指定、値は%11101100
val = cont << 3                    'contの値を左に3ビットシフトしval変数に値を代入
```

上記のプログラムでは、valには、%11101100000　が入ります。

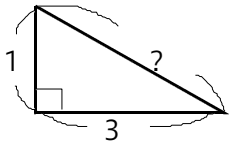
右シフト(>>)の例

```
cont   CON     %11101100          '定数contを指定、値は%11101100
val    =       cont >> 3           'contの値を右に3ビットシフトしval変数に値を代入
```

上記のプログラムでは、valには、%11101　が入ります。

三角形の斜辺(HYP)の例

HYP演算子は、直角三角形の斜辺を求めます。三平方の定理の斜辺だけを求める演算子です。



例えば左図のような直角三角形で斜辺を求める場合には次のようになります。

ans_val = 3 hyp 1

計算すると√10なので答えは3.16227・・・となりますが、PBPでは小数点は扱えないためans_valには3が代入されます。

なお、この演算結果は下記の式と等価です。

ans_val = SQR ((3 * 3) + (1 * 1))

■ＡＢＳ

絶対値を返します。絶対値は0からその値がどれだけ離れているかを表すものですので、下記のようにある値以上の時は、最大値からその値を引いた値として算出されます。
BYTE型で、127以上の場合にはABSは「256-値」で算出した値を返します。
WORD型で、32767以上の場合にはABSは「65536-値」で算出した値を返します。
LONG型の場合は、値が負の値だった時に「4294967296-値」で算出した値を返します。

■ＤＣＤ

DCD命令は該当するビットを1にセットします。

B0 = DCD 2

変数B0には、%00000100（二進数）が代入されます。尚、最下位がbit0....の順でbit7までです。

■ＤＩＧ

DIG命令は変数にセットされた値のうち指定された桁の値を抽出します。
10進数で扱われ、一の位はDIG関数では0です。以下に例を示します。

B0 = 123
B1 = B0 DIG 1

変数B0に格納された数字列123の1桁目を抽出して変数B1に値を返します。
1を指定していますので十の位の2が返され、変数B1には2が格納されます。

■ＮＣＤ

変数にセットされている値のうち、上位ビットで1がセットされているビット数を返します。

B0 = NCD %01001000

この例では、B0には7が返されます。

■ＲＥＶ

REVは指定したビットより下位のビットの値を反転します。
例えば下記の場合は、変数B0には %00110101　が代入されます。SPI通信等で出力したいデータをLSBから出力したい場合などに利用できます。

B0 = %10101100 REV 8

■ＳＱＲ

SQR命令では、変数の値を平方根してその解を返します。尚、答えは8ビットの整数で返されます。　　B0 = SQR W1

上の例では、変数W1の値を平方根して、その解をB0に返します。

■ＤＩＶ32

PBPでは「16ビット数値×16ビット数値=32ビット数値」の乗算ができますが、変数は最大でも16ビットの最高値（65535）までしか扱えません。この"DIV32"はこの問題を解消します。
下記の例を参照してください。

```
a       var word
b       var word
c       var word
d       var word

a = 500
b = 1000

c = a * b
d = DIV32 100
```

左のようにすると、まずc = a × bが実行されるため、結果としてcには「500000」が返るはずです。しかし、変数cはword型なので最高値は65535となり超過してしまいます。

そこで次の行に d=DIV32 100　と記述することで超過した変数cの値が DIV32 100と記載の100で除されます。
よって、変数dには 5000 が返ります。

つまりこの最終的にDIVで返った値dと、DIV32 xxxのxxxで指定した数値を乗すると変数cの値になります。

32ビットの計算は直接できないためこの方法で行うことができます。ただし、このとき変数cの値は、2147483647以下、変数dの値は32767以下でなければなりません。

※PBPLでは、DIV32は使用できません。

■ビット演算子

ビット演算子は、ビット毎に論理積や論理和といった演算をする演算子です。

&	ビット演算のAND
|	ビット演算のOR
^	ビット演算のExOR(排他的論理和)
～	ビット演算のNOT　　※～はチルダを表しています
&/	ビット演算のNOT AND
|/	ビット演算のNOT OR
^/	ビット演算のNOT ExOR

ビット演算の例

ANS = %10000001 & %11110000　　'ビット単位での論理積を求めます

上記のプログラム例では、ANSには %10000000 が入ります。

ビット演算の応用例

NOT演算子を使うと各ビットを反転することができます。これを利用することで、I/Oポートの状態を反転させることができます。下記の例は200ミリ秒毎にPORTBの出力をH-Lで反転させて点滅させるプログラムです。

```
do
        PORTB =~ PORTB
        pause 200
loop
```

■論理演算子

論理演算は、ビット演算と異なり算術結果はTRUE(1)又はFALSE(0)のいずれかになります。
IF文の条件式に用いて、条件式がTRUEになるかFALSEになるかで処理を分岐する時などに使用します。

演算子	詳細
AND 又は &&	論理積
OR 又は ||	論理和
XOR 又は ^^	排他的論理和
NOT 又は !	論理否定
ANDNOT	NAND
ORNOT	NOR
XORNOT	NXOR

論理積で2つのピンの入力を判定する例

論理積は、2つの評価式が1になった時だけ、その結果が真(1)になります。これをIF文の評価式とすることで簡単に処理を分岐できます。下記例はRB0とRB1がともに1(Hレベル)になった時にPORTDのLEDを点灯させる例です。

```
do
        IF (PORTB.0 AND PORTB.1) then
            PORTD = 255
        endif
loop
```

6-3、算術演算の方法

加減乗除は、算数の決まりと同様加減よりも乗除が優先されます。また、( ) 付き演算の場合、( ) 内の演算が優先されます。
演算結果は、代入演算子により右辺から左辺に代入されます。

W1 = W0 * 1000

変数W1には、W0の値を1000倍した値が代入されます。
LONG型変数を使用した場合には、符号付き演算が可能で負の値を扱うことができます。

L1 = 100 * -5

LONG型変数L1には、-500が代入されます。

除算は、あまりを無視する/と、余りを返す//があります。

W3 = 100 // 40

W3には、20が返ります。PBPLを使用してコンパイルする場合、PBPLはLONG型同士の演算の場合には符号付き演算として処理します。WORD型やBYTE型が混在する場合符号なし演算として処理されます。

6-4、ビット演算と論理演算

PBPでは、ビット演算としてAND、OR、排他的論理和(Ex-OR)の3つの演算子を用意しています。
下記に使用例を記載します。

演算	演算子	使用例
AND	&	B0 = B0 & %00000001
OR	|	B0 = B0 | %00000001
Ex-OR	^	B0 = B0 ^ %00000001

論理演算子は下記のようなものがあります。

演算内容	演算子
論理積	AND　又は && OR 又は ||
論理和	XOR 又は ^^
排他的論理和	NOT　又は ！
反転（否定）	ANDNOT
否定論理積(NAND)	ORNOT
否定論理和(NOR)	XORNOT
否定排他的論理和(NXOR)	

6-5、比較演算子

PBPでは以下の比較演算子を使用することができます。

演算内容	演算子
イコール（同等）	= 又は ==
等しくない	<> 又は !=
～よりも小さい	<
～よりも大きい	>
～以下	<=
～以上	>=

比較演算子と論理演算子は、IF文の条件式などを作るときに組み合わせて使用することがあります。

If（A == 3）AND（B >= 50）Then Label

例：Aが3でかつ、Bが50以上の場合、Labelラベルへ処理を分岐する

Memo

7. PICBASIC PRO Compilerのその他の機能と使用上の設定

7-1、動作周波数設定とDEFINE

PBPは動作周波数の指定がない場合にはPICンが動作クロック4MHzで動作しているもとしてプログラムを生成します。動作クロックは必ずプログラムの先頭で、defineを使って定義してください。動作周波数は次の記述で定義することができます。

define OSC 動作周波数（MHz）

PBPで指定できる動作周波数は下記の周波数に限られます。

3(3.58)、 4、 8、 10、 12、 16、 20、 24、 25、 32、 33、 40、 48、 64（MHz）

PLLで動作クロックを逓倍している場合には、逓倍後の実際の動作周波数を記述します。
例えば8MHzの外部水晶発振子を使用して4xPLLで4倍した場合には32MHzですので、その場合には"32"と記述してください。

DEFINEに続く文字は必ず大文字で記述してください。小文字だと正しく認識されません。

その他、各命令にはdefineが用意されていることがあります。PBPで設定されたデフォルト値を変更したい場合や配線情報、タイミング情報などを定義する必要がある場合にはdefineを使って定義します。defineの詳細設定につきましてはそれぞれの命令の項目に説明がありますので、そちらをご参照ください。

7-2、ADコンバータ機能、コンパレータ機能を搭載したPICマイコンを使用する場合の注意

PICマイコンでは多くのデバイスが、ADコンバータやコンパレータ機能を搭載しています。
これらのデバイスではADCの入力に割り当てられているピンや、コンパレータの入力に割り当てられているピンは、電源投入時すべてアナログ電圧入力モードに設定されています。

そのためデジタルIOとしてそのピンを使用しようとしても、そのままではデジタルIOとして使用できません。ADCやコンパレータを使用しない場合にはピンをデジタルモードに設定する必要があります。

IOピンをデジタルIOにするかアナログ入力ピンにするかはレジスタで設定できますが、デバイスによってレジスタ名が異なります。
多くのデバイスでは ANSELxレジスタが設定レジスタです。該当ビットが0でデジタルIO、1でアナログ入力ピンとなります。
その他ADCONレジスタで設定するデバイスもあるので、必ず使用するデバイスのデータシートを確認してください。

プログラムでデジタルIOとして使おうとしたピンがアナログ入力になっていると正しくロジック制御ができずトラブルの原因となります。

電源投入時は原則ADCの入力ピンとして割り当てられているピンはアナログ電圧入力になっていることに注意してください。

7-3、内蔵発振子搭載のデバイスを使用する場合

外部に発振子を取り付けなくてもPIC内部で動作クロックを作り出せる内蔵発振子搭載のデバイスではプログラム中でOSCCONレジスタの設定をする必要があります。

また、コンフィギュレーションビットの設定においてオシレーターの設定を、"INTOSC"（INTRCや内蔵OSCなど表記が異なる場合があります）に設定する必要があります。

内蔵発振子の周波数等はOSCCONレジスタで設定します。デバイスにより異なる場合がありますが多くのデバイスで8段階で設定可能です。PBPでは4MHzの倍数ならば設定が可能です。
OSCCONレジスタの設定はプログラムの先頭部分で記述します。設定値は使用するデバイスのデータシート内にあるOSCCONレジスタのページをご参照ください。

なお、周波数を定義する"DEFINE OSC xx"の記述は設定した周波数の値を記述してください。

PLLを使って発振周波数を逓倍する場合

4逓倍できる4xPLLが内蔵されているデバイスの場合で内蔵発振子を使う場合には逓倍後の周波数が、そのデバイスの最大周波数を超えないように設定する必要があります。

例えば最大周波数が32MHzのデバイスの場合でPLLを有効にした時、8MHzを指定する必要があります。（8MHz x 4xPLL = 32MHz） 間違えて10MHzなどにすると40MHzとなりそのデバイスの最大周波数を超えてしまいます。特に内蔵発振子を使う場合には設定する値に注意してください。

なお、動作周波数を速くすると消費電流が増えますので必ずしも速い動作速度が必要で無ければ、発振周波数を遅くしたほうが低消費電力になります。
PLLの有効/無効はコンフィギュレーションビットで行います。

7-4、ポートの入出力方向設定

PICでは各ピンを入力として利用するか、出力として利用するかの入出力方向を個別にレジスタで設定する必要があります。設定はTRISxレジスタで行います。（xにはポート名が入ります。）
対応するビットが0だと出力に、1だと入力に設定されます。

TRISx = 値

ほとんどのPICで電源投入時はすべてのIOピンが入力ピンになっていることに注意してください。
プログラムを作る場合には必ず各IOピンの入出力方向を確認しながらTRISxレジスタを適切に設定してください。
例えばPORTBの上位4ビットを入力、下位4ビットを出力に設定したい場合には次のようにします。

TRISB = %11110000

入出力方向の設定はとても重要です。
またADコンバータを使う場合には、アナログ電圧入力ピンはTRISxレジスタで入力ピンにする必要があります。

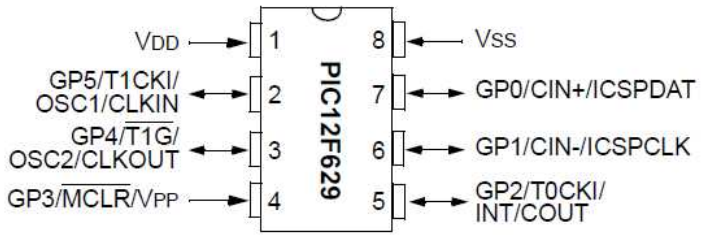
なお一部のPBPの命令では自動的にTRISxレジスタが設定される場合があります。

7-5、12ビットコアのPICや8ピンデバイスを使用する場合の注意

12ビットコアのデバイスには制限付きで対応しています。またPIC10Fシリーズにも制限付きで対応しています。これらのデバイスではデバイスの仕様上リソースに制限があり、下記の命令は使用できません。リソースの制限とは、スタックレベルが2レベルに制限されていることやRAM容量が小さいことなどです。

・ADCIN　・DATA　・DTMFOUT　・EEPROM　・FEREQOUT　・HPWM　・HSERIN　・HSEROUT
・ON INTERRUPT　・READ　・READCODE　・RESUME　・SERIN2　・SEROUT2　・USBIN　・USBINIT
・USBOUT　・WRITE　・WRITECODE

8ピンデバイスでは一部のデバイスでピン名はGPIOとなっています。PORTAやPORTBという呼び方ではない場合がありますのでご注意ください。



7-6、周辺機能のピンがPPS(Peripheral Pin Select)によって選択できるデバイスを使用する場合

PIC16F1xxxシリーズなど比較的新しいデバイスでは内蔵周辺機能で使うピンを任意のピンに割り当てられる機能、PPSが搭載されています。
例えばUARTのTXやRXピンをいくつかの選択肢の中から選んで設定できるようになっています。

このPPS機能を搭載したデバイスを使う場合で、内蔵ハードウエアの周辺機能を使う場合には正しくPPSによってピンを指定しないと思うようにプログラムが動作しません。

PPSとしてアサインできるピンについてはお使いのデバイスのデータシートに一覧が記載されています。

PBP3の命令でハードウエア周辺機能を使う場合には、その命令を使用する前に必ずPPSのピン割り当てを正しく指定する必要があります。
例えば、PIC16F1708でハードウエアUART通信をHSERIN、HSEROUT命令で使う場合には次のように適切にレジスタを設定しPPSによってピンを指定する必要があります。

例：UART通信においてRXピンをRC5、TXピンをRC4に割り当てて通信をする場合

```
ANSELC = 0
TRISC = %00100000

RC4PPS = %10100
RXPPS = %10101

ud  var  byte

main:

  hserin [ud]
  hserout [ud]

  goto main
```

TRISxレジスタは適宜入出力方向を適切に設定する必要があります。

RC4PSSレジスタはRC4ピンに何の機能を割り当てるかを指定するレジスタです。

bit 4-0	**RxyPPS<4:0>:** Pin Rxy Output Source Selection bits
	11xxx = Reserved
	10111 = Rxy source is C2OUT
	10110 = Rxy source is C1OUT
	10101 = Rxy source is DT(1)
	10100 = Rxy source is TX/CK(1)
	10011 = Reserved
	10010 = Rxy source is SDO

このレジスタに機能を指定する値を代入すると、そのピンが指定された機能のピンとして動作します。上図はデータシートの抜粋です。
UARTのTXピンは 10100 なのでこれを割り当てたいピンのPPSレジスタ(RC4PPS)に代入してRC4ピンをTXピンとしてアサインしています。

同様にして入力ピンのRXは、RXPPSピンにRC5を指定する値を代入しています。

bit 4-0	**xxxPPS<4:0>:** Peripheral xxx Input Selection bits 11xxx = Reserved. Do not use. 10111 = Peripheral input is RC7 10110 = Peripheral input is RC6 10101 = Peripheral input is RC5 10100 = Peripheral input is RC4

PPS機能があるものについては、必ずハードウエアの周辺機能を使う命令を実行する前に、PPS関連のレジスタを設定してピンの指定をする必要があります。ピンの設定をしないで使用すると正しく動作しません。
必ず使用するデバイスのデーターシートをお読みの上、確認してプログラミングしてください。

※上記のPIC16F1708の例は一例で、デバイスによってかなり設定するレジスタや値が違いますので必ずデータシートで設定を確認してください。

7-7、マルチステートメント

一行に複数のステートメントを記述する方法をマルチステートメントと言います。
PBPはマルチステートメントを利用できます。
マルチステートメントではステートメント毎に：(コロン) で区切ります。次に例を示します。

High PORTB.0 High PORTB.1 High PORTB.2	＝ 同じ意味になります	High PORTB.0 : High PORTB.1 : High PORTB.2

7-8、ラベルとコメント行

プログラム中には任意の文字列をラベルとして設定できます。
ラベル名には予約語及び記号や数字で始まる文字列は使用できません。

ラベルを設定すると、プログラムの任意の位置にgoto命令でジャンプできます。
ラベルは次のように設定できます。

ラベル名：　　(ラベル名の文字列の後ろにコロンを付けます)

ラベル名として指定したルーチンにはgoto命令またはgosub命令でジャンプできます。
goto命令ではそのラベル先に処理がジャンプするだけですが、gosub命令ではサブルーチンとしてラベル先にジャンプします。サブルーチン先でresume命令を実行するとジャンプ元に処理が戻ります。

プログラムを後から見て「これは何のための記述だっけ」などとならないようにプログラムにコメントを書いておくことができます。コメントは、 ' (アポストロフィー)をつけて記述します。
コンパイラはアポストロフィー以降の文字列を1行だけ無視します。

```
Low PORTE.2      ' LCD R/W Line Low
```

8. ディレクティブ

ディレクティブとは、プリプロセッサへの指示文#defineなどのことをさします。
プリプロセッサとは、コンパイラの中で、ある処理を行うソフトウェアに対してプログラムの準備的な処理を行う内容のことを言います。コンパイラは、ソースコードをコンパイルする前に、ディレクティブとして記述された内容の処理を施して、その後コンパイルを行います。

PBPでは、C言語などと同様にプリプロセッサディレクティブには#を付けて記述します。

8-1、 #CONFIG....#ENDCONFIG

PBPのソースコード内にコンフィギュレーションビットの設定内容を記述します。
#CONFIGは、コンフィギュレーションビットの設定内容をアセンブリ言語の中に埋め込むことで、その処理を行っています。
#CONFIGを使って、コンフィギュレーションビットの設定を記述すると、PBPはデフォルトのコンフィギュレーションビットの設定内容をスキップして、ここでユーザーが記述した内容をコンパイルに反映します。

記述する場合には、次のように記述します。

```
#CONFIG
      __config _CONFIG1, _HS_OSC & _WDT_OFF & _MCLRE_ON & _LVP_OFF & _CP_OFF
#ENDCONFIG
```

又は次のようにも記述できます。

```
#CONFIG
      CONFIG  XINST     = OFF
      CONFIG  PLLDIV    = 5
      CONFIG  WDTPS     = 512
      CONFIG  CPUDIV    = OSC1
      CONFIG  OSC       = HSPLL
#ENDCONFIG
```

より詳しい説明は本書P.23～24を参照してください。

8-2、 #DEFINE

#ＤＥＦＩＮＥ constant_name {value}

#DEFINEは、あかじめ決まっている数値を定数として、名前を付けることができるディレクティブです。
何か、決まった値について意味がある場合、それに名前をつけておくことでプログラムの可読性が向上します。PBPでは#DEFINEディレクティブとして作られた文字列の使用は、ディレクティブである#IF, #IFDEF, #IFNDEF, #MSG, #ERROR, #WARNING　で使用できます。

注意が必要なのは、この#DEFINEで作る定数はあくまでもディレクティブの中で使用するものであり、PBPのソースプログラムの中では使えないということです。ソースプログラムの中で定数を使いたい場合には、CONキーワードを使って、定数を宣言します。

constant_nameには、PBPで使われている予約後や宣言済みの変数、ポート名やPICマイコンのレジスタ名を除いた文字列を指定できます。

valueには値を記述します。
その他、PBPにはあらかじめ用意された文字列があり、この文字列を使用するとコンパイラが自動的にその内容に応じた値を取得します。

__DATE__	パソコンからコンパイルの日にちを取得します。
__TIME__	パソコンからコンパイルの時間を取得します。
__TIMESTAMP__	パソコンからコンパイルの日時を取得します。
__POCESSOR__	ターゲットデバイスの型式を取得します。
__VERSION__	PBPのバージョン値を取得します。
__LONG__	PBPL(long型対応PBP)の使用の有無を取得します。 1ならばPBPLを使用していることを示します。
__LINE__	ライン番号を取得します。(#MSGや#ERRORなどのレポートに使います。)
__FILE__	ファイル名を取得します。

8-3、 #ERROR

ＥＲＲＯＲ {"message"}

#ERRORディレクティブはコンパイラが表示するエラーメッセージを指定します。
#IFと組み合わせて使用することで、エラーメッセージをコンパイル時に表示します。
下記は、PBPL(Long型変数を使うPBP)を無効に設定している際に"This is pbpw."と表示される例です。

```
#IF __LONG__ = 0
    #ERROR "This is not PBPL."
#ENDIF
```

この記述をした状態でPBPLでなくコンパイルするとMicroCode Studioでエラーが表示されます。

```
Results
PICBASIC PRO(TM) Compiler 3.0.1.4, (c) 1998,
All Rights Reserved.
[ERROR] 887_test3.pbp(18): This is not PBPL.
```

8-3、 #IF....#ELSE....#ENDIF

＃ＩＦ condition {AND|OR condition}

コンパイルをする前に、コンパイルの動作をIF文で分岐ができるのが#IFディレクティブです。
よく利用される目的として、1つのプログラムをいくつかのPICマイコンで使うような場合（別のPICマイコンに移植するような場合)、プログラムの大部分は共通であっても、レジスタの設定などは異なることが多いですが、#IFディレクティブで分岐することで、デバイスの種類によってレジスタの設定などを変えることができます。

```
#IF __PROCESSOR__ = "16F877A"
    ADCON1 = 7
#else
    ANSEL = 0
    ANSELH = 0
#ENDIF
```

例えばPIC16F887AではADコンバータに関するレジスタはADCON1です。PIC16F887ではANSELレジスタです。下記のように記述するとデバイスによって処理を分岐できます。
命令にあるIF命令と、#IFディレクティブの違いは、#IFディレクティブは、コンパイルの時にその記述内容が評価され、プログラムが動作している時には動作しません。

8-4、 #IFDEF....#ELSE....#ENDIF

＃ＩＦＤＥＦ constant

#IFDEFは、#DEFINEの定数が定義されているかどうかをコンパイル時にチェックするディレクティブです。すでに定義されている場合と、定義されていない場合で処理を分岐できます。
例えば、ADコンバータの使用の有無を設定したい時、通常は、ADコンバータ関連のレジスタに値を代入して無効/有効を設定します。#IFDEFを使えば、文字列でその有効/無効をコントロールできます。

次の例は、"#DEFINE SET_DIGITAL"という定数ディレクティブを記述した場合には、"ADCON1 ＝ 7"の文がコンパイル時に適用されてピンがデジタルI/Oになります。
"#DEFINE SET_DIGITAL"を記述しなかった場合は、"#ELSE"以降がコンパイルされますので、"ADCON1 ＝ %00000010" がコンパイルに適用されます。

```
#DEFINE  SET_DIGITAL

#IFDEF  SET_DIGITAL
      ADCON1 = 7
#ELSE
      ADCON1 = %00000010
#ENDIF
```

#IFDEFと逆の動作をする #IFNDEF...#ELSE...#ENDIF というディレクティブもあります。
こちらは、定数の定義がない時に真、ある時に偽となります。

8-5、 #MSG

＃ＭＳＧ "message"

#MSGは、コンパイル時にメッセージを表示させます。文字列は+演算子で連結できます。

```
#MSG "Compiling for target " + __PROCESSOR__
```

```
Results
PICBASIC PRO(TM) Compiler 3.0.1.4, (c) 1998, 2011 microEng
All Rights Reserved.
[MESSAGE] 887_test3.pbp(17): Compiling for target 16F887
```

MSCでは上記のように表示されます。

8-6、 #WARNING

#WARNINGは、コンパイル時に警告メッセージを表示させます。文字列は+演算子で連結できます。
コンパイル時に誤った設定等で問題が起きないように#IFディレクティブなどと組み合わせて使います。

```
#IF  __LONG__ = 0
      #WARNING  "This program recommended for PBPL"
#ENDIF
```

上記例はLong変数を使うプログラムでPBPLが選択されていない場合に警告を表示させます。

9. PICBASIC PRO Compilerの命令

文中の｛　｝は省略可能の内容を示します。［　］はかっこごと記述します。
\（エンマーク）は英語環境では＼（バックスラッシュ）として表示されます。

9-1、@ （アセンブラの挿入）

BASICプログラム中にアセンブリ命令を挿入したい場合、その行の先頭に@を付けます。
この記号を使用することによって、BASIC言語中にアセンブリ言語を挿入できます。

例：下記例はnop命令を挿入します。
　@　nop

例：拡張子.asmのアセンブリファイルをインクルードする場合にも使えます。
　@　Include　"file.asm"

@を使用してページやバンクを変更することはできません。

9-2、ADCIN　（ADコンバーターから値を取得する）

ＡＤＣＩＮ Channel 、 Var

ADCIN命令はADコンバータ機能を内蔵しているPICマイコンにおいて、指定したチャンネルから電圧値を読み取って変数Varに格納する命令です。

Channelの値はアナログ入力のチャンネル番号です。ピン名でないことにご注意ください。
PICマイコンによってチャンネル番号の組み合わせが異なりますので詳細は使用するPICマイコンデータシートをご覧ください。データシートの各ピンの名称の部分に AN2 などとANに続く数値がチャンネル数です。

ADCIN命令の使用に際しては、ADCON1レジスタやANSELレジスタなどADコンバーター関連のレジスタを正しく設定する必要があります。設定が間違っているとADコンバーターは動作しません。

ADコンバーター関連のレジスタでは、まずどのピンをアナログ入力モードにするのかという点と、Vref(基準電圧)をどのように設定するのかを決めます。設定はPIC内のレジスタで設定を行います。

デバイスによってアナログ電圧入力ピンに設定するためのレジスタが異なります。比較的昔いデバイスではADCON1レジスタで設定しますが、その他のほとんどのデバイスではANSELxレジスタでIOピンをデジタルIOとするか、アナログ入力ピンとするかを設定するようになっています。
デバイスによって違うため、デバイスのデータシートをご確認ください。

ADコンバーターはデバイスの種類によって、分解能値が8ビット、10ビット、12ビットとあります。ADコンバーターの値を格納するレジスタは物理的に8ビットです。そのため、10ビットや12ビットタイプのデバイスでは、2ビット又は4ビット分の格納のために別のレジスタを使用します。このとき上位2ビットまたは4ビットを別のレジスタとするのか、下位2ビットまたは4ビットを別のレジスタとするのかを、ADコンバーター関連のレジスタで決めるようになっています。
PBPは必ず上位2ビット又は4ビットを上位レジスタに格納する指定方法、"右詰指定"としなければなりません。データーシートをご覧になり適切な値に設定してください。

ADCIN命令を使用する場合には、必ず該当ポートの入出力方向設定用のレジスタTRISxレジスタで該当ピンを入力ピンとします。この設定が適切でないと思わぬ故障や動作をしてしまうことがあります。アナログ入力モードに設定したピンは必ず入力モードになるようTRISxレジスタの該当ピンの値を1としてください。

結果を格納する変数Varは、ADコンバーターの分解能に応じて型を決めます。8ビットのものであればbyte型、10ビット及び12ビットのものの場合にはWord型の変数を指定します。

ADCIN命令使用に際して、DEFINEが3つありますので、ADCIN命令を使用する場合には必ずプログラムの先頭部分で定義を行ってください。

"DEFINE　ADC_BITS　10"

A/Dコンバーターの分解能を指定します。使用するPICマイコンの種類に応じて設定を行います。

"DEFINE　ADC_CLOCK　3"

AD変換の際のサンプリング時間の生成には、内蔵されている専用のRC発振子か、動作クロックかのどちらかを選択できます。設定できる値は0～3です。3に設定すると内蔵のADコンバーター専用のRC発振回路が選択されます。

"DEFINE　ADC_SAMPLEUS 50"

マイクロ秒単位で、サンプリング時間を設定します。このサンプリング時間とは、A/D変換の準備が完了し変換が開始されてからA/D変換が終了するまでのプログラムが待機する時間を示します。通常は50マイクロ秒程度に設定します。

下記の例はPIC18F45K22でAN0の入力値をLCDに10進数で表示した場合です。

```
DEFINE  ADC_BITS     10
DEFINE  ADC_SAMPLEUS 50
DEFINE  ADC_CLOCK    3

AN0RES  var word
TRISA = 1
ANSELA = 1
ADCON0 = %00000001
ADCON2 = %10000000

main:
  adcin 0, AN0RES
  lcdout $fe,1, dec4 AN0RES

  pause 100
  goto main
```

9-3、ARRAYREAD（配列からデータを読み込む）

ＡＲＲＡＹＲＥＡＤ ArrayVar, {Maxlength , Label,} [Item....]

ArrayVarで指定したバイト配列から値を検索してItemで指定したバイト型変数に値を格納します。シリアル通信等で取得したバイト型データをバイト配列に格納した後、指定したデータを配列の中から検索したり、指定した要素のデータだけを取得したりすることができます。一般的には、ARRAYREAD命令はシリアル通信で受信したデータの解析に使用するための命令です。
省略可能なMaxlengthに最大の読み取りバイト数を指定すると、読み取りを行う配列のサイズの最大値を指定できます。Maxlengthで指定できる最大値は256までです。また特定のデータを検索する場合には、省略可能なLabelに飛び先ラベルを指定しておくと、検索したデータが見つからなかった場合に指定したラベルにジャンプさせることができます。
省略可能なMaxlength並びにLebelを指定する場合には、"WAIT"や"WAITSTR"の修正子を使う場合が一般的です。
ARRAYREAD命令では、各種修正子が使用できます。データ読み取り時にデータを修正したり、配列に取り込んだり、配列の中から指定したデータを検索することができます。

修正子	動作内容
BIN{1～32}	2進数として読み込みます。
DEC{1～10}	10進数として読み込みます。
HEX{1～8}	16進数として読み込みます。
SKIP n	nバイトデータをスキップします。
STR ArrayVar2\n{\c}	nで指定したバイト数のデータをArrayVar2配列に読み込みます。省略可能なcで終端します。
WAIT( )	( )内で指定したデータを受信するまで待機します。
WAITSTR ArrayVar2{\n}	バイト配列ArrayVar2内格納されている文字列を受信するまで待機します。省略可能なnを指定すると配列の長さを指定できます。

例1：バイト型配列Aの要素0と要素1のデータをバイト型変数B0とB1に格納する。

```
A var byte[10]
B0 var byte : B1 var byte
ARRAYREAD A, [B0 , B1]
```

例2：バイト型配列Aの要素3のデータを取得する。

```
ARRAYREAD A, [SKIP 3, B0]
```

例3：バイト型配列Aの中から文字"X"を検索し、該当する文字があった場合には、その文字の次の要素から3バイト分のデータをバイト型配列Bに格納する。但し、検索するデータの長さは最大10バイトとし、10バイト内に"X"が見つからない場合には、ラベルtlabelにジャンプする。

```
A var byte[20] : B var byte[5]
ARRAYREAD A, 10, tlabel, [WAIT("X"), STR B\3]
```

9-4、ARRAYWRITE（配列にデータを書き込む）

ＡＲＲＡＹＷＲＩＴＥ ArrayVar, {Maxlength , Label,} [Item....]

ArrayVarで指定したバイト配列に、Itemで指定したデータを書き込みます。
I2CWRITE命令やLCDOUT命令など、配列を扱う別の命令とともに使用することを前提としています。
Itemで指定したデータは、ArrayVar配列の先頭から書き込まれます。

省略可能なMaxlengthには書き込むデータの最大要素数を指定します。指定できる最大値は256までです。書き込もうとするデータがここで指定した最大要素数を超えた場合には、Labelで指定したラベルへジャンプします。不定長のデータを書き込む場合で、書き換え元の配列に対して指定したサイズよりも大きな書き換えが発生しないようにすることができます。
ARRAYWRITE命令では各種修正子が使用できます。データ書き込み時にデータを修正したりすることができます。

修正子	動作内容
{I}BIN{1～32}	2進数として書き込みます。
{IDEC{1～10}	10進数として書き込みます。
{I}HEX{1～8}	16進数として書き込みます。
REP c\n	キャラクタcをn回繰り返して書き込みます。
STR ArrayVar2\n	ArrayVar2配列からnで指定したバイト数分のデータを書き込みます。

補助修正子"I"はデータの進数形式を%,#,$で数値の前に付加します。2進数の場合には%、10進数の場合には#、16進数の場合は$をそれぞれ付加します。

9-5、ASM･･･ENDASM　（インラインアセンブラの指定）

BASICプログラム中にアセンブラ言語を挿入します。（インラインアセンブラ機能）
ASMで開始し、ENDASMで終了します。ASM～ENDASM間にアセンブリ言語を記述します。

```
例      ASM
            BSF     PORTA、0            'PORTAのbit0を1にセット
            BCF     PORTB、0            'PORTBのbit0を0にセット
        ENDASM
```

※インラインアセンブラ使用中には、アセンブラ言語でバンクの切替や設定はしないでください。
また、汎用レジスタをアセンブラ内で使用すると、BASICの動作に影響を与えます。

9-6、BRANCHL（指定したラベルへ分岐）

ＢＲＡＮＣＨＬ　Index、[Label1{、Label1....}]

BRANCHLはIndexの値（順列）で指定したLabelの分岐先へ実行内容を分岐します。

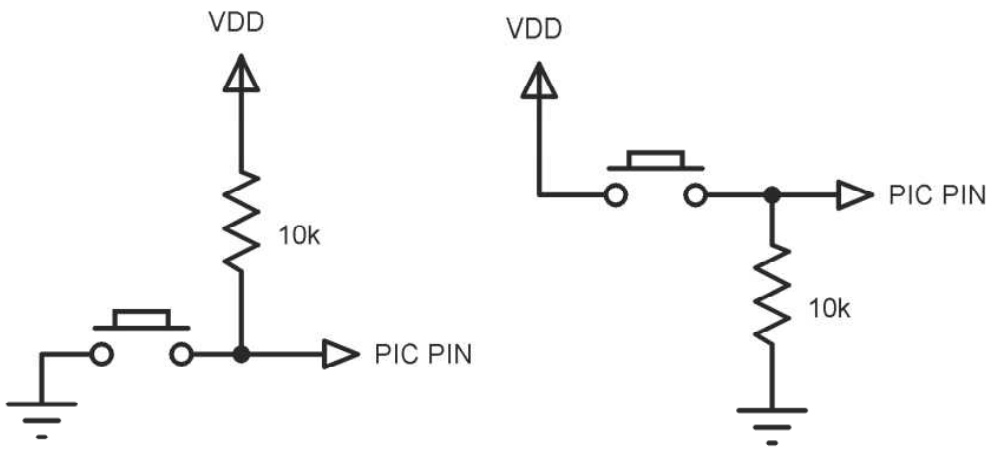
Index　　変数又は定数で値が0～Nの分岐指定
Label　　分岐先のラベル名

［　］内に記述するラベルの順番は、0123・・・の順番となっています。Indexで指定した値に対応したラベルに処理を分岐します。Indexの値によって処理を複数のラベル先に分岐できるため、分岐先が多い場合にはIF文で記述するよりも、プログラムが見やすくなります。

9-7、BUTTON（チャタリング対策を施したボタン入力処理）

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

ピンに接続した押しボタンスイッチなどを使用し、ピンの状態をH-Lに変えて、それを判定する命令です。
ピンのH-L状態は、IF文でも判定できますがBUTTON命令では、オートリピート（長く押し続けた時に何回押したことにするか）とそれに伴うパラメータを設定できます。チャタリングの直接的な防止の機能はありません。ピンの論理状態の判定にはIF文やSELECT文を使うのが一般的です。



PICマイコンの入力ピンはハイインピーダンスですので、スイッチの接続は上記のように抵抗器でプルダウンするかプルアップして使用します。左側はスイッチを押すと、PIC PINが0になります。右側はスイッチを押すとPIC PINが1になります。

・Pin　　PORTB.0 などのように判定するスイッチを接続したピン名を記述します。

・Down　　0又は1。Hレベル又はLレベルどちらでボタンが押されたとみなすかを指定します。
0・・・該当ピンがLowレベルでボタンを押されたことになります。
1・・・該当ピンがHighレベルでボタンを押されたことになります。

・Delay　　0～255の値　オートリピート開始前のカウント数です。ここで指定した値は実際にはループ回数で、ある一定時間を1ループとして何ループ経過した時に、まだスイッチが押された状態になっているかを判定します。0に設定するとオートリピートが有効になります。
255に設定するとオートリピートは無効になり、スイッチ長押しの時に何回も押されたことになりません。

・Rate
0～255の値　上記オートリピートループの周期の設定です。値を小さくするとリピートが早くなります。0でオートリピートは無限、255でオートリピートサイクルは1に設定。

BVarはBUTTON命令作業用スペースです。BYTE型変数を記述します。使用前に0を代入します。
Actionは0か1を記述し、0で押されていない時、１で押されている時に指定した分岐先へ分岐します。
Labelは、Actionで指定した状態の時に分岐する分岐先ラベル名を記述します。

上記説明でオートリピートという言葉が出ていますが、オートリピートとは、いわゆる長押しした場合に入力を何回入力したかということを言います。引数のdelayで設定した値は具体的にはループの回数で、0～255で指定したループ回数分スイッチが押された状態になっていると、オートリピートを開始するという意味になります。この時、255の最大値にするとループ回数が255となり、オートリピートをしなくなります。0に設定すると、ループが回らない状態でスイッチが押されたことになりますので、（つまり時間遅延がなくなるため、スイッチを押してすぐにオートリピート＝何回もスイッチ入力があったこととなる）オートリピートが有効になります。よって何回も押されたことになるのでチャタリング防止機能はなくなります。

以上の理由からチャタリング防止機能の直接的な関係はありませんが、ある瞬間だけをONにするパルス波を印加することは機械スイッチではできませんので、複数押しの防止になります。
複数押し防止のためここでは255を設定することが多いです。

引数のRateは、オートリピートの周期で、値を小さくするとリピートが早くなります。引数として

設定するBvarは上記のループ用のカウンタとして使用されます。何回も同じ変数名を使用する場合には、Button命令の前で0を代入しておきます。（宣言直後でしたら初期値0が入っているため不要です）

なお、このBUTTON命令だけでチャタリングの防止を完全に行うことはできません。
スイッチ入力はピンの状態変化ですので、IF文で作ることも多くあります。IF文でスイッチ入力を判定後、飛び先ルーチンである一定時間、例えば100mS～300mS待って再度IF文で今度はスイッチが離された状態であるかを確認すると、簡易的なチャタリング防止を施すことができます。離されていなければ再度ループを回って確認し、離されるまでループを回します。

9-8、CALL（アセンブラ言語内のサブルーチンをコール）

ＣＡＬＬ Label

アセンブリ言語内の任意のラベルをサブルーチンとして呼び出します。
アセンブリ言語内のラベルには、_（アンダーバー）がついています。

PBPのソースプログラム内での任意のラベルへのサブルーチンへのジャンプはGOSUB命令を使います。本CALL命令は、アセンブリプログラム内に記述されたラベルへのジャンプですのでご注意ください。

9-9、CREAR（すべてのRAMレジスタをリセット）

ＣＬＥＡＲ

バンクに関係なくすべてのRAMレジスタの内容をクリアします。
宣言済み変数の内容もすべて0になります。

PICマイコンは電源投入時にはRAMの内容はすべてランダムであり定められた値にはなっていません。CLEAR命令を使うと、簡単にすべてのRAMを0にクリアできます。
但し、レジスタ(SFR)のクリアはしません。RAMのクリアのみです。

※12ビットコアデバイスでは、本CLEAR命令はバンク0のレジスタは0にしません。

9-10、CLEARWDT（ウォッチドッグタイマーをクリアー）

ＣＬＥＡＲＷＤＴ

ウォッチドッグタイマーは、PICマイコンが正常かどうかを常に監視するためのタイマです。ウォッチドッグタイマーは、タイマがタイムアップするとハードウェアにリセットをかけます。
CLEARWDT命令を実行すると、ウォッチドッグタイマをクリアして、タイムアップしないようにします。一定時間毎にウォッチドッグタイマーをクリアすることでプログラムが停止してウォッチドッグタイマーがリセットできなくなると、ウォッチドッグタイマーがタイムアップしてリセットがかかります。

アセンブラ言語レベルで挿入されるclrwdt命令をすべて取り除くには、下記のDEFINEをプログラムの先頭で記述します。

DEFINE NO_CLRWDT 1

9-11、COUNT（パルスのカウント）

ＣＯＵＮＴ Pin、Period、Var

指定した時間(Period=ミリ秒単位)に指定ピン(Pin)に入力されたパルスの数をカウントし、その結果を変数Varに代入します。
COUNT命令は、ピンに印加されたLowからHighへの変遷をカウントします。

・Period　　カウント計測する時間　単位はミリ秒。
　　波形のデューティ比は50%であることが条件です。(H-Lの間隔が等しい)
・Var　　カウント値を格納する変数。Word型変数を使用することをお奨めします。

動作周波数4MHzの時、20μ秒ごとに、20MHzの時は4μ秒毎にピンの状態をチェックします。
よって、この時間間隔よりも短い時間のパルスはカウントできません。
この命令はPICマイコンを動作させる周波数により影響をうけます。4MHz動作の場合20マイクロ秒ごとに、20MHz動作の場合は、4マイクロ秒毎にピン状態を検出します。よって4MHz動作の場合には最大25KHzまで、20MHz動作の場合には最大125KHzまでのカウントが可能です。
なお、デューティー比が50%でない場合、正しい計測ができない場合があります。

計測時間を1000ミリ秒に設定すると周波数として値を取得できます。

9-12、DATA（EEPROMに定数を記憶させる）

ＤＡＴＡ {@Location、}Constant{、Constant...}

EEPROMメモリを搭載したPICマイコンのEEPROMに定数を記憶させます。
@Locationが省略された場合、最初のデータはアドレス0番地に書き込まれ、以降のデータをそれに引き続き、書き込まれます。

Constant　　数値を記述します。

プログラム実行中に、このDATA命令で記憶された値を参照することができますが、値の更新はできません。プログラム実行中に値を更新する場合には、WRITE命令を使用します。

9-13、DEBUG（UARTでデータを出力する）

設定した条件にて、変数の内容をシリアル通信(UART)で出力します。（TXのみです）
PBPにはこの他にもいくつかのシリアル通信命令がありますが、このDEBUG命令は簡単に変数の内容等をシリアル出力できるようにしたもので、あくまでもデバッグ作業に利用するようにしたものです。データ入力はできず、単にデータをTXデータとして出力するのみとなります。

ＤＥＢＵＧ Item{、Item...}

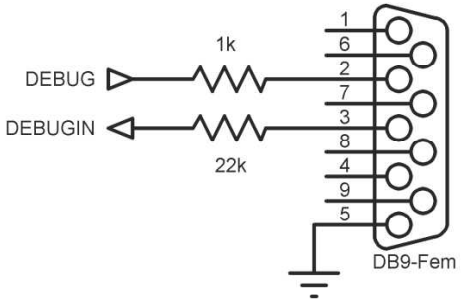
Itemには変数又は文字列を指定します。変数の場合には、変数の前に#記号を付けます。
この他、変数の内容を10進数で出力する場合にはDEC、16進数の場合にはHEXなどいくつかの修正子があります。修正子の詳細は、本書のSERIN2命令の項目をご覧ください。
文字列を指定する場合には、" "（ダブルクォーテーション）で囲みます。

DEFINEを利用して通信ポート及び通信速度を設定します。
データプロトコルは、データ長8ビット、1ストップビット、ノンパリティに固定されていて変更できません。

```
DEFINE   DEBUG_REG     PORTB       出力ポートをPORTBに設定します。
DEFINE   DEBUG_BIT     0           出力ビットをビット0にします。
DEFINE   DEBUG_BAUD    2400        通信速度をbps単位で設定します。
DEFINE   DEBUG_MODE    0           0で正論理、1で反転します。
```

ソフトウエア的にUART通信を実行しますのでハードウエアUARTのないPICや、ハードウエアUARTのTXピンとしてアサインされていないピンも指定することができます。

通常パソコンのRS232Cポートと接続する場合には、MAX232等のレベル変換ICを介してPICマイコンと接続します。直接接続する場合には必ず抵抗器を入れる必要があります。



※PICの動作クロックは通信速度9600bpsで4MHz、115.2kbpsの場合には20MHz以上の必要があります。

9-14、DEBUGIN（UARTからデータを入力する）

シリアル通信(UART)でデータを受信します。(RXのみです)
PBPには、この他にもいくつかのシリアル通信命令がありますが、このDEBUGIN命令は簡単に外部からの信号を入力できるようにしたもので、あくまでもデバッグ作業に利用するようにしたものです。

ＤＥＢＵＧＩＮ {Timeout、 Label、 } [Item{、Item...}]

Itemに記述した変数に、シリアル通信で受信したデータを格納します。
DEBUGIN命令では、データを受信するまでプログラムはDEBUGIN命令の場所で停止し、受信を待機します。この受信待ちの状態では、受信するまで処理が停止するため、他の処理ができない状態となります。そこで、定義した時間が経過してもデータを受信できない場合には、この命令を終了するのが、Timeout時間の設定です。Timeoutにミリ秒単位で、タイムアウトの時間を設定します。Labelにはタイムアウト時にジャンプするジャンプ先のラベル名を記述します。

DEFINEを利用して通信ポート及び通信速度を設定します。
データプロトコルは、データ長8ビット、1ストップビット、ノンパリティに固定されています。

```
DEFINE   DEBUGIN_REG     PORTA       入力ポートをPORTAに設定します。
DEFINE   DEBUGIN_BIT     3           入力ビットをビット3にします。
DEFINE   DEBUGIN_BAUD    2400        通信速度をbps単位で設定します。
DEFINE   DEBUGIN_MODE    0           0で正論理、1で反転します。
```

ソフトウエア的にUART通信を実行しますのでハードウエアUARTのないPICや、ハードウエアUARTのTXピンとしてアサインされていないピンも指定することができます。

通常パソコンのRS232Cポートと接続する場合には、MAX232等のレベル変換ICを介してPICマイコンと接続します。直接接続する場合には必ず抵抗器を入れる必要があります。(DEBUG命令の項をご参照ください。)

9-15、DO...LOOP（条件を指定してルーチンをループさせる）

ＤＯ｛ＵＮＴＩＬ Condition｝｛ＷＨＩＬＥ Condition｝
　ステートメント....
ＬＯＯＰ ｛ＵＮＴＩＬ Condition｝｛ＷＨＩＬＥ Condition｝

DOとLOOPで囲われた部分のルーチン（ステートメント）を繰り返し実行します。
省略可能なUNTIL又はWHILEを省略すると、永久ループとなります。

UNTILを指定すると終了条件が、WHILEを指定すると継続条件がそれぞれ設定できます。
DOの後ろに記述するとステートメントが実行される前に条件文が判定されます。LOOPの後ろに記述すると最低1回ステートメントが実行された後、条件文が判定されます。

例1：永久にPORTBを100ミリ秒間隔で点滅させる

```
DO
    PORTB.0  =~ PORTB.0
    PAUSE 100
LOOP
```

例2：変数iの値が10以下の間、処理を継続させる

```
i = 1
DO WHILE i < 11
    PORTB =~ PORTB
    PAUSE 100
    i = i + 1
LOOP
```

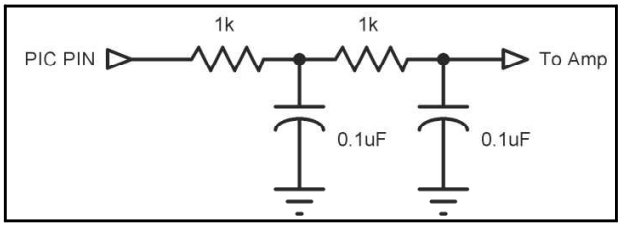
9-16、DTMFOUT（DTMF信号を出力する）

DTMF(Dual-Tone Multi-Frequency)を指定したピンから出力します。
DTMFは、16種類の符号を低側、高側2つの周波数帯域によって信号を発声させます。電話機のトーン信号（ピ・ポ・パの音）です。

ＤＴＭＦＯＵＴ pin、{Onms、Offms、}[Tone{、Tone...}]

pin	音のパルスを出力するピンを指定します
Onms	それぞれの音の発生時間　単位はミリ秒、省略時は200msになります
Offms	次の音が出るまでの無音時間　単位はミリ秒、省略時は200msになります
Tone	0～15の番号で、0～9はプッシュ電話機のトーン信号と同じ周波数の音を発生 10は*キー、11は＃キーに相当します、12～15はA～Dに相当します

PICのピンとオペアンプを接続する際には、下記のようなフィルター回路を作り接続します。



DTMFOUT命令を使用する場合、動作周波数20MHz以上が推奨されます。

9-17、EEPROM（EEPROMに値を記憶させる）

ＥＥＰＲＯＭ {Location、}[Constant{、Constant....}]

EEPROM命令はPICに内蔵のオンチップEEPROM(データROM)に定数を書き込む命令です。
この命令はWRITE命令と異なりプログラム実行中にデータEEPROMに値を書き込むのではなくHEXファイルをPICマイコンに書き込む際に定数として値をEEPROMの指定した番地に書き込む命令です。
EEPROMに定数を複数バイト連続して記録します。書き込み開始アドレスはLocationで指定します。省略するとアドレス0から書き込まれます。それ以降は、次のアドレスから順に書き込まれます。
Constantは書き込む整数又は文字列を記述します。文字列の場合はASCIIコードとして記憶されます。

EEPROM 4、［10、20、30］　　'4番地より10、20、30を順にEEPROMに書き込みます

なお、EEPROMに書き込んだ内容はPICマイコンの電源を切断しても保持されます。

9-18、END（プログラムを終了する）

ＥＮＤ

プログラムを終了します。各I/Oピンの状態はこの時点の状態を維持します。
END命令は、スリープ命令をループすることで実行されています。
END命令ははサスペンドではないため、一度END命令を実行すると再起動又はハードウエアリセットをしないとプログラムは実行できなくなります。

PICマイコンの周辺モジュール(USARTなど)を使用している場合でサスペンドしたい場合にはSTOP命令を使います。

9-19、ERASECODE（フラッシュメモリエリアのデータを消去する）

ＥＲＡＳＥＣＯＤＥ Block

PIC18Fシリーズや12ビットコアデバイス(PIC12F519等)では、WRITECODE命令でデータを書き直す前にコードの消去が必要です。これらのデバイスでは、1回のデータ消去にあたり1ブロック毎のブロック単位で行います。ブロックのサイズは、デバイスによって異なっていますが、多くは32ワード(64バイト)単位となっています（デバイスにより異なるためデータシートでの確認が必要です)。

このERASECODEにおけるブロックサイズの指定は通常、書き込みを行うブロックサイズより大きくする必要があります。消去するブロックの最初の場所は、引数のBlockで指定します。PIC18Fデバイスに関しては引数のBlockはワードアドレスではなく、バイトアドレスです。このBlockの値を指定する時は、プログラムコードのエリアを指定しないように注意する必要があります。プログラムコードが格納されているエリアを指定してしまうと、プログラムコードが喪失してしまいプログラムの挙動が不正になり重大なバグを引き起こすことになり致命的な問題となります。この命令を使用する場合にはPICのメモリー構造に関して十分な知識を必要とします。

なお、ここで指定するBlockの値は、そのブロックの最初のアドレスと記載しましたが、このアドレスは相対番地です。例えば多くのPICではプログラムメモリーは5番地から始まりますが、あくまでもこのERASECODEで指定するBlockのアドレスは先頭が0です。すなわち"0"を引数として指定すると、実際には5番地から1ブロック分イレースされることになります。プログラムメモリーは5番地からだから先頭アドレスは5と間違わないようご注意ください。(ここの"5"は絶対番地です)

ERASECODE命令で使用されるBlock引数の値は、マイクロチップ社によって定義されている「ブロックの開始アドレスの値と一致」しなければなりません。もしこの決まりを無視して、ブロックの最初のアドレス以外を指定しまった場合、動作は全く保証されずひいては、プログラム全体の挙動に対しても重大な問題を引き起こす原因となります。

もし、PIC18Fシリーズを使用している場合には、たいがいがブロックサイズは64バイト単位です。そのため指定するブロックのアドレスは64の倍数になります。(0,　64,　128,　..... 3200,　3264など・・・)
例としてPIC18F452のデータシートを見てみましょう。データシートには次のような記載があります。

5.0　FLASH PROGRAM MEMORY

The FLASH Program Memory is readable, writable, and erasable during normal operation over the entire VDD range.

A read from program memory is executed on one byte at a time. A write to program memory is executed on blocks of 8 bytes at a time. Program memory is erased in blocks of 64 bytes at a time. A bulk erase operation may not be issued from user code.

イレースについては64バイト毎に行われると記載があります。
同様にして他のデバイスであってもデータシートには必ずブロックサイズの記載がありますので、使用するデバイスのデータシートを必ず確認ください。

また、12ビット命令セットのデバイス、例えばPIC12F519やPIC16F526では同様にWRITE命令でデータを書く前にERASECODE命令を使ってデータを消去する必要があります。

ERASECODE命令を使えるようにするためには、HEXファイルをPICライターで書き込む際に設定するコンフィギュレーションビットの設定で、「フラッシュプログラムの書き換え」に関する項目を有効（=Enabled）に設定しておかなくてはなりません。
ブロック単位でのイレースをサポートしていないデバイスでこの命令を使うとエラーになります。

※前ページのデータシートの抜粋でも分かるとおり、消去する時の1ブロックのサイズと、データを書き込む時の1ブロックのサイズは異なります。例えばPIC18F452の場合、イレースは1ブロックあたり64バイト、書き込みの場合には1ブロックあたり8バイトです。

※PIC16Fデバイスでは実際にはプログラムメモリは14ビットサイズ(0～16383)です。一般的に1ワードという場合には16ビットのことをいいますが、ここでは1ワードとはPICにおける1単位という意味で14ビット長をいいます。

9-20、EXIT（ループ構造から抜ける）

ＥＸＩＴ

FOR...NEXT等のループや、IF文の構造からプログラムの処理を抜けさせ、次の文から実行させる命令です。ループ等で、ループを抜けるための条件式の条件が満たされなくても、この命令で強制的にループから抜けることができます。
EXIT命令は下記の命令で使えます。

DO...LOOP
FOR...NEXT
IF...THEN
WHILE...WEND
REPEAT...UNTIL

9-21、For...Next（指定した回数繰り返し実行する）

Ｆｏｒ Count = start TO end {ＳＴＥＰ {-} Inc}
{Body}
ＮＥＸＴ {Count}

Countで指定した変数の値がstartからendの値になるまで{Body}部分を繰り返し実行します。
Countの値がendで指定した値に達するとFor文は終了し、Next以降の命令を実行します。

Countには、変数を指定します。この変数はループの回数を管理することからループ変数と呼ばれます。

ループが1回まわる毎に（{Body}が1回実行される毎に）Count変数の値は{step {-} Inc}で指定した値ずつ変化します。なお、この{step {-} Inc}を省略すると、1ずつインクリメント（増加）されます。
ループ計算は16ビットで行われます。（Count変数にはbyte型又はword型が使えます。）

Count	byte型又はword型変数を指定します。ループした回数がこの変数に代入されます。
start	変数又は定数です。開始値を指定します。
end	変数又は定数です。最終値を指定します。
Inc	省略可能です。カウンターの変化値を指定します。-1を指定するとデクリメントします。 省略するとループカウンタは１ずつ加算されます。

例： RB0を0.5秒間隔で10回点滅させるプログラム

```
i      var    byte
i = 0

For  i  =  1  to  10

       High   PORTB.0
       Pause  500
       Low    PORTB.0
       Pause  500

Next i
```

※ 例えば、 "For i = 10 to 1 step -1" と記述しても同様に10回点滅後停止します。

ループカウンタは使用前に、必ず初期化(i = 0など)してください。
ループカウンタは、ループの回数を記憶していますので、ループ中にループカウンタを使用してプログラムを作ることも可能です。

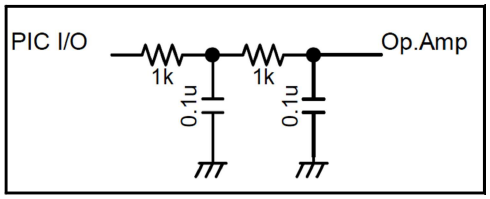
※ループ中にループから抜けたい場合には、Goto命令を使用して任意のラベルにジャンプする方法を使用します。また、ループ中でもGosub命令のサブルーチンが使用できます。

※割り込みを有効にしている場合、ループ中に割り込みが発生するとループを抜けて直ちに割り込みルーチンが実行されます。

9-22、FREQOUT（指定した周波数のパルスを出力）

ＦＲＥＱＯＵＴ Pin、Onms、Frequency1{、Frequency2}

指定した周波数のパルスを最大2波まで合成して指定したPinからOnmsミリ秒間出力します。
指定できる周波数は0Hz～32767Hzまでです。FREQOUT命令はPWMを使用して任意の周波数を作り出します。PICにスピーカー等を接続して音を出すには、下記のようなフィルタ回路を作ります。



Pin	出力するピン	
Onms	パルスの発生時間　単位はミリ秒	
Frequency1	出力周波数　　単位はHz	0Hz～32767Hzまで。
Frequency2	第二周波数　　単位はHz	0Hz～32767Hzまで。省略可能です。

例：　時報音を出力するプログラム

```
i      var    byte
i = 0

For  i  =  1  to  3
       FREQOUT 0,100,400
       Pause 900
Next i

FREQOUT 0,2000,880

Stop
```

9-23、GOSUB（サブルーチンをコールします）

ＧＯＳＵＢ Label

Labelで指定したラベルのサブルーチンへジャンプします。
GOTO命令と異なりジャンプ先からはRETURN命令でジャンプを開始した位置に戻ります。サブルーチンをコールしたい時に使います。

ジャンプ先のサブルーチン内で更に何回か次のサブルーチンを呼び出すこと（ネストといいます）もできますが、その際にはスタックの深さに制限があります。

・12ビット及び14ビットコアデバイス	4レベル
・14ビットエンハンスドコアのPIC17シリーズ	12レベル
・PIC18シリーズ	26レベル

※GOSUB命令は、FOR文で繰り返し実行している場合にも使用できます。

9-24、GOTO（指定したラベルに処理を飛ばす）

ＧＯＴＯ Label

Labelで指定したラベルへジャンプします。
ジャンプした後、元のプログラム位置へ戻ることはできません。処理はジャンプ先に行ったまま継続されます。

処理を別のラベル位置に飛ばしたい時に使用します。またGOTO命令は永久ループ（無限にループを繰り返す場合）を作る際にも使用できます。

例：　0.5秒間隔でRB0が点滅を繰り返すプログラム

```
main:
       High PORTB.0
       Pause 500
       Low PORTB.0
       Pause 500

       Goto main
```

※永久ループの場合にはDO...LOOP命令のほうがシンプルに記述できます。

9-25、HIGH（指定したピンをHighレベルにする）

ＨＩＧＨ Pin

指定したのピンをHighレベル(Vccの電圧レベル)にします。
指定された該当ビットの状態はLowレベルからHighレベルになります。
HIGH命令によって指定されたビットは自動的に出力方向に設定されます。

High PORTB.0

上記の例では、PORTBのビット0をHighにします。

```
LED    var      PORTB.0
High   LED
```

上記の例では、PORTB.0を文字列LEDにアサインしています。"High LED"でPORTB.0がHighレベルになります。

ポートを点滅させたい場合は、下記のようにします。

```
Loop:  HIGH   PORTB.0          'PORTBのBIT0をHにする
       PAUSE  1000             '1000ミリ秒＝1秒待機する
       LOW    PORTB.0          'PORTBのBIT0をLにする
       PAUSE  1000
       GOTO   Loop             'ラベルLoopに戻り繰り返す
```

★ポートの各ビットの状態をまとめて記述する場合

PBPではポートの各ビットの状態を0と1を利用してまとめて記述することもできます。これは、PBPでは、基本的に「レジスタ名=値」で、レジスタに直接、値を代入できるからです。
例えば、PORTAのすべてのビットを1にしたい場合は次のように記述します。

```
TRISA =        0                          'PORTAをすべて出力に設定
PORTA=         %11111111
```

PORTAの状態を01010101としたい場合は同様にして PORTA = %01010101と記述します。
なお、ポートに直接、値を代入して出力を行う場合には、必ずTRISxレジスタの設定を記述して該当ピンを出力に設定した状態にします。

※TRISxレジスタは、入出力方向を設定するレジスタです。該当のビットが1の場合そのピンは入力ピンに、該当ビットが0の場合には、そのピンは出力ピンとして設定されます。詳しくはPICマイコンのデータシートをご覧ください。

まとめてポートを制御したい場合にはHIGH命令を利用するよりも効率が良く簡単です。但し、すべてのポートを同時に制御する際にはPICマイコンの定格電流を越えないよう設計に注意してください。

9-26、HPWM（ハードウエアPWMを使用してPWMを出力）

ＨＰＷＭ Channel1、 Dutycycle、 Frequency

ハードウエアPWM機能(CCP)を搭載したPICの機能を使用したPWMを生成します。ハードウエアPWMは、別の命令実行中であってもバックグラウンドで動作し連続して信号を出力することができます。

PWMは「Pulse Width Modulation」の略でパルス幅変調とも呼ばれます。デジタルの2値（HとL）によりパルス幅を決めることでアナログの状態値を表現する方式です。周期を一定にしてパルスのHとLの割合（デューティー比）を変えることで出力電圧を制御するものです。RCサーボにも応用されます。

Channle1は、PWMで使用するチャンネルを指定します。使用するPICによって1～3のチャンネルがあります。例えば16F887ではCH1がCCP1でRC2に、CH2がCCP2でRC1にアサインされています。ピン名とPWMのチャンネル名については使用するPICマイコンのデータシートを参照ください。

DutycycleはH-Lレベルの周期を0～255の範囲で指定します。0を指定するとOFF（永久にLの状態）になり、255を指定するとON（永久にH）になります。中間値127は50%デューティーということになります。

FrequencyはPWM信号の周波数を指定します。2チャンネルを持つデバイスの場合には、両方のチャンネルとも同じ周波数を設定しなければなりません。PICマイコンの動作周波数によって下限の周波数が決まります。上限の周波数はいずれの動作周波数であっても32767Hzとなります。下限周波数は下記の通りです。
4MHz→245Hz, 8MHz→489Hz, 10MHz→611Hz, 12MHz→733Hz, 16MHz→977Hz
20MHz→1221Hz, 24MHz→1465Hz, 33MHz→2015Hz, 40MHz→2442Hz

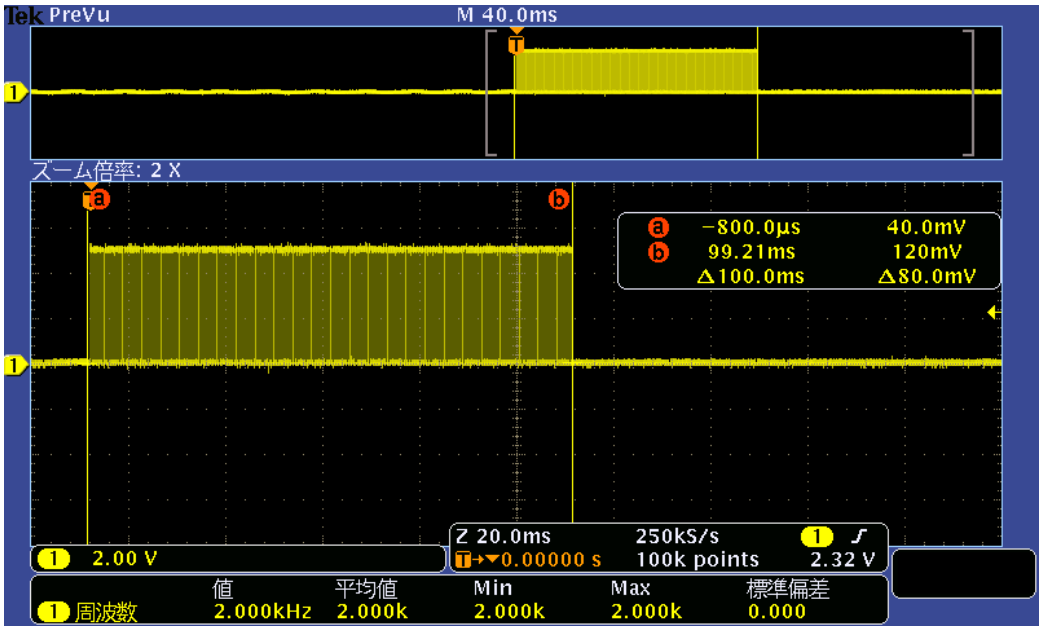
18F452にてHPWMを使用する場合には、DEFINEにてピンを定義します。

```
DEFINE CCP1_REG   PORTC
DEFINE CCP1_BIT    2
```

例：1KHz、デューティー比50%の方形波を出力
HPWM 1、127、1000

HPWM命令は1度実行されると、CCPを制御するレジスタをPWMモードに自動的に設定します。そのため、HPWM命令の後でCCPピンを通常のデジタルI/Oピンとして使用したい場合には、手動でCCPを制御するレジスタを操作してPWMモードをOFFに設定する必要があります。

一度命令を実行すると、基本的には指定したデューティー比、周波数にてパルスを出力し続けます。例えば一定時間でパルスを停止したい場合には、PAUSE命令等でパルスを出力したい時間分処理を停止し、その後、デバイスによりますが例えば16F887の場合には、ECCPASレジスタのビット7に1を代入することでパルス出力を停止します。（ECCPAS.7 ＝ 1）　下図は2KHzのパルスを100m秒出力した場合の波形です。



9-27、HSERIN(2)（ハードウエア非同期式シリアル通信受信）

ＨＳＥＲＩＮ {ParituLabel、}{Timeout、Label、}[Item(、Item...)]

ＨＳＥＲＩＮ２ {ParituLabel、}{Timeout、Label、}[Item(、Item...)]

ハードウエアUSART搭載のデバイスにおいてハードウエアUSARTを利用して非同期式シリアル通信(UART)でデータを受信する命令です。RS232Cと同様データはLSBファーストです。

HSERINはハードウエアUSART機能が1つのデバイス又は、ハードウエアUSARTが2つあるデバイスの1つ目のシリアルポートにアクセスする命令です。
HSERIN2は、ハードウエアUSARTが2つあるデバイスの2つ目のハードウエアシリアルポートにアクセスする命令です。

省略可能なParityLabelにラベル名を記述するとはパリティ使用時にエラーが検出された場合、指定したラベルにジャンプします。省略可能なTimeout、Labelには受信可能なデータが指定したTimeout秒(単位はミリ秒単位)こなかった場合にはLabelで指定したラベルにジャンプします。

[ ]内にbyte型変数を記述し、変数に受信したデータを格納します。また複数のデータ列を受信する場合にはbyte型配列を使います。

※ハードウエアUSARTの機能を内蔵しないPICを使う場合や、やむを得ずハードウエアUSARTのTX,RX ピンに割り当てられているピンを別のことで使用しなければならない場合には、ソフトウエアでUART通信を行うSERIN2,SEROUT2命令を使用します。

HSERIN(2)命令は通常のルーチンに記述してUART通信を行うことができますが、シリアル通信受信割込を使用した割込受信に対応しています。(割込処理はPIC側で行い、割込ルーチン先で本命令でデータを受信します。) 割込を使わずに、通常ルーチンで本命例を使う場合にはUART通信で受信を待機する場所に本命令を記述します。通信速度は、プログラムの先頭部分で"DEFINE HSER_BAUD"にてbps単位で定義します。

```
DEFINE OSC 16
DEFINE HSER_BAUD 9600

rec var byte

main:
    HSERIN [rec]
    HSEROUT [rec]
    goto main
    END
```

上の例はボーレート9600bpsで受信したデータをそのままUART通信で送り返すプログラムです。
1バイト単位でデータを受信する場合にはbyte型変数を使います。複数バイトを連続して受信したい場合にはbyte型配列を使い後述のSTR Array修飾子を使います。
なお、HSERIN命令は受信が完了するまでその場で待機します。そのためUART通信でデータを受信できないと、永久にその場でプログラムが止まります。それを避ける方法として省略可能なタイムアウト時間を指定することができます。

なお、上例の場合には受信を待機して受信後次の処理をする通常のルーチンとしてHSERIN命令を使いましたが、例えば定常時は別の処理を実行しながら、UART通信のデータが到来したら直ちに受信処理を開始するようにプログラムを作ることもできます。そのような場合にはシリアル通信割込を使います。

■ハードウエアUSARTに関連するレジスタとHSERIN(2)の関係

この命令を割込で使用するには、ハードウエアUSARTに関連する各種レジスタの設定が必要な場合があります。なお本書ではRS232Cに代表される非同期シリアル通信について解説します。
ハードウエアUSARTではPIR1レジスタ、PIE1レジスタ、TXSTAレジスタ・RCSTAレジスタを操作します。各レジスタの内容を下記に示します。
最初にTXSTAレジスタとRCSTAレジスタを設定します。両レジスタはシリアル通信における送信及び受信のフォーマットなどを設定します。通常RS232Cのような非同期通信の場合で、
"データ長8ビット長・パリティなし・1ストップビット"の場合にはTXSTAレジスタは$20(16進数20)RCSTAレジスタは$90に設定します。データ長を9ビット長にする場合などは別途データシートをご覧の上設定してください。
この設定にて、ハードウエアUSARTによる非同期通信が有効になります。PBPでは両レジスタの値の設定はDEFINEで行います。

```
DEFINE        HSER_TXSTA   20h
DEFINE        HSER_RCSTA   90h
```

続いてボーレートの設定もDEFINEで行います。

```
DEFINE        HSER_BAUD    9600
```

ボーレートは1200bps～115.2kbps（発振子が4MHzの場合には1200bps～50kbpsまで）まで設定できますが、通信速度はPICマイコンのクロック周波数により上限が決まります。19.2kbps以上の場合には、動作周波数を20MHzにすることをお奨めします。

以下ハードウエアUSARTの仕組みを説明します。割込受信をする場合には必要な知識となりますのでご一読ください。HSERIN(2)命令だけを使用する（割込を使用しない）場合にはHSERIN(2)命令によって以下の仕組みや手順は自動的実行されます。

受信したデータはRCREGレジスタに格納されます(RCREGレジスタに空きがある場合)。RCREGレジスタは深さ2のFIFOバッファ＝すなわち2バイトを格納できるレジスタです。
受信が完了するとPIR1レジスタのビット5のRCIFビットが1にセットされます。この時INTCONレジスタやPIE1レジスタでUSART受信割込を有効にしておくと、割込が発生します。
割込発生の可否はPIE1レジスタのビット5、RCIEレジスタで設定できます。
先ほど記述した通り、RCREGレジスタは2バイトを受信できますが、3バイト目のデータのストップビットを検出した時点でRCREGレジスタはオーバーフローし、OERRビットが1にセットされます。
このOERRビットが1にセットされるとそれ以降受信ができなくなります。PBPでは自動的にこのビットを0にクリアして再度受信可能に設定するDEFINE文があります。またOERRビットが1に設定されると同時にCRENビットが0にセットされ継続受信を無効に設定してしまいます。PBPにおいてOERRビットを自動的にクリアして再度受信を可能にする場合には次のように記述します。

```
プログラムの先頭で              →  DEFINE   HSER_CLROERR   1
データ受信後再度受信する場合    →  RCSTA.4 = 0
                                   RCSTA.4 = 1
```

RCSTA.4すなわちCRENビットは一度0にセットしてから1にセットしないと設定できない仕組みになっています。但し上記のようにオーバーフローを無視するとデータの取りこぼしが発生する場合があります。これはOERRビットをクリアすることで、RCREGレジスタにデータがあってもどんどん新しいデータに上書きされてしまうためです。9600bpsで通信した場合次のデータが来るまでの時間が1ms程度ありますので、受信したら次々と変数に格納していく必要があります。

HSERIN(2)命令では複数バイトのデータを受信する場合にはbyte型の配列を使用します。
この時は"STR修飾子"を使用します。STR修飾子は "STR ArrayVar\n{\c}" の書式で指定したバイト数nバイトを受信して配列0～nまで変数に格納します。STR修飾子を使用すると通常は指定したn

バイトのデータを受信するまでは、HSERIN(2)命令が処理を譲りませんが、オプションの{\c}を指定しておくと値cを受信すると直ちに受信動作を終了します。

ArrayVar　var　byte[10]
HSERIN [STR ArrayVar \5 \$60]

上記のように記述すると配列ArryVarには[0]～[4]までの4バイトが1バイトずつ受信されます。
$60（16進数60）を受信しなければ5バイトを格納後、次の行に処理が移動します。
$60を受信すると直ちに次の行の処理に移動し、受信できなかった配列には0が代入されます。

■シリアル通信受信割込とHSERIN(2)の使い方

シリアルデータが到来した時に割込を発生させ、割込先ルーチンでデータを受信する場合にもHSERIN命令を使います。PBPではOn Interrupt Goto命令と併せて割込を行います。各種レジスタは一般的に次のような設定になります。(設定は一例であり動作の内容等により変わります。)

INTCON =	%11000000	割込機能と周辺割込を有効にします
PIE1 =	%00100000	受信割込のみ有効にします（送信割込は無効）
PIR1.5 =	0	USART受信割込フラグビット（バッファは空）※

※PIR1レジスタのビット5は、USART通信でデータを受信すると1になるフラグビットですが、データをバッファから読み出すと自動的にクリアー(0)されます。

下は一般的なシリアル通信受信割込を使ったシリアル通信受信例です。

```
DEFINE  HSER_RCSTA      90h
DEFINE  HSER_TXSTA      20h
DEFINE  HSER_BAUD       9600                    'ボーレート 9600bps
DEFINE  HSER_CLROERR    1                       'OverRunErr を自動的にクリア

On Interrupt goto       SERINT

INTCON = %11000000                              'INTCON レジスタの設定
PIE1 = %00100000                                'PIE1 レジスタの設定(受信割込有効)
PIR1.5 = 0                                      'USART 受信バッファフラグ初期化
rec_ser     var     byte[16]                    'rec_ser 配列を 16 バイト分作成

main:   High    PORTD.1                         ┐
        Pause   200                             ├ メインルーチン(通常時はここを実行中)
        Goto    main                            ┘

Disable                                         '割込ルーチン開始位置
SERINT:
        High PORTD.2                            '割込通知用 LED 点灯
        Hserin [STR rec_ser ¥4 ¥$60]            '$60 を受信するまで 4 バイト分受信

        Lcdout $fe,1,   dec2 rec_ser[0] , "-" , dec2 rec_ser[1]         '1 行目
        Lcdout $fe,$c0, dec2 rec_ser[2] , "-" , dec2 rec_ser[3]        '2 行目
        INTCON = %11000000                      'レジスタ初期化
        PIR1.5 = 0                              'USART バッファフラグ初期化　※
        RCSTA.4 = 0                             '継続受信を設定をリセット
        RCSTA.4 = 1                             '            〃
        resume                                  'メインルーチンに戻る
Enable                                          '割込ルーチン終了
```

HSERIN命令、HSEROUT命令ではmodifierが使用できます。modifierは変数の値などを目的の形に修正したり、配列の場合には配列の制御をしたりするものです。
上記例では、LCDOUT命令において、DEC2を使用して、変数の値を10進数2桁で表示するよう修正しています。

※PIR1レジスタのビット5は、USART通信でデータを受信すると1になるフラグビットですが、データをバッファから読み出すと自動的にクリアー(0)されます。よって、プログラム中で明示的に0を代入しなくてもご使用できます。

modifier	内　容
BIN{1～16}	受信したデータを2進数で変数に格納します。
DEC{1～5}	受信したデータを10進数で変数に格納します。
HEX{1～4}	受信したデータを16進数で変数に格納します。
SKIP n	nで指定したバイト数分受信データをスキップします
STR ArrayVar\n{\c}	ArrayVar配列にnバイト分受信して格納します。 値cを受信すると受信作業を直ちに終了して次の行を実行します。

配列を使って複数バイトのデータを受信する場合にはSTR修飾子を使います。配列はbyte型配列を指定します。\nには受信するデータの総バイト数を10進数の値で指定します。省略可能な{\c}には、このデータを受信したら即、受信処理を完了する、というデリミタを指定します。
下の例は、総バイト数128バイトのデータを受信するまで受信処理を継続します。但しデータの中にキャリッジリターン($0D)が含まれていたら処理をすぐに終了する場合です。
通信速度は9600bpsです。配列recは要素数128のbyte型配列です。

```
DEFINE  HSER_BAUD       9600

init:
    rec var byte[128]

main:
    hserin [STR rec¥128¥$0D]
    hserout [Str rec]

    goto main
    END
```

$0D(キャリッジリターン)を受信しなければ128バイト受信するまでデータを受信し続けます。データの中に$0Dがあると、すぐにデータ受信を終了して、受信したデータをシリアル通信で出力します。
センサなどでコマンドの終端がキャリッジリターンになっているものなどの受信に便利です。

※プログラム中の"\"マークは、英文字環境ではバックスラッシュ（\）として表示されますが同じものです。

HSERIN命令は、ハードウエアシリアル通信機能の搭載されたデバイスでのみ使用できます。
シリアルUSARTの搭載されていないデバイスや、ハードウエアUSARTを使用したくない場合にはSERIN2命令を使用します。

9-28、HSEROUT(2)（ハードウエア非同期式シリアル通信送信）

ＨＳＥＲＯＵＴ［Item{、Item...]

ＨＳＥＲＯＵＴ２［Item(、Item...)]

ハードウエアUSART搭載のデバイスにおいてハードウエアUSARTを利用して非同期式シリアル通信(UART)でデータを送信する命令です。RS232Cと同様データはLSBファーストです。

HSEROUTはハードウエアUSART機能が1つのデバイス又は、ハードウエアUSARTが2つあるデバイスの1つ目のシリアルポートにアクセスする命令です。
HSEROUT2は、ハードウエアUSARTが2つあるデバイスの2つ目のハードウエアシリアルポートにアクセスする命令です。

```
DEFINE OSC 16
DEFINE HSER_BAUD 9600
main:
    HSEROUT ["HELLO"]
    pause 500
    goto main
```

上の例は9600bpsで0.5秒間隔で文字列"HELLO"を繰り返し送信する例です。
通信速度は、プログラムの先頭部分で"DEFINE HSER_BAUD"にてbps単位で定義します。
シリアル通信のデータフォーマットは上記設定でデータ長8ビット長・パリティなし。1ストップビットです。

修飾子を使うとデータを色々加工して送信することができます。

modifier	内　容
BIN{1～16}	データを指定したビット数で2進数で送信します
DEC{1～5}	データを指定した桁数で10進数で送信します
HEX{1～4}	データを指定した桁数で16進数で送信します
REP c\n	cで指定した値をn回繰り返して送信します
STR ArrayVar{\n}	ArrayVar配列の内容をnバイト分送信します {\n}を省略すると配列の内容すべてを送信します

STR修飾子を使った例がHSERIN命令のページに記載ありますのでご参照ください。

9-29、I2CREAD（I2C通信でデータを読み込み）

Ｉ２ＣＲＥＡＤ Datapin、Clockpin、Control、{Address、}[Ver{、Var...}]{Label}

I2CREAD命令はI2C通信で、データ（1バイトもしくは連続したデータ）をI2Cデバイスから読み込みます。
I2CインターフェイスのEEPROM（例：24LC64、24LC256等）などPIC外部のEEPROMからデータを読み込むなどの操作ができます。なお、本命令はソフトウエア的にI2C通信を実行しています。

DatapinではSDAラインを、ClockpinではSCLラインを指定します。読み込まれたデータはVarで指定した変数に格納されます。
読み込まれる値は、1バイト単位ですので変数varはbyte型変数を用います。

通信相手デバイスのスレーブアドレスはControl値として記述します。

省略可能なAddressは、EEPROMからデータを読み込む場合に使う引数で、EEPROMの読み出しアドレスを指定できます。I2C通信センサー等と接続する場合には指定しません。

■Controlの値の算出について
*Control*の値はI2Cデバイスのスレーブアドレスを指定します。
センサー等のI2Cデバイスの場合にはここにスレーブアドレスを記述します。
I2CタイプのEEPROMの場合には、上位4ビットが1010となっており下位4ビットはデバイスにより主に下表のようになっています。

デバイス	サイズ	コントロール値	アドレスサイズ
24LC01B	128B	%1010xxx0	1バイト
24LC02B	256B	%1010xxx0	1バイト
24LC16B	2KB	%1010bbb0	1バイト
24LC32B	4KB	%1010ddd0	2バイト
24LC64	8KB	%1010ddd0	2バイト
24LC256	32KB	%1010ddd0	2バイト

※bbb ＝ ブロックセレクトビット、ddd ＝ デバイスセレクトビット、xxx ＝ 任意（0にします）

上表のdはデバイス選択ビットです。EEPROMデバイスのA0、A1、A2ピンの物理的な状態と対応させます。A0、A1、A2をすべてGNDに接続している場合には、dddの部分を000とします。

アドレスサイズはメモリーアドレスのサイズで24LC32以降のデバイスではアドレスサイズは2バイトに拡張されています。よって、アドレスを指定する変数は16ビットのword型変数を使用することになります。

なお、コントロール値はI2C通信ではビット0の部分をRead（読み込み）で1、Write（書き込み）で0にするという決まりがありますが、I2CREAD命令ではここは0に固定されます。

省略可能な*Label*部分にLabel名を記述した場合、I2Cデバイスからのデータが受け取れなかった時、そのLable先にジャンプします。エラー用に利用できます。省略してもかまいません。
I2Cデバイスから読み込んだ値を格納する変数Varは、byte型の変数を使用するようにします。

プログラム例（24LC256からデータを読み込む場合）

```
B2     var     byte
addr   var     word          'アドレス指定用変数は word 型で宣言（ 2 バイトサイズ）
cont   con     %10100000     'control 用の定数を宣言（スレーブアドレス）
addr   =       10            'アドレス指定用変数にアドレスの値を代入（ここでは例として 10 ）

I2CREAD  PORTC.4 、 PORTC.3 、 cont 、 addr 、[B2]
```

■アドレス指定の方法（重要）
24LC64や、24LC256は先の表でも分かるとおりアドレスサイズが2バイトです。アドレスサイズが2バイトのデバイスの場合には、必ずword型の変数をアドレス指定用に使います。定数やbyte型の変数は使用することはできません。（定数も使用できないことに注意してください。）
例えば24LC512からデータを読み込む下記のようなプログラムを記述した場合、コンパイル上はエラーは表示されませんが（書式上は間違いはありません）、動作に問題が発生します。

```
I2CREAD  PORTC.4 、 PORTC.3 、 cont 、 10 、［B2］          '間違った記述例
```

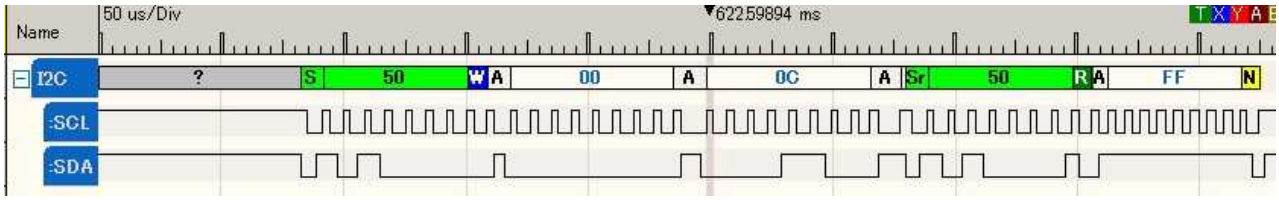
上記例では、アドレス指定を定数で指定しています。これは誤った記述です。アドレスを指定する場合には、必ず変数を準備してそこに値を代入してアドレスを指定するようにします。

■ストップコンディションの発行
I2C通信規格では、通信終了時にストップコンディションの発行が必要ですが、PBPでは特にそれを記述する必要はありません。[var]で指定された変数の分だけデータを受信し、それが終了した時点でPBPがストップコンディションを発行しています。ユーザーが明示的にストップコンディションの発行について記述する必要はありません。
多くのEEPROMの場合には、シーケンシャルリードモード（連続読み込みモード）を搭載していますので、読み込みを開始する先頭アドレスを指定するだけで、あとは自動的に1ずつインクリメント（加算）されたアドレスから順番にデータが変数に読み込まれます。

■通信速度について
I2C通信規格は、400kHzの高速モードと、100kHzの低速モードがあります。PBPではPICマイコンのクロック周波数が8MHz以上の時は100kHzの通信速度となります。400kHzモードにするには、PICマイコンのクロック周波数を20MHz以上にする必要があります。
例えば20MHzのクロック周波数で動作しているPICマイコンから古いデバイス等へのアクセスが必要で、明示的に100kHzにスローダウンさせたい場合には、次のDEFINEを記述します。

DEFINE I2C_SLOW 1

【参考データ】ロジックアナライザで見たI2CREAD命令実行時の波形
下記にI2CREAD命令で24LC64から1バイトのデータ(データの内容は255)を受信した際のロジックアナライザでの波形データを参考に示します。（値はすべて16進数で表記されています）

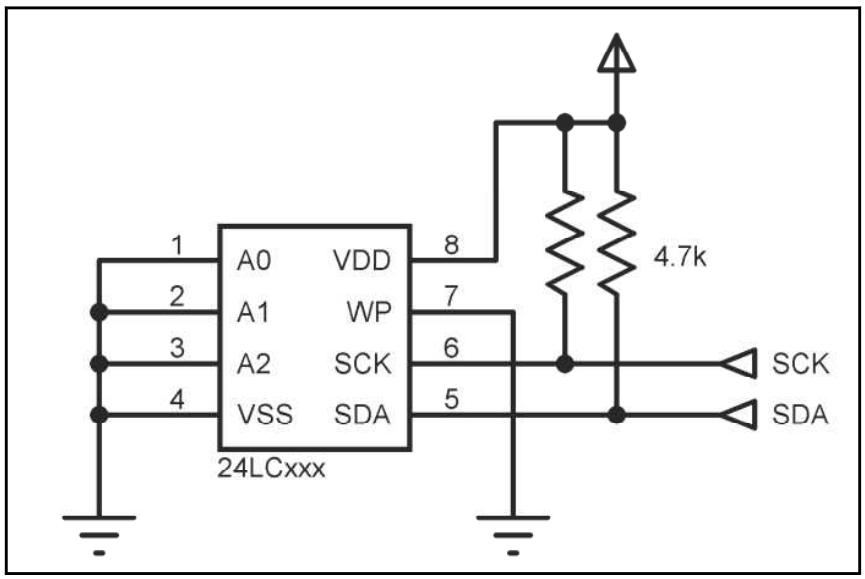


SDAがHでSCLがLになった所がスタートコンディションです。先頭部はスレーブアドレスと番地をWモードで書き込みます。先頭の0x50は、スレーブアドレスです。スレーブアドレスは書き込みますのでWモードですので、8ビット目は0になります。読み出す番地は12番地です。0x0C(12)を送信します。
続いて、データを読み込みますので、スレーブアドレスを送信後に今度はReadモードですので、ビット目を1にしてRモードとして送信します。戻り値として、0xFFが返ります。

PBPでは、コントロール値（スレーブアドレス）と、アドレスの番地を指定するだけでPBPが自動的にWモードとRモードの処理をして、I2CREAD命令であればデータを読み込むようになっています。一般的にI2C通信をする場合には、スレーブアドレスに対して、読み込みならば1を、書き込みならば0を足して・・・というモードの設定をプログラムに書きますがPBPの場合には自動的に行いますので、ユーザーはその手順を意識する必要がありません。

I2Cデバイスと通信する場合には、コントロール値にスレーブアドレスを記述することになります。

■ハードウエア例
下図は、24LC64等のEEPROMを接続する場合の参考回路図です。本図ではデバイスセレクトビットは0になっています。I2C通信では信号線は4.7KΩ程度の抵抗でプルアップする必要があります。



9-30、I2CWRITE（I2C通信でデータを書き込み）

Ｉ２ＣＷＲＩＴＥ Datapin、Clockpin、Control、{Address、}[Value{、Value...}]{Label}

I2CWRITE命令はI2C通信で、I2C規格のセンサーやEEPROMに変数の値を書き込みます。

Clockpinピンで指定したピンからクロック信号が出力され、そのクロックに同期してDatapinで指定したデータピンからデータが出力されます。DatapinはSDAピンと、ClockpinはSCLピンと接続します。

Controlの値は、I2Cデバイスのスレーブアドレスを指定します。
EEPROMの場合の設定値については、I2CREAD命令のページに表がありますのでそちらをご参照下さい。

省略可能なAddressは、EEPROMにデータを書き込む場合に使う引数で、EEPROMの書き込み先アドレスを指定できます。I2C通信センサー等と接続する場合には指定しません。

Controlの値及びAddress値についてはI2CREAD命令の項をご参照ください。

■ページモードでの書き込み
多くのEEPROMでは1バイトずつデータを書き込むバイトモードと、複数のデータをまとめて書き込むページモードがあります。ページモードの最大値はデバイスによって決まっていて、例えば24LC512では、1ページあたり最大128バイトまでと決まっています。すなわち、1回の命令で最大128バイトまでデータを書き込めると言うことになります。その場合、1バイトずつ書き込むわけではないので、書き込みにかかる時間を減らすことができます。またEEPROMのメモリー寿命を延ばすことができます。
PBPのI2C命令では、ストップコンディションの発行はI2C命令が終わった際に自動的にPBPによって発行されます。よって、ページモードで複数のデータを1つの命令で書き込んだ場合には、その複数のデータを一度に書き込んで最後にストップコンディションが発行されますので、1バイトずつ書き込む場合と違い、待ち時間を減らすことができます。ページモードでもバイトモードでも本命令の後には最低でも10ミリ秒の遅延を入れるようにします。

ページモードで書き込む場合には、byte型配列を使用します。指定できる要素数の最大値はそのデバイスの仕様で決められたページモードの1ページ辺りの最大バイト数までとなります。例えば24LC512ならば128要素が最大となります。アドレスは1バイトずつインクリメントされます。Addressで指定するアドレスは、先頭アドレスということになります。
配列を使う場合には、STR修飾子を使用し、書き込むバイト数を指定することもできます。

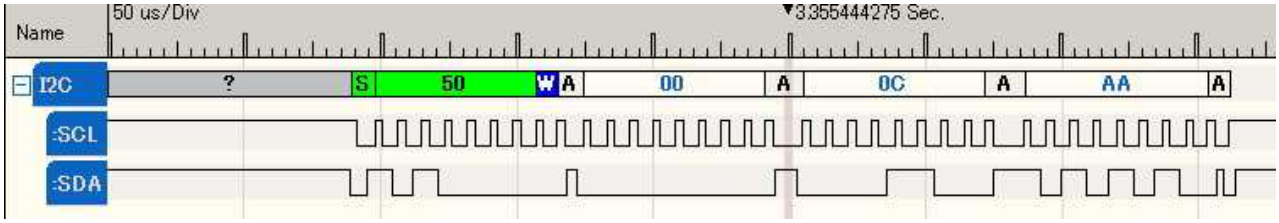
```
write_data var byte[8]
addr var word
addr = 0

I2CWRITE SDA,SCL ,%10100000 , addr , [STR write_data¥8]
```

※英語版マニュアルにはword型配列を使う場合も記述がありますが、お奨めできません。

■通信速度について
I2C通信規格は、400kHzの高速モードと、100kHzの低速モードがあります。PBPではPICマイコンのクロック周波数が8MHz以上の時は100kHzの通信速度となります。400kHzモードにするには、PICマイコンのクロック周波数を20MHz以上にする必要があります。
例えば20MHzのクロック周波数で動作しているPICマイコンから古いデバイス等へのアクセスが必要で、明示的に100kHzにスローダウンさせたい場合には、次のDEFINEを記述します。

DEFINE I2C_SLOW 1

【参考データ】ロジックアナライザで見たI2CWRITE命令実行時の波形
下図は、I2CWRITE命令で24LC64のEEPROMの12番地に値0xAAを書き込んだ時のロジックアナライザのデータです。



最初にスタートコンディションがあり、その次にコントロール値（スレーブアドレス）を書き込み、アドレス0x0C(12)を送信後、書き込むデータ0xAAを送信しています。

※スレーブアドレスには通常、読み込みならばビット0に1を、書き込みなら0を加算しますが、PBPのI2C命令ではスレーブアドレスの書き込みやデータの読み込み等を1つの命令ですべて行いますので、プログラマがW/Rの方向をスレーブアドレスに加算するなどの手続きは必要ありません。

9-31、IF...THEN（条件式による処理の分岐）

```
ＩＦ Comp {ＡＮＤ／ＯＲ Comp...} ＴＨＥＮ Label                    形式Ａ
```

```
ＩＦ Comp {ＡＮＤ／ＯＲ Comp...} ＴＨＥＮ                          形式Ｂ
      Statement.....
ＥＮＤＩＦ
```

```
ＩＦ Comp {ＡＮＤ／ＯＲ Comp...} ＴＨＥＮ                          形式Ｃ
      Statement1.....
ＥＬＳＥ
      Starement2....
ＥＮＤＩＦ
```

IF...THEN命令には3つの形式があります。1つは評価式を評価してすぐに指定ラベルに飛ぶ形式（形式A)、評価式が真の時"ENDIF"までの間のステートメントを実行する形式(形式B)、評価式が真の場合と偽の場合両方のステートメントを記述する形式(形式C)があります。

形式Ａは
Compの内容が「真」とき、Lableへジャンプします。

形式Ｂは
Compの内容が真のとき、Statementの内容を実行します。Compの内容が偽の場合には、ステートメントは実行されず、"ENDIF"の次の行から実行されます。

形式Ｃは
Compの内容が真のとき、Statement1の内容を実行します。Compの内容が偽のときはStatement2の内容を実行します。　（Compの内容に応じてStatement1又はStatement2のどちらかが必ず実行されます）

Comp	評価式です。比較、評価する式やポートの状態などを指定します。
Lebel	Compを調査してその結果が真の場合はここで指定したラベルにジャンプします。
Satement	実行する処理です。

Compにはあらゆる評価式が記述できます。例えば変数内の値の大きさを比較する場合には、比較演算子や等号を使用します。

```
If age > 50 Then Older          '変数ageの値が51以上の時に真になります
If time >= 20 Then Timeover     '変数timeの値が20以上の時に真になります
```

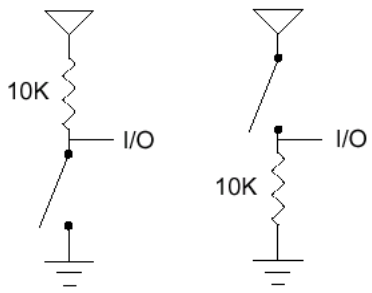
Compにピンの状態を記述すると、ピンの状態に応じた分岐処理ができます。

```
If PORTB.2 = 1 Then Pushed      'PORTBのBit2(RB2)がHighの時に真になります
If PORTB.0 = 0 Then main        'PORTBのBit0(RB0)がLowの時に真になります
```

この方法でピンの状態を評価することで押しボタンスイッチの判定やピン状態のH/Lで処理を分岐できます。IF文は評価式が真でない場合、次の行に処理が飛ばされてしまうのでボタン入力を待機したりピンの状態を繰り返し監視したい場合にはループ構造にしてピン状態の判定を繰り返します。

```
label:  IF   PORTB.2 = 1   THEN   main     'RB2がHレベルでラベルmainへジャンプ
        IF   PORTB.3 = 0   THEN   term     'RB3がLレベルでラベルtermへジャンプ
        GOTO label                         'labelへ戻って繰り返し待機
```

GOTO文で無限ループにすることで、入力状態を繰り返し監視するようにします。
先の例では、該当ピンがHになった時とLになった時で処理を指定したラベルにジャンプさせていました。
PICマイコンのピンに押しボタンスイッチを接続する場合には、押しボタンスイッチを押したときに該当ピンがHレベルになるか、Lレベルになるかで、ボタンの接続方法が変わります。下記に例を示します。



Active Low　　Active High

・左側；アクティブロー　　スイッチがONの時にI/OはLレベル
・右側：アクティブハイ　　スイッチがONの時にI/OはHレベル

プルアップ抵抗又はプルダウン抵抗は4.7kΩ～10kΩ程度を選定します。

通常は外部からのノイズによる誤作動を防止するため、スイッチの場合にはアクティブローを使用することが多いです。

ピンに機械的なスイッチを取り付けた場合、チャタリング（スイッチが押されたり離れたりする際に発生するノイズ）が発生します。IF文ではチャタリングのことは全く考えられていませんので、対処はプログラムによって施す必要があります。また1つのスイッチに対してIF文を繰り返す場合は、「押しすぎ（長押し）」の対策も必要です。例えば次のような「スイッチを押す毎にカウントが増えるプログラム」があります。

```
main;    If  PORTB.2 = 0  then  cnt_up
         Goto main

cnt_up:  cnt = cnt + 1
         Goto main
```

※cntは変数

上記のプログラムはチャタリングや長押しの対策をしていないため、スイッチを押してRB2がLレベルになった瞬間、"cnt_up"ルーチンが何度も繰り返し実行されてしまい、カウントも意図しない値になってしまいます。わずか1秒スイッチを押しただけでカウント値は200位になるかもしれません。接続されている押しボタンスイッチが押された回数を精密にカウントしたい場合には一度ボタンから手を離した（該当ポートの論理が変わった）ということを検出するプログラムを作る必要があります。

その他、IF文では条件式CompにANDやORなどの論理演算子を使用することもできます。また、任意のレジスタ(SFR)の値を条件時として使用して分岐処理を行うこともできます。

例：RB3とRB4が同時にHighレベルの時だけLCDに文字を表示する

```
main:
        If PORTB.3 = 1 AND PORTB.4 = 1 then

         Lcdout $fe, 1, "RB3&RB4 are High"
         pause 500

        Endif

        Lcdout $fe, 1

        Goto main
```

※AND演算子を使う場合、各評価式を( )で囲う必要はありません。

9-32、LCDOUT（LCDに文字を出力）

ＬＣＤＯＵＴ Item{、Item}

日立製LCDコントローラ44780及びその互換コントローラを使用したLCDを制御する命令です。
一般に市販されているキャラクタLCDに幅広く対応しています。
固定した文字列を表示できるほか、変数の内容や配列の内容なども表示できます。また、コマンドを指定することで、LCDの各種制御ができます。

Item　文字列、変数、制御コマンドを記述します

"（ダブルクォーテーション）で文字列を囲むと、固定した文字列を表示します。

例：LCDOUT $fe, 1, "HELLO WORLD"

制御コマンド　表示する文字列

変数の内容をASCIIコードの値を文字列でLCDに転送して表示させたい場合には、#（シャープ）を変数の前に記述します。

アイテムは、複数記述することができ、複数記述する場合にはカンマ（,）で区切ります。

例：LCDOUT B0, ,"=", B1　　B0の値の後ろに文字=が表示されその後ろにB1の内容が表示される

LCDOUT命令を使用して、LCDを初期化したり、キャラクタを表示した後に何らかの原因でLCDの電源が切断され、再度電源が投入された場合には、次にLCDOUT命令を使用する前に、内部のフラグを初期化する必要があります。LCDOUT命令の前に、下記の文を実行します。

FLAGS = 0

LCDを各種制御する制御コマンドは主に下記の通りです。

コマンド	動作内容
$FE, 1	ディスプレイを消去
$FE, 2	先頭ラインにカーソルを移動
$FE, $0C	カーソルをオフ
$FE, $0E	アンダーラインカーソルON
$FE, $0F	カーソル点滅
$FE, $10	カーソルを左に1つ進める
$FE, $14	カーソルを右に1つ進める
$FE, $80	1行目の先頭にカーソルを移動
$FE, $C0	2行目の先頭にカーソルを移動
$FE, $94	3行目の先頭にカーソルを移動
$FE, $D4	4行目の先頭にカーソルを移動

複数行のLCD使用時に、異なる行の先頭にカーソルを移動するコマンドがあります。
LCDでは、表示するデータを一度LCD内のメモリーに書き込みを行います。LCDの内部では、カーソル位置が文字を描画する位置となり、カーソルは右側にシフトしていきます。
このLCD内部のメモリーでは、1行目の先頭位置は$80、2行目の先頭位置は$C0と決まっています。

上記の表にある通り、カーソルの位置はコマンドで移動させることができます。
例えば、16文字×2行のLCDにおいて1行目の先頭に文字を描画する場合には下記のようにします。

LCDOUT $FE, $80, "HELLO WORLD"
$80は、1行目の1文字目のアドレスですので例えば、1行目の4文字目に文字を描画したい場合には、下記のように記述することで表示位置を右にシフトすることができます。

LCDOUT $FE, $80+4, "HELLO WORLD"

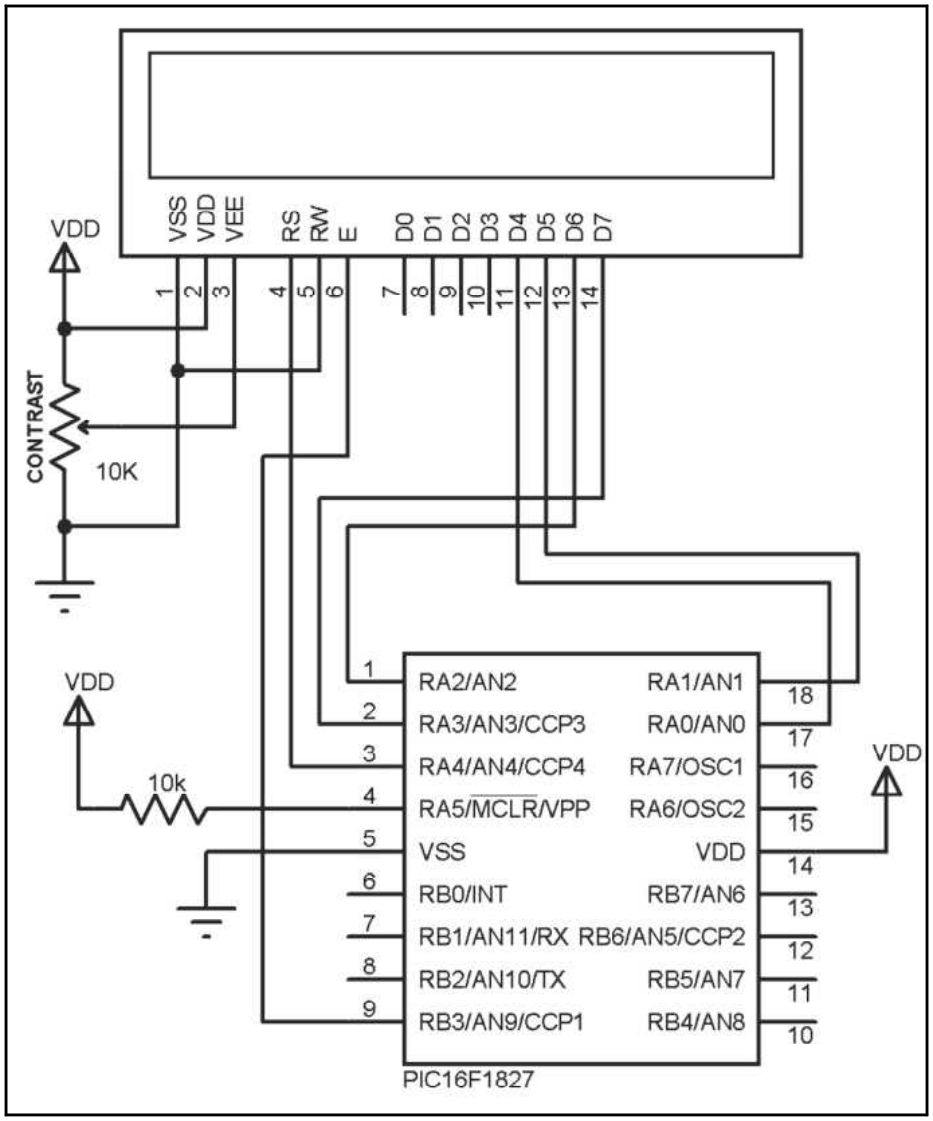
なお、多くの場合LCDをクリアーするコマンド、 LCDOUT $fe, 1 にて内部メモリー初期化されて先頭アドレスに移動しますので、1行目の1文字目から文字を表示する場合には、必ずしも $FE, $80 が必要なわけではありませんが、LCDの種類によって正しく表示できない場合などは、コマンドで文字の描画カーソルの位置を指定する必要があります。

複数行のLCDを使用している場合、2行目に文字を表示させたい場合には下記のように記述します。

LCDOUT $FE、 $C0、 "This is 2nd line"

なお、LCDによって内部のメモリーマップが異なる場合がありますのでご使用に際しては、LCDのデータシートをご参照ください。

■PICマイコンとLCDの接続について
LCDとPICマイコンの接続方法によって、その接続内容をDEFINEを使用して必ず定義する必要があります。唯一下記の回路構成の場合にのみ、DEFINEでの配線定義は必要ありません。



PBPの初期設定では、LCDのデータライン上位4ビットのDB4-DB7をPICのPORTAのbit0からbit3に接続、またレジスタ選択端子は、PORTAのbit4(RA4)に、Enable端子はPORTBのbit3(RB3)に接続します。この配線以外でPICとLCDを接続する時は、プログラムの先頭にて下記のDEFINEを使用し、結線方法を定義してます。

```
DEFINE LCD_DREG     PORTD    'データポートを設定します。
DEFINE LCD_DBIT     0        '使用するポートのLSB番号の指定です。0又は4 ※
DEFINE LCD_RSREG    PORTE    'レジスタセレクトポートを設定します。
DEFINE LCD_RSBIT    0        'レジスタセレクトのピンを指定します。
DEFINE LCD_EREG     PORTE    'ENABLEポートを指定します。
DEFINE LCD_EBIT     1        'ENABLEビットを指定します。
DEFINE LCD_BITS     4        'LCDのバスサイズを指定します。4ﾋﾞｯﾄ又は8ﾋﾞｯﾄ
DEFINE LCD_LINES    4        'LCDの行数を指定します。1行～4行
```

※データバスとして使用するポートのLSB値を指定します。LCDのデータバスサイズに4ビットを選択した場合には、該当するPIC側のポートの0～3を使用するのか、4～7を使用するのかを指定します。0～3の場合には0を、4～7の場合には4を指定します。上記のDEFINEは、下記のように定義されます。

LCD(SC1602BSなど)	PICマイコン	関係するDEFINEの記述
レジスタセレクトピン（RS）	PORTE.0	LCD_RSREG　及び　LCD_RSBIT で定義
イネーブルピン（E）	PORTE.1	LCD_EREG　及び　LCD_EBIT で定義
ライト/リード(W/R)	GND	LCDへ文字表示の際にはGNDに接続
DB0	接続しない	LCD_BITSで4を指定した場合は4ビットで通信 LCD_DBITで4を定義したのでDB4から接続する
DB1	接続しない	
DB2	接続しない	
DB3	接続しない	
DB4	PORTD.0	LCD_DBITで0を定義したのでDB4は、RD0に接続し、以降RD0～RD3を使用する
DB5	PORTD.1	
DB6	PORTD.2	
DB7	PORTD.3	

LCDOUT命令において、固定した文字列ではなくて変数に格納されている値を表示したい場合には、変数の前に#記号を付けます。これにより変数に格納されている値がそのまま出力されます。

LCDOUT命令ではデフォルトでは2行のLCDに設定されています。2行以外のLCDを使用する場合には、"DEFINE LCD_LINES"でLCDの行数を指定してください。

なお、LCDのR/WピンはGNDと接続してください。

PBPでは変数に格納されている数値を2進数・10進数・16進数のいずれかで、指定した桁数（又はビット数）で表示させることが可能です。また、値だけでなく、配列に格納された文字を表示することも可能です。これらの指定には、modifierを使用します。修正子は下記のようなものが用意されています。

modifier	動作内容
BIN{1～16}	2進数で指定したビット数で表示
DEC{1～5}	10進数で指定した桁数で表示
HEX{1～4}	16進数で指定した桁数で表示
SDEC	Long型変数の値で負の値を符号付きで表示
REP c\n	指定したキャラクタcをn回繰り返し表示
STR ArrayVar{\n}	配列に格納された文字をn文字表示

■BINの使い方
BINは変数の内容を2進数で表示します。例えば変数numに3が格納されている時は

LCDOUT BIN5 num

と記述するとLCDには 00011 と表示されます。これは変数の内容を5ビット精度で2進数表示せよということになります。

■DECの使い方
DECは変数の内容を10進数で表示します。例えば変数numに23が格納されている時は

LCDOUT DEC5 num

と記述するとLCDには 00023 と表示されます。

■REP c\nの使い方
cで指定したキャラクタをn回繰り返し表示します。cはASCIIコードとなります。

LCDOUT 67\4

と記述すると、LCDには CCCC と表示されます。

※10進数67は、ASCIIコードでCのため、Cと表示されます。

■STR ArrayVar{\n}の使い方
STR *ArrayVar{\n}* は、配列の内容を指定した桁数（文字数）で表示します。
例えばbyte型配列の場合1要素が1バイトですので、文字にするとASCIIコードで半角英数1文字分に相当します。大きさ10の配列の場合には、最大で10バイトの値が入ります。
STRは、ArrayVarで指定した配列の値をASCIIコードとして参照して文字として表示します。省略可能な\nを省略すると、宣言した配列のうち値が格納されている配列分の表示を行います。\nに値を記述すると任意の配列分を表示します。

例： LCDに ABC と表示させるプログラム

```
disp var  byte[5]
disp[0] = 65
disp[1] = "B"
disp[2] = $43

main:
       LCDOUT $FE,1, STR disp

       END
```

STRは主に、シリアル通信命令等で複数バイトのデータを配列に受信し、その内容をLCDに出力したい場合などに使用します。

■使用例

・単に変数内の値をLCDに出力する場合	LCDOUT $fe、 1、 #variable
・変数の値を16進数でLCDに出力する場合	LCDOUT $fe、 1、 hex variable
・文字列をLCDに出力する場合	LCDOUT $fe、 1、 "Hello World"
・文字列と変数の値の両方を出力する場合	LCDOUT $fe、 1、 "Result---" 、 #variable
・LCDの2行目に文字列を表示する場合	LCDOUT $fe、 $C0、 "Hello Line2"
・LCDの表示をクリアする	LCDOUT $fe、 1

LCDモジュールは一般的に電源投入後、約100ミリ秒以上経過してからでないと、動作しないもの

が多くあります。LCDモジュールを使用する場合には、最初電源投入後は約100ミリ秒程度の時間が経過してからLCDOUT命令をご利用ください。
なお、モジュールによってスタートアップ時間は違いますので、詳しくはお使いになるLCDのデータシートをご覧ください。

また、LCDのメーカーや型式によっては、コマンドの時間的なタイミングによって正しく動作しない機種もあります。ほとんどのLCDでは、問題になることはありませんが、ごく希にコマンド及びデータ信号のタイミングの相性によって、正しい表示ができない場合があります。その場合には、下記のようなタイミングを調整するDEFINEがありますので、ご使用のLCDに合わせて設定をしてご使用ください。

・制御コマンドの遅延時間をマイクロ秒単位で設定→　DEFINE LCD_COMMANDUS 1500
・データ信号の遅延時間をマイクロ秒単位で設定→　DEFINE LCD_DATAUS 44

上記設定では、制御コマンドを1500マイクロ秒、データ信号を44マイクロ秒遅延させています。多くのLCDでは、この設定は必要ではありませんが、希に相性の問題で表示に問題が発生する場合には、上記のDEFINEを挿入しておためしください。

Long変数の値を表示させる場合には、必ずDEC又はHEX又はBINの修正子を使用します。"#"を使用しての数値表示では、16ビット長までの範囲しか値を表示できませんのでご注意下さい。

例：long型変数に値70000を代入してLCDに表示させる場合

```
valiable var long
valiable = 70000
LCDOUT DEC5 variable
```

Long型変数を使用する場合、負の値を入れることができますが、符号付きの値を表示させたい場合には、SDEC修正子を使用します。SDEC修正子を使用しない場合、負の値は表示できません。

```
valiable = -120
LCDOUT SDEC variable
```

9-33、LOOKDOWN（文字列から検索）

ＬＯＯＫＤＯＷＮ Search、[Constant{、Constant...}]、Var

Constantには定数の数字や文字を指定し、Searchで指定した内容と同一の値がConstantの中にあった場合、その順番の値を変数Varに代入します。見つからなかった場合には変数の内容は変化しません。

Constantは、一番左側から012...の順番となります。Constantは数値でも文字列でも又は混在していてもかまいません。文字列の場合には各一文字毎をASCII定数として取り扱います。

下記にシリアル通信（UART）で、16進の値の文字を受信し、その値を10進数の数字に変換してUART経由で送信する例を示します。

```
HSERIN Rec_Data

LOOKDOWN Rec_Data,["0123456789ABCDEF"],Out_Data

HSEROUT Out_Data
```

9-34、LOOKUP2（指定したインデックス値のデータを取得）

ＬＯＯＫＵＰ２ Index 、［Value {、 Value}］、 Var

Indexで指定されたValueの値を取得して、変数Varに代入します。Indexに0を指定した場合、最初のValueの値がVarに代入されます。

Valueの値は、16ビット数値や文字、変数などが混在して使用できます。文字の場合にはASCII値がVarに代入されます。

例　　LOOKUP2 B0、［222、 950、 1024］、 W1

9-35、LOW（指定したピンをLowレベルにする）

ＬＯＷ Pin

Pinで指定したピンをLowレベルにします。指定したピンは自動的に出力ピンとして設定されます。TRISxレジスタをあらかじめ設定している場合には、Low命令の代わりに直接レジスタに値を代入するこもできます。

PORTB.0 = 0

9-36、NAP（短時間の一時休止）

ＮＡＰ Period

Nap命令はウォッチドッグタイマーの1サイクル分のスリープをする命令で、Nap命令を使うとPICマイコンは低消費電力モードになり周辺モジュールはOFFになります。
Periodの値によって下表の通りNap命令が実行される時間が変わります。

Period	Delay (Approx.)	
	PIC12F, PIC16F	PIC12F1, PIC16F1
0	18 mS	1 mS
1	36 mS	2 mS
2	72 mS	4 mS
3	144 mS	8 mS
4	288 mS	16 mS
5	576 mS	32 mS
6	1.152 Second	64 mS
7	2.304 Seconds	128 mS
8		256 mS
9		512 mS
10		1 Second
11		2 Seconds
12		4 Seconds
13		8 Seconds
14		16 Seconds
15		32 Seconds
16		64 Seconds
17		128 Seconds
18		256 Seconds

9-37、ON GOSUB（インデックスの値に応じて指定したサブルーチンにジャンプ）

ＯＮ Index ＧＯＳＵＢ Label { ,Label... }

Indexで指定した変数の値に応じて、Labelで指定したサブルーチンにジャンプします。
Labelは、左から順番にインデックスの値が0、1、2・・・の順で対応します。
飛び先のサブルーチンからは、RETURNで元のプログラム位置に返ることができます。

PIC18Fシリーズの場合には1024個、それ以外のPICマイコンの場合には127個のラベルを指定できます。但し、ネスト（サブルーチン内でさらに深いサブルーチンに飛ぶ）できるレベルには制限があり、14ビットコアのPICの場合には4レベル、PIC18シリーズの場合には27レベルまでとなっています。

例：下記の例はいずれも同じ動作をします。

```
ON Variable GOSUB label1, label2, label3

IF Variable=0 THEN label1
IF Variable=1 THEN label2
IF Variable=2 THEN label3
```

上記のON GOSUBと同じ動作となります。

9-38、ON GOTO（インデックスの値に応じて指定したルーチンにジャンプ）

ＯＮ Index ＧＯＴＯ Label { ,Label... }

Indexで指定した変数の値に応じて、Labelで指定したルーチンにジャンプします。
Labelは、左から順番にインデックスの値が0、1、2・・・の順で対応します。
PIC18では最大1024個、その他のデバイスでは最大127個のラベルを指定できます。

ON GOTO命令の代わりとしてBRANCH命令も使えます。

9-39、OWIN（ワンワイヤインターフェイスでデータを受信）

ＯＷＩＮ Pin、 Mode、 [Item...]

DS1820などワンワイヤインターフェイスのデバイスからデータを受信します。指定したPinから受信し、受信したデータをItemで指定した変数に格納します。また、モードの設定によって、リセットパルスを送信することができます。

ワンワイヤーインターフェイスは、別名MicroLANと呼ばれる1線式のバスシステムです。1本の信号線の途中に各デバイスを複数ぶら下げることができ、I/Oポート数の少ない組込用マイコンのインターフェイスとして開発されました。

Modeは下記のビット表を参照にMode番号を作り記述します。

Modeビット番号	内　容
0ピット目	1=データの前にリセットパルスを送信します
1ピット目	1=データの後にリセットパルスを送信します
2ビット目	0=バイトサイズデータ、 1=ビットサイズデータ

例えば、Mode番号を%000(10進数で0)の場合、リセットパルスは送信せずデータサイズはバイトサイズになります。同様に%001(10進数で1)の場合、データの前にリセットパルスを送信します。またデータサイズはバイトサイズです。%100(10進数で4)の場合、リセットパルスは送信せずビットサイズのデータを受信します。

Itemには1つ又は複数の変数を記述します。変数にはSERIN2命令などでも使用するモディファイヤを付加できます。例えば、STR修正子をつけると下記のようになります。

OWIN PORTC.1、 %000、 [STR temp\2、 SKIP 4、 count_remain、 count_per_c]

ポートCのビット1よりデータを受信します。リセットパルスは送信しません。バイトサイズの変数、tempに
2バイト受信します。次の4バイトをスキップします。そして最後の2バイトを2つの変数(count_remain変数と、count_per_c変数）にそれぞれ分離させて格納します。

9-40、OWOUT（ワンワイヤインターフェイスでデータを送信）

OWOUT Pin、 Mode、 [Item...]

ワンワイヤインターフェイスのデバイスへデータを送信します。指定したPinから、Itemの内容を送信します。また、モードの設定によって、リセットパルスを送信することができます。

Modeは下記のビット表を参照にMode番号を作り記述します。

Modeビット番号	内　容
0ピット目	1=データの前にリセットパルスを送信します
1ピット目	1=データの後にリセットパルスを送信します
2ビット目	0=バイトサイズデータ、 1=ビットサイズデータ

例えば、Mode番号を%000(10進数で0)の場合、リセットパルスは送信せずデータサイズはバイトサイズになります。同様に%001(10進数で1)の場合、データの前にリセットパルスを送信します。またデータサイズはバイトサイズです。%100(10進数で4)の場合、リセットパルスは送信せずデータサイズは、ビットサイズです。

Itemには1つ又は複数の変数を記述できます。なおモディファイヤとしてSTRを使用できます。

OWOUT PORTC.0、 %001、 [$cc、 $be]

上記の例では、PORTCのビット0よりデータを送信します。データの前にリセットパルスを送信します。その後16進数で$cc及び$beを送信します。

9-41、ON INTERRUPT（各種割込を有効にする）

ON INTERRUPT GOTO Label

割込制御命令です。この命令をプログラムの先頭で記述しておくと、割込種類を設定するINTCONレジスタ等で割込発生を許可したすべての割り込みを検出し、Labelで指定したラベルにジャンプします。

※割り込みにはRB0の外部割り込みや、TMR0の割り込み、ハードウエアUSARTの割り込みなど各種割り込みがあり、いずれの割り込みを使用する場合であっても、必ずレジスタの設定を適切に設定しなければ割り込みは実行されません。この"On Interrupt Goto"命令は、INTCONレジスタのビット7にあるGIEビットが1の際に割り込みが発生すると、Lableにジャンプするものです。

※通常割り込みが発生すると、PICは現在の実行アドレスをスタックメモリに待避させて、4番地の割り込み処理プログラムを実行します。"On Interrupt Goto"命令を記述すると、Goto命令の後に記述したLabel先が4番地に自動的に設定され、割り込み発生時には、そのラベル以降が実行されます。

どの種類の割込を使用するかはINTCONレジスタやPIR1、PIE1レジスタなどで設定します。例えばPIC16F877Aには14の割り込み要因があり、割り込みを使用する場合には、これらのレジスタの値を適切に設定する必要があります。また複数の割込を有効にしている場合、どの割込が発生しても飛び先ルーチンは同じです(Labelで指定したラベル先に飛びます)。よって、複数の割込を使用する場合には、どの割込が発生した飛び先ルーチンに飛んだのかは、飛び先ルーチン内でIF文等でレジスタの値を評価して処理を分岐させることになります。

割込ルーチン先からは"RESUME命令"で元のプログラム実行位置へ戻ります。(割込ルーチン先でのGOTO命令による別のラベルへのジャンプやGOSUB命令による別サブルーチンへのジャンプは原則として行うことは許されません。処理を分岐する必要がある場合には、必ず割込処理完了後、メインルーチンに復帰してから処理を記述するようにします。)

PBP内で割り込みルーチンを記述する場合は、"DISABLE"命令で始まり、"ENABLE"で閉じます。以下のサンプルプログラム参照してください。
通常のアセンブラやC言語で記述する時同様、RESUMEでメインルーチンに復帰する前に必ずINTCONレジスタを初期化します。

■例：PORTB.0の外部割込

```
On Interrupt Goto myint     '割り込みの宣言
INTCON = %10010000          '割り込みを制御するINTCONレジスタを設定します。

   ・・通常処理・・

DISABLE                     '通常処理のルーチンと割り込みルーチンの境目に置く命令
myint:
      High PORTA.0          '割込発生時の処理ルーチン
      INTCON = %10010000    'RESUME前にINTCONレジスタを初期化
      RESUME                'ジャンプ元に戻る

ENABLE                      '割込ルーチン終了
```

■例：タイマ0のオーバーフローによる割込

```
On Interrupt Goto TMRONT    '割り込みの宣言
INTCON = %10100000          'TMR0オーバー割込を有効に設定(INTCONレジスタのビット
OPTION_REG = %01010101      5)
                            'TMR0のプリスケーラーの設定(この設定では1/64)
   ・・通常処理・・

DISABLE
TMRONT:                     '通常処理のルーチンと割り込みルーチンの境目に置く命令
      High PORTA.0
      INTCON = %10010000    '割込発生時の処理ルーチン
      RESUME                'RESUME前にINTCONレジスタを初期化
                            'ジャンプ元に戻る
ENABLE
                            '割込ルーチン終了
```

※ハードウエアUSARTによる割り込みについてはHSERIN(2)命令の項をご覧下さい。
※割込ルーチン内での、サブルーチンは使用できません。
※On Interrupt Goto命令を使用する場合には、ブートローダーを使用する場合であっても"DEFINE ONINT_USED 1" は記述しないでください。On Interrupt命令を記述している場合

には上記のDEFINEを記述しないでもブートローダーは使用できます。

個々の割り込みについては、別途データシート等をお読みの上、レジスタ等の設定をしてください。

9-42、PAUSE（指定した時間プログラムを一時停止）

ＰＡＵＳＥ Period

Periodで指定した時間（ミリ秒単位）だけ処理を停止します。
この命令を実行しても省電力モードにはなりません。最大値は65535ミリ秒（約65.5秒程度）です。

16ビット精度で最大65535ミリ秒まで任意の時間一時停止できます。

※この命令は厳密に正確な時間を保証するものではありません。PBPでは、時間を動作クロックから擬似的に作り出しており、PAUSE命令では多少の時間のずれがあることをあらかじめご理解の上ご使用ください。そのため時計アプリケーションやストップウォッチアプリケーションなどには精度的にPAUSE命令は使えません。より精度の高い時計機能を作るには内部のTMR0などを使用します。

9-43、PAUSEUS（指定した時間プログラムを一時停止）

ＰＡＵＳＥＵＳ Period

Periodで指定した時間（マイクロ秒単位）だけ処理を停止します。
最大値は65535マイクロ秒までです。
最小ディレイ時間は、動作クロックに依存して変わります。4MHzの場合には最小時間は24μ秒、8MHzでは12μ秒、20MHzでは3μ秒、それ以上の周波数では2μ秒となります。詳しくは下表をご参照ください。

OSC	Minimum delay	Minimum delay PIC18
3 (3.58)	20us	20us**
4	24us	19us**
8	12us	9us**
10	8us	7us**
12	7us	5us**
16	5us	4us**
20	3us	3us**
24	3us	2us**
25,32,33	2us*	2us**
40,48,64	-	1us**

* PIC17 only. ** PIC18 only

※この命令は厳密に正確な時間を保証するものではありません。PBPでは、時間を動作クロックから擬似的に作り出しており、PAUSE命令では多少の時間のずれがあることをあらかじめご理解の上ご使用ください。そのため時計アプリケーションやストップウォッチアプリケーションなどには精度的にPAUSE命令は使えません。より精度の高い時計機能を作るには内部のTMR0などを使用します。

9-44、PEEKCODE（指定した番地のプログラムメモリーの値を取得）

ＰＥＥＫＣＯＤＥ Address、Var

Addressで指定したプログラムメモリー内の番地に書き込まれている値を読み出し、変数Varに代入します。PEEKCODE命令は、プログラムメモリーの値を読み出しますので、指定したスペシャルファンクションレジスタ(SFR)の値を読み込めます。変数VarはBYTE型変数を使用します。

その他、SFRには汎用ユーザーレジスタ(GPR)がありますので、そこに値を書き込んで、PEEKCODE命令で値を取得することができます。なお、データの書き込みにはPOKECODE命令を使用します。
POKECODE命令と組み合わせることで、プログラムメモリー内に定数テーブルを作ることができます。

9-45、POKECODE（指定した番地のプログラムメモリーに値を書き込む）

ＰＯＥＫＣＯＤＥ @Address、Value {、Value ...}

Address(先頭に@が付くことにご注意ください)で指定したプログラムメモリーの番地に、Valueで指定した値を書き込みます。Valueをカンマ(,)で区切り、複数記述すると、Addressで指定する番地を先頭番地として、順番に番地がインクリメントされてプログラムメモリーに書き込まれます。Valueで書き込みができる値は8ビット長ですので0～255までとなります。

なお、本命令は必ずEND命令の後に記述する必要があります。この命令はプログラムメモリーにアクセスする関係上、プログラムカウンター(PC)の値を操作します。そのため、この命令をEND命令より前のプログラム本文中に記述してしまうと、プログラムの動作が不正となりますので十分ご注意ください。

本命令は、プログラムメモリー上に、任意の定数テーブルを作る場合に使用します。指定した番地に定数を書き込んでおくことで、プログラム中ではPEEKCODE命令でアドレスを指定して値を読み出すことができます。値は、SFRに書き込まれることになりますが、主に使用できるエリアは、汎用ユーザーレジスタ(GPR)となります。汎用ユーザーレジスタの番地は、使用するPICマイコンのデータシートに記載がありますので、必ず参照して正しい番地を指定するようにします。
例えばPIC16F887のデータシートを参照するとGeneral Purpose Registersは、20h～7Fh・A0h～EFh・120h～16Fhの3つがあることが分かります。よって、このアドレスの中から番地を指定して値を書き込むことになります。

例：

```
rec VAR byte
PEEKCODE $121,rec          '120h番地の値を取得して変数recに代入します

LCDOUT dec rec             'LCDには20と表示されます

END                        'POKECODE命令はEND命令の後ろに記述します
POKECODE @$120,40,20,30    '120h番地を先頭に値を3つ書き込みます
```

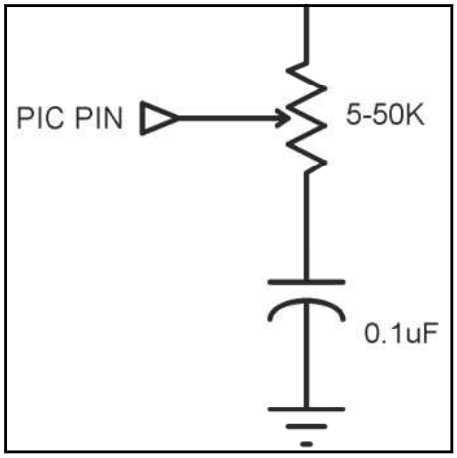
9-46、POT（ポテンショメーターの値を取得）

ＰＯＴ Pin、Scale、Var

※本命令は、主にADコンバータの搭載されていないデバイスで、擬似的にアナログ入力を実現する命令です。現在PICマイコンには多くの種類でADコンバータが内蔵されており、POT命令より高精度です。よってADコンバータ内蔵デバイスの場合にはADコンバータを使うことを強く推奨致します。

Pinで指定されたピンに接続されたポテンショメーター（可変抵抗器やボリューム等のこと）のスケールを0～255の範囲で読み込み、変数Varに読み取った値を代入します。

抵抗値の読み取りは、ポテンショメーターに直列に接続したキャパシタの放電のタイミングによって行われます。よって、配線は下記のようにします。



ポテンショメーターは、通常5KΩ～50KΩのものを使用します。
ADコンバーターの搭載していないデバイスでは、おおよその抵抗値を返された値から推測できます。

仕組みとしては最初にポテンショメーターに接続されたピンをHにしコンデンサーをフル充電した後、Lにします。コンデンサを放電してLowレベルまで下がった時間を計測することでR/Cの時定数に応じた放電時間の性質を利用しています。コンデンサの値は一定ですから放電時間は抵抗値の関数となり、時間から抵抗値が推測できます。

*Scale*はポテンショメーターの最小抵抗値の時に0、最大抵抗値の時に255となるように任意の値を指定します。通常127を最初に入れて結果を確認し微調整します。基本的にRCの時定数が大きい場合には小さい値を時定数が小さい場合には大きい値を設定します。

9-47、PULSIN（パルス幅を計測）

ＰＵＬＳＩＮ Pin、Stare、Var

Pinで指定したピンに入力されたパルス信号のパルス幅を計測します。結果は16ビット精度で変数に返されます。10us～65535usの幅のパルス信号が計測できます。

Stateの値が0の場合、Lowレベルの間の時間を計測します。1の場合には、Highレベルの間の時間を計測します。パルス幅が計測できない場合、例えばパルスの立ち上がり、立ち下がりエッジが検出できなかった場合には変数に0が返ります。
パルス幅はPICマイコンの動作周波数に応じて返される分解能が変わります。4MHzの場合には10マイクロ秒単位で、返されます。20MHzの場合には2マイクロ秒単位になります。
この精度は、DEFINE OSCの設定に関係なく、実際の動作周波数に依存しています。

PULSIN命令は、通常パルスが検出できない場合最大65535カウントまで待ちます。パルスの検出の最大値を変更したい場合には、DEFINEで設定できます。

DEFINE　PULSIN_MAX　1000

次の例では、PORTB.4からのHighレベルのパルス幅を計測して変数W3にその結果を返します。

PULSIN PORTB.4、 1、 W3

※この命令は、入力されたパルスのHレベル又はLレベル時の、パルス幅を取得する命令ですので、周期をカウントする命令ではありません。周波数の測定には、COUNT命令を利用します。

※周波数を測定するには次の例にようにします。（参考例です。PICを20MHzで動作させた場合です）

```
 v1   var    word
 v0   var    word
 v    var    word
 f    var    word

main:  pulsin PORTB.4, 1, v1
       pulsin PORTB.4, 0, v0

       v = (v1 + v0) * 2
       f = (50000 / v) * 20

       lcdout $fe, $c0, #f, "Hz"
```

9-48、PULSOUT（パルスを出力）

ＰＵＬＳＯＵＴ Pin、Period

Pinで指定したピンから、Periodで指定した時間のパルス信号を発生します。
パルス幅は動作周波数により変わります。4MHz動作の場合、Periodで指定した値よりも10マイクロ秒ずつ増加します。20MHzの場合には2マイクロ秒ずつ増加します。

PULSOUT PORTB.5、 100

上記の場合、PICを4MHzで動作させていた場合には、1ミリ秒のパルス（10μs×100=1ms）を、20MHzで動作させている場合には、200マイクロ秒のパルスをRB5から出力します。

※この命令は単発のパルスを出力する命令です。

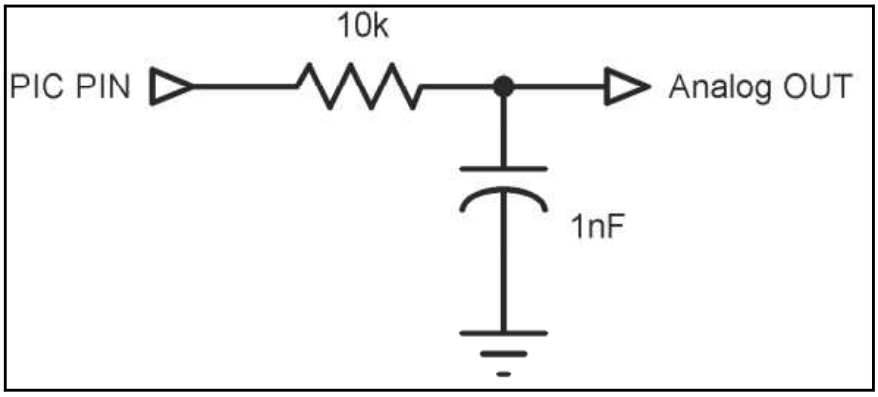
9-49、PWM（ソフトウエアPWM出力）

ＰＷＭ Pin、Duty、Cycle

パルス幅変調信号（PWM）を指定したPinより出力します。パルス幅変調は、パルス幅のONとOFFの比率を変える変調方式です。
このPWM命令は、ハードウエアPWMを搭載しないPICでもPWM信号を出力できるソフトウエアPWMです。
ハードウエアPWM搭載のデバイスをお使いの場合には、HPWM命令をお使いください。より正確な制御ができます。
Dutyには0（0%）～255(100%)の値を記述します。Dutyは256ステップで設定できます。50%の場合（HとLの間隔が同じ）には、127を指定します。

Cycleにはパルスの繰り返し回数を記述します。Cycleの値は0～255です。Cycleの時間は、動作周波数によって変わります。動作周波数が4MHzの場合1Cycleは約5ミリ秒、20MHzの場合には約1ミリ秒の長さとなります。よって、Cycleの値を100とした場合、4MHz動作の場合には、500ミリ秒、20MHz動作の場合には、100ミリ秒間パルスが出力されます。

PWMとしてアナログ的に変化する出力を得たい場合には下図のようなローパスフィルターを付けることで、矩形波の高調波成分をカットした波形を取り出せます。



例：　デューティー比50%で100サイクルのPWM信号をRB7より出力する場合

PWM PORTB.7、 127、 100

9-50、RANDOM（疑似乱数の発生）

ＲＡＮＤＯＭ Var

疑似乱数を発生してVar変数にその値を返します。Varの値は16ビットのため必ずWord変数を利用します。
0を除いた最大65535までのランダムな値が返ります。

※疑似乱数は、完全な乱数ではなく算術処理で生成します。その為完全な乱数ではなく、場合によっては一定の規則性が発生する場合もあります。

9-51、READ（EEPROMからデータを読み込む）

ＲＥＡＤ Address、Var

PIC12F/16F/18FシリーズのようなEEPROM搭載タイプのPICマイコンに内蔵されたEEPROMの指定した番地から、WRITE命令で書き込んだデータを変数に読み込みます。

Address	変数又は定数（0～255）数値を指定。読み込むEEPROMアドレスを指定します。
Var	EEPROMより読み込んだデータを格納するbyte変数を指定します。

例：　READ ５ 、 B2

上記の例では、PICマイコン内部のEEPROMの5番地からデータを読み込んで結果を変数B2に格納します。

※PICマイコンに内部されたEEPROMではなく、外部のI2C通信式のEEPROM(24LC256等)にアクセスする場合には、I2CREADやI2CWRITE命令を使用します。PIC12CEシリーズでは本命令は使用できません。

9-52、READCODE（コードスペースからデータを読み込む）

ＲＥＡＤＣＯＤＥ Address、Var

READ命令はデータEEPROMからのデータを読み込むのに対し、READCODE命令は、コードスペースからのデータを読み込みます。引数Addressで指定した場所からデータを読み込み変数varに格納します。
いくつかのPIC16FシリーズやPIC18Fシリーズではプログラムコードエリアに対して、プログラム実行中でもアクセスできます。
PIC16Fのデバイスでは、14ビットサイズのデータが、PIC18Fデバイスではバイトサイズ又はWORDサイズのデータが読み込まれます。使用するデバイスに合わせてVarで指定する変数の型を選択してください。

なお、コードスペース（プログラムメモリー）からの読み込みに関してはブロックあたりのバイト数は関係ありません。詳しくはERASECODE命令の項目をご覧ください。

9-53、REPEAT....UNTIL（条件式に従い繰り返し処理）

ＲＥＰＥＡＴ
　　Statment...
ＵＮＴＩＬ Condition

Statmentに記述したプログラムをConditionの値が真になるまで、繰り返し実行します。

```
i var byte
i = 0

REPEAT
       PORTB.0 = 0
       i = i + 1
UNTIL i > 7
```

上記の例では、変数iが8以上になるまで繰り返します。なお、この命令を使う場合には必ずカウント対象の変数は、REPEATの前で0を代入して初期化してください。

9-54、RESUME（割込先サブルーチンからメインルーチンへ復帰）

ＲＥＳＵＭＥ {Label}

割込み発生後、割込ルーチンから、メインルーチンへ処理をジャンプ（復帰）します。
RETURN命令と似ていますが、RESUME命令は割込みルーチンからの復帰の際に使用します。RETURN命令はGOSUB命令でジャンプしたサブルーチンからの復帰の際に利用します。

省略可能なLabelに、ラベル名を記述すると、ジャンプ元に復帰せず指定したラベルへ処理を移動させます。しかしこの方法を使うと、PIC16Fシリーズなどではスタックポインタに問題を発生させ、以降割込が使用できなくなるという問題が発生します。
PIC18シリーズでは、ラベル指定してラベルへ処理が移動後はスタックポインタは0にクリアされるため問題は発生しません。Labelの指定はできる限りしないようにすることが望ましいです。

※詳しくは割込命令 "On Interrupt Goto"命令の項をご参照下さい。

9-55、RETURN（サブルーチンからメインルーチンへ復帰）

ＲＥＴＵＲＮ

GOSUB命令で、コールし実行したサブルーチンからメインルーチンへ復帰する命令です。
戻る場所は、GOSUB命令を実行した次の行からです。詳しくはGOSUB命令の項をご覧ください。

9-56、REVERSE（ピンの入出力方向の反転）

ＲＥＶＥＲＳＥ Pin

Pinで指定したピンの入出力方向を反転させます。
入力になっていたら出力に、出力になっていたら入力に反転します。

9-57、SELECT CASE（条件式による複数の分岐）

```
ＳＥＬＥＣＴ ＣＡＳＥ Var
            ＣＡＳＥ Expr1 {、Expr...}
                    Statment...
            ＣＡＳＥ Expr2 {、Expr...}
                    Statment...
            {ＣＡＳＥ ＥＬＳＥ
                    Statment...}
ＥＮＤ ＳＥＬＥＣＴ
```

Select Case命令はVar変数の値を条件（評価して）として処理を分岐する分岐命令です。
Varには評価する変数を指定します。

各Caseの後のExprには条件を記述します。Varの値が真となるExprへ処理を分岐し、そのCaseに対応したStatmentを実行します。
Exprの条件が1つも真でない場合には、CASE ELSEにジャンプしStatmentを実行します。CASE ELSEは省略可能で、省略した場合にはEND SELECT以降にプログラムが移動します。

その他、var変数に対して比較を行う場合には、比較演算子を用いて"Case Is"を使用します。

```
SELECT CASE x                    '変数 x の値が評価されます
        Case 1                   'x の値が 1 の時実行
                y = 10
        Case 2,3                 'x の値が 2 又は 3 の時実行
                y = 20
        Case Is > 5              'x の値が 6 以上の時実行
                y = 30
        Case Else                '上記いずれの条件も真でない時実行
                y = 0
ENDSELECT
```

9-58、SERIN2（ソフトウエア非同期式シリアル通信UARTでデータを受信）

ＳＥＲＩＮ２ Datapin{/Flowpin}、 Mode、 {PartityLabel、} {Timeout、Label1} [Item...]

非同期式シリアル通信(UART)でデータを受信します。本命令はソフトウエアでUART機能を実現するための命令です。パソコンのRS232Cや、UART通信を行う機器間との通信に使用します。

本命令はUART通信をソフトウエアで実行するものです。多くのPICマイコンにはハードウエアUSART可能が搭載されており、ハードウエアUSART機能を使ってUART通信を行いたい場合には、HSERIN命令及びHSEROUT命令を使います。このSERIN2命令はハードウエアUSARTの搭載されていないデバイスでUARTを行う場合のみ使います。

１つまたはそれ以上のItems（データ）を指定したDatapinより受信します。
Datapinはデータを入力するポート(Rxピン)を指定します。
通信のプロトコル（通信速度やデータ長などのパラメーター）は、Modeによって設定します。Modeの値は表を参考にしてください。

Datapin　データ受信を行うピン(RXとして使用するピン)を指定
Flowpin　フロー制御入力　（変数又は定数）　［省略可］
Mode　以下の表を参照して設定、通信プロトコルの設定

Mode	ﾎﾞｰﾚｰﾄ	ﾃﾞｰﾀ	ﾊﾟﾘﾃｨ	極性
3313	300ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
813	1200ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
396	2400ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
188	4800ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
84	9600ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
32	19200ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ

極性とは、信号の極性のことで、RS232C通信をする場合、レベル変換ICを介して通信する場合には、ノーマルに設定してください。レベル変換ICを介さずRS232Cと直接PICマイコンのピンを接続する場合には、極性を反転させます。
上記の表以外でのボーレートを設定したい場合には計算によりボーレートを算出することができますが、正式にサポートされたものではないため、速度によっては通信できない場合があります。

ボーレート ＝ （1000000 / 希望のボーレート）－２０

例：300ボーにしたい場合
(1000000 / 300)-20 = 3333 - 20 = **3313**

上記の表の通り、データ長8ビット長・1ストップビット・ノンパリティ・極性ノーマルの一般的な設定の場合には通信速度の違いによりModeの値が変わるだけです。データ長やパリティビットを付加したい場合にはModeの値を上記の表以外に、以下のように変更します。

■パリティを挿入したい場合
パリティを挿入すると、データビットの最後の1ビットがパリティとして利用されます。よってパリティを使用すると、データ長は1ビット少なくなります。
偶数パリティを挿入したい場合には、上記の通信速度により決めたModeの値に $1000 を加算します。
パリティを挿入するデータ長は1ビット少なくなりますので「通信速度9600bps、偶数パリティ、データ長7ビット長」の場合には、次のようになります。

```
SERIN2 Tx、 84+$1000 、［REC］        '9.6kbps、偶数パリティ、7ビット長、極性ノーマル
```

なお、8ビット長でパリティを挿入したい場合にはデータ長を9ビットにします。データ長は次のDEFINEをプログラム先頭で記述します。これでパリティが挿入されていればデータ長は8ビット長になります。

```
DEFINE SER2_BITS 9                    'データ長が9ビット長になります
```

この他、奇数パリティにしたい場合には、次のDEFINEをプログラムの先頭に記述します。

```
DEFINE SER2_ODD 1
```

■極性の変更
PICマイコンとRS232Cポートの間にMAX232C等のRS232CドライブICが挿入されている場合には極性はノーマルに設定し、通常は前ページのModeの値（通信速度の値）だけで動作します。
しかし、RS232CドライブICを用いずにPICマイコンのピンとRS232Cポートを直接接続させて通信を行いたい場合には、極性を反転させる必要があります。極性を反転させるためにはModeの値に $2000 を加算します。

```
SERIN2 Tx、 84+$2000 、［REC］        '9.6kbps、偶数パリティ、8ビット長、極性反転
```

但し、この設定では信号の論理状態は反転しますが、電圧レベルはあくまでもPICマイコンはVccを上限とするTTLレベルです。RS232Cとは電圧レベルが違いますので、直結すると場合によってはマイコンを壊してしまうことがあります。できる限りRS232CドライブICを介してRS232Cポートと接続してください。

ParityLabel　パリティ設定時、パリティエラー発生時の飛び先ラベルの指定［省略可］

Timeout、Label
待ち受け限度時間の設定です。単位はms。最高は65535ミリ秒。
ここで指定した時間待ってもデータが入力されない場合はLabelへ分岐します。
Timeout.Labelを省略するとデータを受信するまでﾌﾟﾛｸﾞﾗﾑは待機します［省略可］

Item　データを格納する変数です。あらかじめ、宣言した変数又は配列を使用します。
Itemにbyte型変数を用意すると、受信できるデータサイズは1バイトとなります。
ASCIIコードでは、1文字分の受信となります。複数バイト受信したい場合には、変数ではなくて配列を使用します。受信されたデータは1バイト毎に配列1要素ずつに格納されます。

例として変数に配列を用いないで、数値をASCIIコードで1文字分受信するプログラムを示します。

```
REC  var  byte
RX   var  PORTC.7

SERIN2 RX, 84, [REC]              '9.6kbps で受信
LCDOUT $FE, 1, #REC               '変数 REC に受信した内容を LCD に表示
```

この場合、受信できるデータ長は8ビットです。受信データは変数RECに格納され、LCDにその内容が表示されます。#マークを変数の前に付けたため、変数に格納された値に対応するASCIIコードより、LCDには文字が表示されます。例えば70を受信した場合LCDには"F"が表示されます。

複数バイト受信する場合には、配列を使用します。配列を使用した場合には、STR修正子を用いて、受信するバイト数を指定します。例えば5バイト受信したい場合には下記のようにします。

```
REC  var  byte[5]
RX   var  PORTC.7

SERIN2 RX, 84, [STR REC¥5]        '配列 REC に 5 バイト受信信
LCDOUT $FE, 1, STR REC¥5          '配列の内容を 5 バイト分 LCD に表示
```

STR修正子は配列を用いた場合には必ず使用します。書式は、"SRT ArrayVar\n{\c}"で配列にnバイトまでデータを受信する、という内容になります。省略可能なcは、受信終了のキャラクタで、ここで指定した値を受信すると、\nで指定したサイズに受信したデータサイズが達していなくても受信を終了します。

STR修正子を用いることで受信するデータサイズを定義しています。よって例えば、要素数10の配列を準備した場合でも、 "SRT Arry\5" と記述すると5バイト分受信したところで受信完了となります。

各配列に格納された内容を要素毎に参照するには、　変数名［配列数］　で指定できます。
例えば、受信したデータが "ABC" の3バイトであるかどうかを判定して分岐処理をするには、IF文を用いて次のようにします。

```
REC  var  byte[3]                                    '配列を宣言
RX   var  PORTC.7

SERIN2  RX, 84, [STR REC¥3]                          '8 ビット ×5 を受信

If  REC[0]="A" AND REC[1]="B" AND REC[2]="C"   Then   '配列の判定
      Lcdout $fe, 1, "REC = ABC"
ENDIF
```

配列の要素1つずつを参照して、AND演算子を用いて判定を行います。
なお、実際にSERIN2命令で受信する値は数値です。文字として認識しているのはASCIIコードとしているためで、例えば上記の例で、"ABC"という文字列は実際には"65、66、67"という3バイトのデータです。
わかりやすいよう文字列で表記する場合には、上記の例のように文字を"（ダブルクォーテーション）で囲みます。

SERIN2命令は割り込み処理を利用しているわけではなく、ソフトウエア的に非同期通信を実現しています。そのため、SERIN2命令では、データが受信できるまで処理を待機します。また配列を用いて複数バイト受信するようSTRを使用している場合には、指定したデータサイズを受信するまで、受信待機状態となりプログラムはSERIN2命令の位置で待機することになります。

省略可能なTimeoutとLabelを記述することで、指定したTimeout時間データが受信できない場合には、指定したLableに処理をとばすことができます。Timeoutはミリ秒単位で指定します。
SERIN2命令はあくまでも、データ受信時にSERIN2命令が実行されていないとデータを受信できません。他のプログラムを実行中の時にデータが入ってきても、SERIN2命令が実行されていなければデータは受信できずすべて取りこぼすことになります。
いつ来るか分からないデータを受信する場合には、通常割り込みを使用します。割り込みは、ハードウエアUSARTを搭載したデバイスの際使用できます。使用する場合には、HSERIN命令を使用します。割り込み受信する場合には、HSERIN命令の項をご覧ください。

SERIN2命令は、受け取るデータの形を変換したり特定の作業を指定できるModifier（修正子）が利用できます。　{　} 内の数字は桁数を指定します。

Modifier	動　作　概　要
BIN{1…16}	データをバイナリ形式で受け取ります。16ビットまで指定できます。
DEC{1…5}	データを10進数で受け取ります。
HEX{1…4}	データを16進数で受け取ります。
SKIP n	nで指定したバイト分の受信をスキップします。
WAIT（　）	（　）で指定したキャラクターが受信されるまで待ちます。指定したキャラクタ受信後は通常通り変数に受信内容を格納します。
STR ArrVar\n{\c}	nバイト受信し、配列として宣言したArrVar配列に受信した値を格納します。省略可能な\cに値を記述した場合、\nで指定したサイズ分受信していなくても、その値を受信するとすぐに受信処理を完了します。

■BINの使い方
BINは変数の内容を2進数で処理します。例えば変数numに10進数の3が格納されている時は

LCDOUT BIN5 num

と記述すると変数には 00011 と格納されます。これは変数の内容を5ビット精度で2進数表示せよということになります。BINの後に数値をつけるとビット数を指定できます。

■DECの使い方
DECは変数の内容を10進数で処理します。例えば変数numに10進数の23が格納されている時は

LCDOUT DEC5 num

と記述すると変数には 00023 と格納されます。DECの後ろに数値をつけると桁数を指定できます。

■HEXの使い方
HEXは変数の内容を16進数で処理します。例えば変数numに10進数の50が格納されている時は

LCDOUT HEX6 num

と記述すると変数には 000032 と格納されます。HEXの後ろに数値をつけると桁数を指定できます。

■WAITの使い方

WAITは、指定した値を受信するまで受信の処理を待ちます。
例えば、

SERIN2 RX 、 84、［WAIT（"A"）、 B0］

と記述すると、文字"A"が受信されるまで待ち、文字"A"を受信後は変数B0に受信した内容を格納します。その他WAITでは特定の値を記述することもできます。

■STR ArrVar\n{\c}の使い方

STRを利用した場合、\cを省略するとnバイト受信しないと通信は完了しません。\cで値を指定するとその値を受信した時点で受信処理を終了します。
文字を指定する場合には" （ダブルクォーテーション）で囲んでください。値の場合には、そのまま記述します。例を下に示します。

SERIN2 PORTC.7、 84、［STR BB\10\"D"］

文字"D"を受信すると直ちに受信処理を終了します。文字列"D"を受信できない場合は10バイト分受信します。
SERIN2命令で受信した内容をLCDOUT命令でLCDに出力する際、\cを設定した場合は出力する文字数が確定しないため、LCDOUT命令側のSTRの\nは記述しないようにします。

LCDOUT $fe、 1、 STR BB

※LCDOUT命令では「STR 配列名」で、実際に受信した配列すべての内容を表示します。
特定の配列分だけ表示する場合には、 「STR 配列名 \n」 と記述します。

※なお、ハードウエアUART並びにシリアル通信割り込みを使用したい場合には、HSEROUT命令又は
HSERIN命令をご利用ください。SEROUT2命令及びSERIN2命令は、ソフトウエア的にシリアル通信を実現していますので、割り込み処理などはできません。
また、PICマイコン同士でUART通信を行う場合には必ずHSEROUT命令又はHSERIN命令を使用します。

9-59、SEROUT2（ソフトウエア非同期式シリアル通信UARTでデータを送信）

ＳＥＲＯＵＴ２ Datapin {/Flowpin}、 Mode、 {Pace、} {Timeout、Label} [Item...]

非同期式シリアル通信(UART)でデータを出力します。本命令はソフトウエアでUART機能を実現するための命令です。パソコンのRS232Cや、UART通信を行う機器間との通信に使用します。

本命令は、UART通信をソフトウエアで実行するものです。多くのPICマイコンにはハードウエアUSART可能が搭載されており、ハードウエアUSART機能を使ってUART通信を行いたい場合には、HSERIN命令及びHSEROUT命令を使います。このSEROUT2命令はハードウエアUSARTの搭載されていないデバイスでUARTを行う場合のみ使います。

非同期式シリアル通信で、指定した変数又は配列の内容をDatapinから出力します。
１つまたはそれ以上のItemの内容を出力します。指定した各ピンは自動的に入出力方向が設定されます。

Datapin	データを出力するピン(TXとして使用するピン)を指定します。
Flowpin	フロー制御入力（変数又は定数）　　[省略可]
Pace	次の１バイトを送信するまでの時間。単位はmsで最大65535ms。　[省略可]
Timeout、Label	待ち受け限度時間の設定です。単位はmsで最大65535ms。この時間待ってもデータが出力できない場合はLabelへ分岐します。[省略可]
Item	出力する内容を格納した変数、配列又は定数も指定できます。
Mode	通信速度やデータ長、パリティなど通信プロトコルを表す数値です。数値は以下の表を参照ください。

Mode	ﾎﾞｰﾚｰﾄ	ﾃﾞｰﾀ長	ﾊﾟﾘﾃｨ	極性
3313	300ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
813	1200ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
396	2400ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
188	4800ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
84	9600ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ
32	19200ﾎﾞｰ	8bit	ノンパリティ	ﾉｰﾏﾙ

極性は、PICマイコンとRS232Cを、レベル変換IC(MAX232等)を介して接続する場合にはノーマルに設定します。PICマイコンとRS232Cを直結する場合には、反転させます。通常はレベル変換ICを介してRS232Cと接続しますので、極性はノーマルにします。
上記の表以外でのボーレートをお使いの場合には計算によりボーレート（通信速度）を算出することもできますが、速度によっては正しく通信できないことがあります。

ボーレート ＝ （1000000 / 希望のボーレート）－20

通信速度は、上記の表から選択してください。データ長、パリティの有無、極性が表の通り、「8ビット長、ノンパリティ、極性ノーマル（レベル変換IC適用)」でよければModeの値の設定は完了です。プロトコルに変更を加える場合には、次の手順で行います。

■パリティを挿入したい場合
パリティを挿入すると、データビットの最後の1ビットがパリティとして利用されます。
偶数パリティを挿入したい場合には、上記の通信速度により決めたModeの値に $1000 を加算します。
パリティを挿入するデータ長は1ビット少なくなります。例えば、「通信速度9600bps、偶数パリティ、データ長7ビット長」の場合には、次のようになります。

SEROUT2 Tx、 84+$1000 、[REC]　　'9.6kbps、偶数パリティ、7ビット長、極性ノーマル

なお、8ビット長でパリティを挿入したい場合にはデータ長を9ビットにします。データ長は次のDEFINEをプログラム先頭で記述します。これでパリティが挿入されていればデータ長は8ビット長になります。

DEFINE SER2_BITS 9　　'データ長が9ビット長になります

この他奇数パリティにしたい場合には、次のDEFINEをプログラムの先頭に記述します。

DEFINE SER2_ODD 1

■極性の変更
PICマイコンとRS232Cポートの間にMAX232C等のRS232CドライブICが挿入されている場合には極性はノーマルに設定し、通常は前ページのModeの値（通信速度の値）だけで動作します。
しかし、RS232CドライブICを用いずにPICマイコンのピンとRS232Cポートを直接接続させて通信を行いたい場合には、極性を反転させる必要があります。極性を反転させるためにはModeの値に$2000 を加算します。

SEROUT2 Tx、 84+$2000 、［REC］　　'9.6kbps、偶数パリティ、8ビット長、極性反転

但し、この設定では信号の論理状態は反転しますが、電圧レベルはあくまでもPICマイコンはVccを上限とするTTLレベルです。RS232Cとは電圧レベルが違いますので、直結すると場合によってはマイコンを壊してしまうことがあります。できる限りRS232CドライブICを介してRS232Cポートと接続してください。

PBPでは、パリティの有無及び極性の設定は、Modeの値にそれぞれの設定値を加算する形で記述します。「Mode ＝ ボーレート設定値　＋　パリティ有無　＋　極性指定」　となります。

通信速度9600bps、データ長8ビット長、パリティなし、1ストップビット、極性ノーマルの一般的なプロトコルで通信を行う場合には、下記のようになります。下記例では、変数Varの値を出力します。

SEROUT2 TX、 84、［Var］

Varには変数を指定します。バイト型は1バイトですので、ASCIIコードでは1文字相当です。
複数バイト出力したい場合には、[ ]内に固定した文字列を記述するか、配列を記述します。
固定した文字列を送信する場合には送信したい文字例を［　］内に "（ダブルクォーテーション）で囲んで直接指定できます。例えば文字列 Hello World を送信したい場合には次のようにします。

SEROUT2 TX、 84、["Hello World"]

配列の場合には、STR修正子を用いて、[ ]内に配列を記述します。
下記の例では、文字列 ABC を送信します。

```
S_DATA var byte[3]                          '配列を宣言

S_DATA[0] = "A"                             '各配列に文字列( 8 ビット)を格納
S_DATA[1] = "B"
S_DATA[2] = "C"

SEROUT2 PORTC.7, 84, [STR S_DATA]           '文字列"ABC"が送信される
```

STRは、宣言された配列の全要素を出力する場合に配列の前に付ける修正子です。
上記の例でSTRを付けずに［S_DATA]とだけ記述すると、文字列"A"だけが送信されます。

修正子を用いることで、送信フォーマットを任意の形式に修正することができます。

Modifier	動　作　概　要
BIN{1～16}	データをバイナリ形式で出力します。
DEC{1～5}	データを10進数で出力します。
HEX{1～4}	データを16進数で出力します。
REP c\n	cで指定した値をn回リピートして出力します。例えば"0"\4と指定した場合、0000 と出力します。
STR ArrVar{\n}	配列ArrVarの内容をnバイト(要素数n個)を出力します。 \nを省略すると、ArrVarの全要素を出力します。

BIN、DEC、HEXでは変数の内容を指定した形式に変換すると共に、{　}内の数値を記述することでビット数や桁数を指定できます。
下記の例では変数"S_DATA"の内容を10進数、3桁で出力します。
変数には、12345の値が格納されていますが、　DEC3　のによって、送信される数値は"345" になります。

```
S_DATA  var  Word
S_DATA  =  12345

SEROUT2  PORTC.7, 84, [DEC3 S_DATA]            '文字列"345"が送信される
```

※なお、ハードウエアUART並びにシリアル通信割り込みを使用したい場合には、HSEROUT命令又は
HSERIN命令をご利用ください。SEROUT2命令及びSERIN2命令は、ソフトウエア的にシリアル通信を実現していますので、割り込み処理などはできません。

9-60、SHIFTIN（クロック信号に同期してデータを受信）

ＳＨＩＦＴＩＮ DataPin、Clockpin、Mode、[Var{＼bits}...]

同期式のシリアルデータを指定したDatapinから、クロックに同期して読み込みます。読み込んだ値を変数Varに格納します。オプションの"\bits"にビット数を指定すると指定されたビット数を読み込みます。指定しない場合は8ビットに設定されます。

Modeでモード番号を指定します。モード番号は下表の通りです。

Datapin　　データ入力ピン
Clockpin　　クロックピン
Var　　シフトデータ用変数
bits　　ビット数1～16　省略した場合8ビット

Mode	Mode No	動　作　概　要
MSBPRE	0	上位のビットから順番にデータを読み込みます。クロックを送る前にデータを読み込みます。クロックの立ち上がりでデータを読みます。
LSBPRE	1	下位のビットから順番にデータを読み込みます。クロックを送る前にデータを読み込みます。クロックの立ち上がりでデータを読みます。
MSBPOST	2	上位のビットから順番にデータを読み込みます。クロックを送信後データを読み込みます。クロックの立ち上がりでデータを読みます。
LSBPOST	3	下位のビットから順番にデータを読み込みます。クロックを送信後データを読み込みます。クロックの立ち上がりでデータを読みます。
	4	上位のビットから順番にデータを読み込みます。クロックを送る前にデータを読み込みます。クロックの立ち下がりでデータを読みます。
	5	下位のビットから順番にデータを読み込みます。クロックを送る前にデータを読み込みます。クロックの立ち下がりでデータを読みます。
	6	上位のビットから順番にデータを読み込みます。クロックを送信後データを読み込みます。クロックの立ち上がりでデータを読みます。
	7	下位のビットから順番にデータを読み込みます。クロックを送信後データを読み込みます。クロックの立ち下がりでデータを読みます。

9-61、SHIFTOUT（クロック信号に同期してデータを送信）

ＳＨＩＦＴＯＵＴ DataPin、Clockpin、Mode、[Var{＼bits}...]

Var変数に格納されたデータをクロックに同期して、指定したdatapinから出力します。
Modeには、下表の該当するモード番号を記載します。

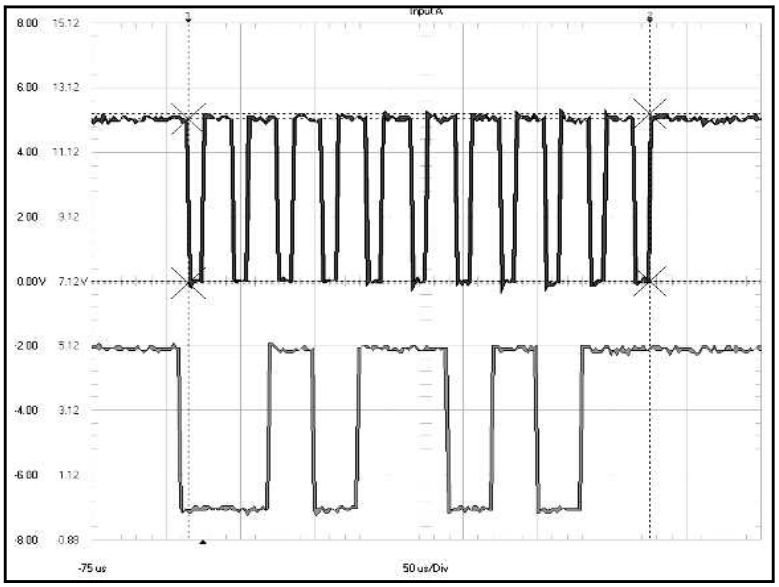
Datapin　　データ出力ピン
Clockpin　　クロック出力ピン
Var　　シフトデータ出力変数
bits　　（省略可能）ビット数1～16　省略した場合には8ビットに指定

モード番号0と1は、クロックのアイドル時は該当のクロックピンはLowレベルです。すなわち、クロックの立ち上がり時にデータを出力します。
モード番号4と5はクロックアイドル時は該当のクロックピンはHighレベルです。すなわち、クロックの立ち下がりでデータを出力します。

Mode	Mode No	動　作　概　要
LSBFIRST	0	下位のビットから順にデータを出力します。クロックの立ち上がりでデータを出力します。クロックアイドルはLowです。
MSBFIRST	1	上位のビットから順にデータを出力します。クロックの立ち上がりでデータを出力します。クロックアイドルはLowです。
	4	下位のビットから順にデータを出力します。クロックの立ち下がりでデータを出力します。クロックアイドルはHighです。
	5	上位のビットから順にデータを出力します。クロックの立ち下がりでデータを出力します。クロックアイドルはHighです。

例えば下記のようなプログラムを実行すると、次のような波形が観察できます。

```
out_data     =    %11010110100
Shiftout  PORTB.0、  PORTB.1、  4、  [out_data\11]
```



クロックの周波数は動作周波数に依存しますが約50KHz程度です。クロックは下記のDEFINEによってマイクロ秒単位で変更できます。

```
DEFINE  SHIFT_PAUSEUS   100                    '100マイクロ秒に設定
```

9-62、SLEEP（デバイスを指定時間一時休止モードにする）

ＳＬＥＥＰ Period

Periodで指定した時間だけ低消費電流のスリープモードになります。

Periodには秒単位で値を指定します。
PIC18用のPBPLを使用している場合にはかなり長い時間をPeriodに指定できます。PBPL以外を使用している場合には16ビット長なので最大時間は0～65535秒（約18時間）となります。

SLEEP命令を実行すると、周辺モジュールもスリープしますのでSLEEPモードに入る前にその機能を停止させる必要があります。例えばADコンバータの場合には、レジスタの設定でADコンバータモジュールをOFFにします。

このSLEEP命令は、ウォッチドッグタイマー(WDT)を使用してスリープ時間の時間を管理しています。WDTをチェックしてウェイクアップをしますが、まだ指定した時間に到達していない場合には再度スリープモードに移行し次のWDTのウェイクアップを待ちます。

よって、発振周波数に依存しません。Periodの単位は秒で、まで指定できます。
復帰するためには、ハードウエアリセット又は割込による復帰が可能です。

9-63、SOUND（トーン又はホワイトノイズの発生）

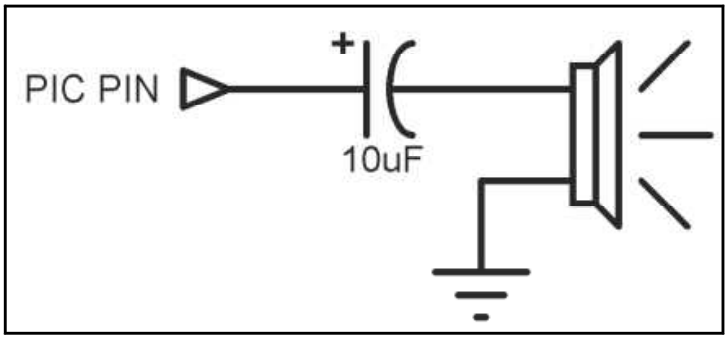
ＳＯＵＮＤ Pin、[Note、Duration{、Note、Duration...}]

指定したPinからトーンもしくはホワイトノイズを発生します。

Note　　0の時は無音です。1～127までの間はトーンを発生します。
128～255までの区間はホワイトノイズを発生します。
数値が大きくなるほど高音階になります。
値が1の時では約78.74Hz、値が127では約10000Hzです。

Duration　　音の長さを0～255の範囲で設定します。変数又は定数が利用できます。
0～255の値はNoteの音を出す時間を約12ミリ秒単位で決定します。

SOUND命令はTTLレベルの矩形波ですがPICマイコンはI/Oピンの駆動能力が大きいため、圧電スピーカーはコンデンサーを通すだけで直接駆動できます。必要に応じてアンプ等を接続してください。



9-64、STOP（プログラムの強制停止）

ＳＴＯＰ

プログラムを強制的に停止させます。PICマイコンは低消費電力モードにはなりません。
再度プログラムを走らせるには、MPUのハードウエアリセットが必要です。

9-65、SWAP（2変数間でデータの交換）

ＳＷＡＰ Var1、Var2

2つの変数、Var1とVar2の間でその格納されている値をそれぞれ交換します。
バイト変数とワード変数間など異なるデータ型のデータ交換にも使用できます。

9-66、TOGGLE（ピンの状態を反転）

ＴＯＧＧＬＥ Pin

指定のピンの出力状態を反転します。現在の出力がHighの時はLowに又はその逆になります。

9-67、USBIN（USB2.0機能内蔵のデバイスにおいてUSB経由でデータを受信）

ＵＳＢＩＮ Endpoint、 Buffer、 Conveter、 Label

USB2.0インターフェイス搭載のデバイスに対応したデータ受信命令です。
指定したエンドポイントにあるデータをBufferで指定したバイト型配列に取り込みます。
Converterは、USBINが実行される前に、あらかじめ受信するバッファのサイズを指定しておきます。バッファに転送されるバイトサイズを指定します。
Labelはエンドポイントにデータを受信していない場合にジャンプするジャンプ先を指定します。

本命令はPIC18F2550や18F4550などUSB2.0通信機能を搭載したPICマイコンで通信を実現しますが、通常のシリアル通信と比べかなり複雑で通信手順も煩雑です。PBPではUSBフレームワークの中でもCDC(Communication Device Class)クラス及びHID(Human Interface Class)クラスの2つをサポートします。

USB関連命令の使用方法については本書の"USB2.0通信機能の使用方法"の項目をご覧下さい。

9-68、USBINIT（USB2.0機能内蔵のデバイスにおいてUSBモジュールの初期化）

ＵＳＢＩＮＩＴ

PIC18F4550などUSB2.0機能搭載デバイスのUSB2.0通信ユニットを初期化します。
USB機能を使用するプログラムを作る場合にはプログラムの先頭部分で記述します。

USB関連命令の使用方法については本書の"USB2.0通信機能の使用方法"の項目をご覧下さい。

9-69、USBOUT（USB2.0機能内蔵のデバイスにおいてUSB経由でデータを送信）

ＵＳＢＯＵＴ Endpoint、 Buffer、 Count、 Label

USB2.0インターフェイス搭載のデバイスに対応したデータ送信命令です。
Bufferで指定したバイト型配列のデータを指定したエンドポイントから送信します。
送信するバイト数は、Countに記述します。
データ送信中などでバッファを格納するための十分な領域を確保できない場合、データは送信されず、Labelで指定したラベルへジャンプします。通常は、送信完了までUSBOUT命令を実行し続ける必要がありますので、"USBSERVICE"命令と共に繰り返し実行するようループ構成とします。

本命令はPIC18F2550やPIC18F14K50などUSB2.0通信機能を搭載したPICで通信を実現しますが、通常のシリアル通信と比べかなり複雑で通信手順も煩雑です。PBPではUSBフレームワークの中でもCDC(Communication Device Class)クラス及びHID(Human Interface Class)クラスの2つをサポートします。

USB関連命令の使用方法については本書の"USB2.0通信機能の使用方法"の項目をご覧下さい。

9-70、USBSERVICE（内蔵USB2.0ユニットをアクティブ化）

ＵＳＢＳＥＲＶＩＣＥ

USBSERVICE命令は、USB2.0通信ユニットを有効にしてサービスを開始させます。
USB通信を行う際には、最低でも10ミリ秒に1回は実行してSIE(Serial Interface Engine)を有効にしておく必要があります。この命令が正しく実行されない場合USBバスは無効となり通信はできなくなります。

USB関連命令の使用方法については本書の"USB2.0通信機能の使用方法"の項目をご覧下さい。

■USB2.0通信機能の使用方法　～はじめに～

ここではPBPにてUSB機能を搭載したプログラムを作る上で必要となる事柄について簡単に解説しています。USBについてより詳しい解説は解説書などをお読み下さい。ここではPBPでUSB関連命令を使用する際に必要な最低限の事項を解説しています。

USB2.0機能搭載のPICデバイスではロースピード(物理速度1.5Mbps)及びフルスピード(物理速度12Mbps)に対応しています。また転送モードもコントロール転送・インタラプト転送・バルク転送・アイソクロナス転送すべてに対応しています。

USBは、1台のホスト機器（通常はパソコン）と複数のノード（PICなどもこれに該当します）が通信しますが、通信はホスト機器がノードに対して定期的にポーリング（問い合わせ操作）をすることで通信を実現しています。すべてはホスト機器が支配しており、ノード同士での通信やノードが勝手に通信を行うことはできない仕組みになっています。

USBでは一般的なRS232Cなどのシリアル通信と比べ機能が豊富である一方、その通信手順は複雑でRS232Cのように手軽な通信はできません。ここでは簡単にUSBの簡単な通信の仕組みを解説します。
USBのホスト機器とノード機器が通信する場合、両者の間には物理的なケーブルの接続の他に、論理的な通信路が形成されます。これら論理的な通信路をパイプと呼んでいます。
パイプにはデバイスドライバなどの動作に必要な基本的な情報をやりとりするデフォルトパイプと、アプリケーションが実際にデータのやりとりを行うパイプがあります。パイプと言ってもこれは論理的なものであり実際には、USBケーブルの中を流れる差動電気信号に他なりません。よって難しく考えずここでは単に通信を行うためのケーブルだと思っておいてください。

ホスト機器とノード機器間で通信をする際、このパイプを通してデータを送受信するわけですが、USB機器では、このやとりとするデータを格納しておくバッファのことをエンドポイントと呼んでいます。エンドポイントというと難しそうですが、ようは送受信データを入れておくメモリ領域だと思ってください。なお、エンドポイント0番はコントロール転送用でデータ通信には使用できません。

PICではこのエンドポイントが送信用16個、受信用16個で計32個あります。これはUSB規格のなかではフルスペックです。1つのエンドポイントには8バイト～256バイトが転送サイズとして指定でき、バッファディスクリプタという設定ファイルで指定することができます。

このようにUSB通信では、エンドポイント間で通信を行うことになるわけですが、何個のエンドポイントを使うのかや、何個のパイプを使用するかなどの詳細は、デバイスディスクリプタとよばれる設定ファイルで設定することができます。
USBの1つの特徴にプラグアンドプレイ機能があります。パソコンのUSBポートにUSB機器を接続すると自動的にハードウエアが検出され、Windowsは最適なドライバを検索しインストールしようとします。
この時、デバイスを区別するためIDコードがデバイスディスクリプタによって送られ、ホスト側は機器を認識します。このコードは、ベンダIDとプロダクトIDとよばれるもので、ベンダIDは製造メーカーを、プロダクトIDは製品を個別に認識するために使用されます。

USBデバイスを製品として出荷する場合には、ベンダIDをUSB-IFという団体から購入する必要があります。このベンダID及びプロダクトIDは、デバイスドライバが機器を認識するときにも重要な役目をします。プラグアンドプレイで検出されると、Windowsは両IDが一致するデバイスドライバファイル(具体的にはinfファイル)を探します。この両IDが一致したファイルでないとデバイスドライバとしてWindowsに認識されません。

USB対応デバイスを作るには、USB規格に準拠したプロトコルソフトウエアが必須です。これらのソフトウエアセットのことをUSBフレームワークとよびます。このUSBフレームワークを使用することで、より簡単にUSBプログラムを記述することができ、またデバイスドライバも簡単な修正だけで作れるようになっています。PBPではマイクロチップが提供するすべてのフレームワークに対応はしていませんが、より簡単にプログラムを記述できるようにあらかじめ2種類のフレームワークを搭載しています。

PBPが使用できるUSBフレームワークのプログラムファイルは下記の2種類のクラスです。

・CDCクラス　(Communication Device Class)
　USBとの接続を仮想RS232C接続として認識させるクラスです。CDCクラスを実装したPICマイコンとパソコンとを接続するとパソコン側からはPICが従来のRS232Cポートと同様にCOMポートとして認識されます。PICマイコン側もUSBINやUSBOUT命令によってデータをRS232Cと同様の手軽さで通信させることができます。RS232Cをエミュレーションするクラスです。

・HIDクラス（Human Interface Device Class）
　キーボードやマウスといったヒューマンインターフェイスデバイスを扱うクラスです。
　HIDクラスの場合にはデバイスドライバが必要ないという利点があります。

主にデータ通信で使用する場合には、CDCクラスを使用することになります。

PBPでプログラムを作る場合にはあらかじめ用意されているフレームワークを使用して、それをコードにインクルードすることで、USBの専門的なプログラムをしなくても、比較的簡単にUSBでCDCクラスを用いた通信を実現することができます。

■USB2.0通信機能の使用方法　～PBPでCDCクラスを使用する～

CDCクラスはUSBポートに接続したPICマイコンをRS232Cポートに接続したようにエミュレーションする方法です。パソコンからは仮想COMポートとして認識されます。

USBは、ご存じの通りプラグアンドプレイ機能によりデバイスをパソコンと接続すると自動的にデバイスドライバーのインストールを実行しようとします。PBPでは、この"USB18"フォルダ内にCDC用のデバイスドライバ"mchpcdc.inf"も収録していますので、比較的簡単にプログラムを作ることができます。

ではPBPで実際にCDCクラスのアプリケーションを作る手順を紹介します。なお本例では使用するデバイスはPIC18F14K50としますが、その他のデバイスでも基本的には同様の方法です。参考例として、CDCクラスを用いて次のようなプログラムを作ってみることにしましょう。

1）サンプルプログラムの作成

CDCクラスを用いて、パソコンと接続し仮想COMポートを介して通信を行います。
パソコン側でCOMポートに対して任意のデータを送信すると、"HELLO"という文字列が返るプログラムを作ります。
なお、ここでは、VIDとPID、そのほかデバイス名等はデフォルトのまま(マイクロチップから配布されているフレームワークファイルの設定のまま)使用することにします。

```
#CONFIG
    CONFIG  CPUDIV = NOCLKDIV
    CONFIG  USBDIV = OFF
    CONFIG  FOSC = HS
    CONFIG  PLLEN = ON
    CONFIG  PCLKEN = ON          コンフィギュレーションビットの設定
    CONFIG  WDTEN = OFF
    CONFIG  HFOFST = ON
    CONFIG  MCLRE = ON
    CONFIG  LVP = OFF
#ENDCONFIG

include "cdc_desc.bas"           CDC ｸﾗｽ用のﾃﾞｨｽｸﾘﾌﾟﾀﾌｧｲﾙをインクルード　①

define OSC 48                    発振周波数は 48MHz に設定

vars:
    buffer var byte[16]          バッファ用の 16 バイト配列　②
    cnt var byte                 カウント用変数

init:
    ANSEL = 0 : ANSELH = 0       すべての IO ピンはデジタル IO ピン　③

    usbinit                      USB のｼﾘｱﾙｲﾝﾀｰﾌｪｲｽｴﾝｼﾞﾝ(SIE)を初期化　④

main:
    usbservice                   USB サービスを実行　⑤
    cnt = 16                     配列のサイズを代入

    usbin 3, buffer, cnt, main   USB からデータを取得し buffer 配列に格納　⑥

    buffer[0] = "H"              配列に文字(ASCII コード)を 1 文字ずつ代入　⑦
    buffer[1] = "e"
```

```
    buffer[2] = "l"
    buffer[3] = "l"
    buffer[4] = "o"
    buffer[5] = 13
    buffer[6] = 10
    buffer[7] = 0                    データの最後には Null を代入

outloop:
    usbservice                       データ送信のために USB サービスを実行
    usbout 3, buffer, 8, outloop     buffer 配列のデータを USB で送信　⑧

    goto main                        main ルーチンに戻る
```

2）ソースプログラムを保存しましょう

プログラムが書けたらこのファイルを"Save as“で名前を付けて保存しますが、インクルードファイルと同じディレクトリーに保存する必要があります。
このプログラムではCDCクラス用のデバイスディスクリプタファイル"cdc_desc.bas"ファイルインクルードしていますので、このファイルと同じディレクトリに保存します。

このファイルが存在するディレクトリは下記ですのでソースプログラムもここに保存してください。

C:¥Program Files (x86)¥PBP3_1¥USB18

メニューバーの"File"→"Save As"をクリックして上記ディレクトリに保存してください。

コンパイルを行います。コンパイルは他のプログラムと同様です。
コンパイルされると、HEXファイルが上記のディレクトリに生成されます。

3）デバイスドライバーのインストール

書き込みが完了したPIC18F14K50をパソコンのUSBポートと接続するとプラグアンドプレイによってデバイスドライバーのインストールが開始されます。
通常自動的にデバイスドライバーはインストールされますが、もし自動でデバイスドライバーがインストールできない場合には手動でドライバーをインストールします。デバイスドライバーファイル(mchpcdc.inf)は、ソースファイルを保存したのと同じ下記のディレクトリにあります。

C:¥Program Files (x86)¥PBP3_1¥USB18

デバイスマネージャで確認して上記フォルダを指定してドライバーをインストールしてください。

4）プログラムを動作させてみましょう

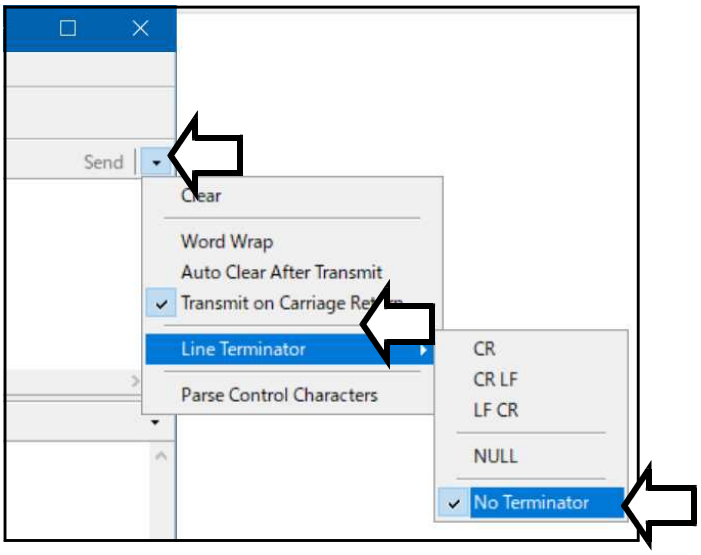
PIC18F14K50はWindowsにて仮想COMポートとして認識されています。
COMポート番号はデバイスマネージャを開いて、"ポート(COMとLPT）"のツリーで表示されますので確認してください。

ここで表示された、"USBシリアルデバイス(COMx)"のCOM番号を控えておいてください。

正しく通信ができるかどうかを確認しましょう。
シリアル通信用のターミナルソフトを起動します。MCSPのメニューバーの"View"→"Serial Communicator"を起動します。
通信の設定は下記のようにセットします。

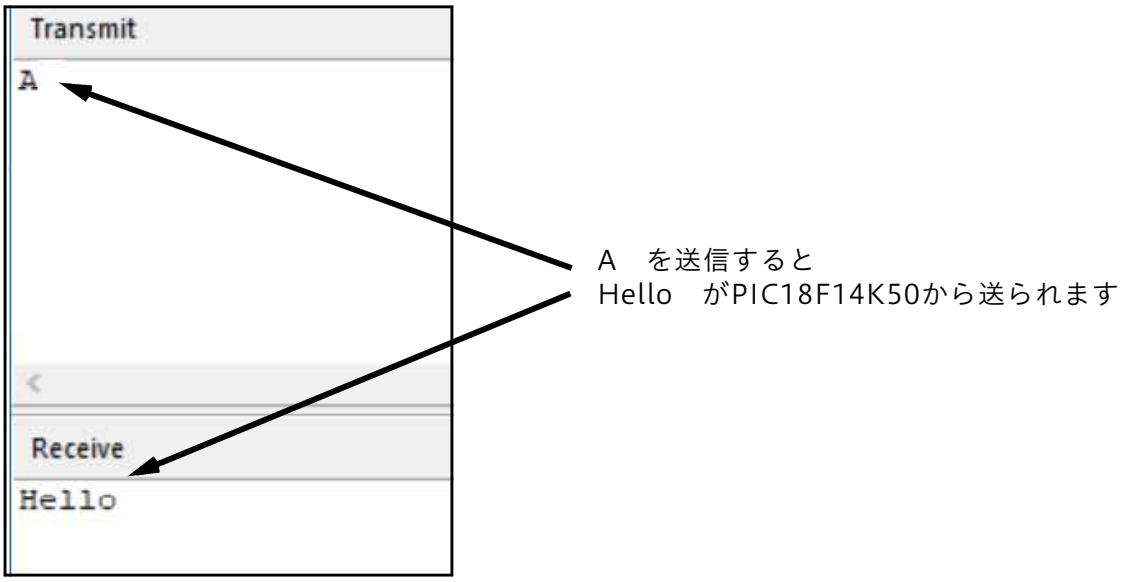
・Port Number　　デバイスマネージャで確認したCOMポート番号
・Baudrate　　9600bps
・Parity　　None
・Byte Size　　8
・Stop Bits　　1

設定ができたら送信するデータの終端をなくします。
"Send"の隣にある▼マークをクリックして"Line Terminator"→"No Terminator"にチェックを入れてください。



"Connect"ボタンを押してポートを開きます。

"Transmit"のところに何か1文字、例えば A などを入力してEnterキーを押してください。
すぐに"Receive"のウインドウに Hello と表示されれば成功です。Hello の後ろにはキャリッジリターンとラインフィードが挿入されていますので自動的に1行ずつ改行が入ります。



5）プログラムの解説

① USB通信をする場合には様々な設定ファイルが必要です。
一番重要なのはデバイスディスクリプタファイルです。ここにはUSB通信に関する基本的なことが書かれています。どんなクラスで使うかやUSB機器としてのPIDやVID、その他通信に使うエンドポイントの情報、データのサイズなどの定義情報などがディスクリプタファイルには書かれています。これはあらかじめ用意されているファイル"cdc_desc.bas"を使いました。
別のファイルをインクルードする場合には include を使います。

② usbin及びusbout命令ではデータを配列に格納します。
バイト型配列を宣言して使用します。ここでは16バイトの配列を用意しました。

③ PIC18F14K50ではANSELとANSELHレジスタの2つです。ここではこれら2つのレジスタに0を代入して全てのIOピンをデジタルIOとして設定しています。

④ PIC18F14K50などのUSB機能内蔵デバイスでは内部に通信を制御するシリアルインターフェイスエンジン(SIE)が搭載されています。USB機能を使用する場合にはSIEを初期化するためプログラムの先頭部分で usbinit を実行します。usbinit命令はUSBバスが設定され有効になるまで待機します。

⑤ USB通信のサービスを有効にするためusbservice命令を実行します。このusbservice命令はUSBからデータを送信時・受信時に必ず全データを処理できるまでの間繰り返し実行する必要があります。
USB通信では通信はホスト機器からのポーリングによって行われるためPIC側は定期的にホストからの問い合わせに応答する必要があります。ここで作ったプログラムでは割込は使っていないのでusbservice命令を少なくとも10ミリ秒毎に実行しておく必要があります。
このusbservice命令を定期的に実行できないとUSB通信が切断されてしまいます。プログラムの構造に工夫が必要です。

⑥ usbin命令はUSBから（厳密にはUSBドライバーで定義されたエンドポイントから）データを受信し、引数で指定した配列に格納します。
受信サイズはcntで指定したサイズ分用意されます。このバッファサイズは厳密でなくてもかまいませんがUSBリソースを有効に使用するために適切なサイズを設定する必要があります。
ここでは配列の最大サイズ16を指定しています。

```
USBIN Endpoint,Buffer,CountVar,Label
```

データがない場合には指定したLabelにジャンプします。
このプログラムではデバイスディスクリプタファイルに記述された内容に即してエンドポイントは3としています。

⑦ 送信する文字列を作るためあらかじめ配列に文字列を代入しています。
文字列を作る場合には必ず最後にNull（0)を入れておく必要があります。これをヌル終端文字列といいプログラミングではよく使われます。

⑧ usbout命令は、bufferで指定した配列の内容をUSBで（厳密にはUSBドライバーで定義されたエンドポイントから）送信します。送信するデータサイズを引数で指定します。
このプログラムでは16バイトの配列を用意していますが実際は8バイトしか送信しないのでcntに8を指定しています。

```
USBOUT Endpoint,Buffer,Count,Label
```

もしUSBバッファが送信待ちでこのデータのためのスペースがない場合にはデータは転送されず転送されるまでLabelで指定したラベルにジャンプして転送を待ちます。

9-71、WHILE....WEND（条件ループ）

WHＩLW Condition
　　Statemant.....
WEND

Conditionの式が真の間だけ（条件と一致するまで）Statementの内容を繰り返しループします。
Conditionには比較条件が入ります。

```
例      i = 1
        WHILE  i  <=  10                          '変数iが１０以下の時ループします
               Serout  0、N2500、["No:" #i 、 13 、 10]
               i = i + 1                          '１ずつインクリメント
        WEND
```

9-72、WRITE（内蔵EEPROMに値を書き込む）

WRＩTE Address,Var

PIC12F/16F/18Fシリーズのようなの内蔵EEPROM搭載タイプのPICマイコンの内蔵EEPROMに値を書き込みます。指定した番地Addressに、byte変数Verの内容を書き込みます。値は8ビット長（byte型）です。
※内蔵EEPROMはデータシート等ではデータメモリーと記載されています。

12ビットコアでデータメモリーを搭載したPIC12F519、PIC12F1822などのデバイスでは、WRITE命令でデータを書き込みできますが、データを書き込む前に必ずERASECODE命令で、データを書き込む番地のデータを事前にイレースしておく必要があります。詳しくはERASECODE命令の項をご覧ください。

書き込んだ値はREAD命令で読み込むことができます。

Address	変数又は定数（0－255）数値、番地を指定します。
Var	EEPROMに書き込むデータを格納するbyte型変数です。

WRITE 5 , B0

上記の例では、PICマイコン内蔵のEEPROMの5番地に変数B0の値を書き込んでいます。
EEPROMは不揮発性ですので、電源を切断しても書き込んだ値は保持されます。

書き込みには10ミリ秒の時間を要します。よって、WRITE命令実行後は必ずPAUSE命令等で10ミリ秒以上待ってから次の処理を行うようプログラムしてください。
プログラムで割込を使用している場合には必ずWRITE命令を実行する前にレジスタによって割込を無効にしておく必要があります。WRITE命令でメモリーにデータを書き込んでいる最中に割込が発生するとデータの書き込みに失敗します。WRITE命令が完了したことを確認した後、割込を有効にするようプログラムする必要があります。

9-73、WRITECODE（コードスペースに値を書き込む）

ＷＲＩＴＥＣＯＤＥ Address,Value

コードスペース（プログラムメモリー領域）に、データを書き込む命令です。引数Addressにはデータを書き込む番地、Valueには書き込む値を指定します。
この命令はプログラムメモリーに値を書き込む命令であり、データメモリー(EEPROM)にデータを書き込む命令ではありません。PIC18Fシリーズやいくつかの PIC16Fシリーズでは、プログラムを実行している時にプログラムメモリーにアクセスしてデータを書き込んだり、読み込んだり、イレースしたりできます。この命令は、その機能を利用してプログラムメモリーにデータを書き込みます。

但し、プログラムメモリーにデータを書き込むという性質上、誤った使い方をするとプログラムそのものを破壊しプログラムが実行不可能になったり、予期しない動作をするなど致命的な問題を引き起こす原因となります。プログラムの破壊につながりますので、この命令の使用には十分な注意とデバッグ、テストが必要です。データメモリーだけでは足りないような大きなデータを保管したい時に便利ですが、それ以外の場合（小さなデータの記憶）には、この命令は使わずに、データメモリーを使用するWRITE命令を使った方が安全です。

コードスペースにおけるデータの書き込みができるサイズはデバイスのファミリーによって異なります。例えばPIC16Fシリーズの場合には14ビットサイズです。よって書き込めるデータも14ビットまでとなります。PIC18Fシリーズの場合には基本的にバイトサイズ（8ビット）です。使い方によってワードサイズ（16ビット）も可能です。

コードスペースへのデータ書き込みにはPIC毎に決まりがあり、必ず1ブロック単位での書き込みが要求されます。1ブロックのサイズはPICのデータシートに記載があります。例えばPIC18F452の場合、書き込み時の1ブロックあたりのサイズは8ワードと決まりがあります。（PIC16F1939など多くのデバイスで書き込みのための1ブロックは8ワードですが、使用する前に必ずデータシートでブロックサイズを確認してください。）よって、この命令で書き込みを実行する場合には、必ず1ブロックサイズを一度に書き込まなければなりません。もしこのブロックサイズを無視して書き込みを行うと、データは書き込まれずすべて破棄されます。例えば下記例は書き込みブロックサイズが8ワードのPICにデータを書き込む場合です。

```
add  var  word
mydata  var  word

For add = $180 to $187
  Writecode add, mydata
Next add
```

たとえ書き込むデータがたったの"1"だったとしても必ず1ブロックまとめてデータを書く必要があります。よってWRITECODE命令を1回だけ使うことはなく、1ブロック分になるように複数回実行します。

なお、PIC18Fシリーズや一部のPIC16FシリーズではWRITECODEを実行する前に、あらかじめ書き込みを行うコードスペースのイレースをしておく必要があります。イレースはERASECODE命令で行います。書き込み前にイレースが必須のデバイスでは、イレースをしないで書き込みをしてもデータは書き込まれません。また、イレースの際の1ブロックのサイズは、書き込みの際の1ブロックのサイズと異なりますので、十分注意してください。
プログラムで割込を使用している場合には必ずWRITECODE命令を実行する前にレジスタによって割込を無効にしておく必要があります。WRITECODE命令でデータを書き込んでいる最中に割込が発生するとデータの書き込みに失敗します。WRITECODE命令が完了したことを確認した後、割込を有効にするようプログラムする必要があります。

10. 製品のサポートについて

10-1、サポート掲示板とメーカーへのサポート依頼

PBPのサポート情報は、当方のサポートページに記載されており、随時更新されております。
皆様からよく寄せられるご質問事項を記載しておりますので、まずはこちらをご確認ください。サポートページは、当方webサイト（http://www.microtechnica.net/）より「サポート情報」をクリックし、製品一覧の項目から PICBASIC PRO Compiler を選択すると表示されます。

PICBASIC PRO Compilerの開発メーカーが運営するPICBASIC PRO Compiler用のユーザーフォーラムがあります。世界中のユーザーが日々活発に意見のやりとりをしていますので何か疑問があることがあればまずここをご覧ください。

■PBPフォーラム：　　http://www.picbasic.co.uk/forum/forum.php

※上記サポートフォーラムページよりカテゴリ"mel PIC BASIC Pro"又は"PBP3"をクリックするとPBPに関するサポートフォーラムが開きます。記事の検索は画面右側の"Search Forum"をクリックするとキーワードが設定できますので、検索したいキーワードを入力して検索してください。

本製品のプログラミングの技術的なご質問や、各種命令の使い方、プログラムの実装上のご質問等につきましては、当方マイクロテクニカでは行っておりません。ご質問の際には、まず上記のサポートフォーラムをご利用ください。最初に関連記事を検索し、同様の事例等が過去に投稿されていないかを必ずご確認ください。膨大なデータベースのため、ほとんどの内容は過去の記事に掲載があります。
下記に同様のご質問等が掲示板に投稿されていない場合には、新規に記事を投稿し、世界中のユーザーからの回答をお待ちください。素早い回答が得られます。
なお、個別に開発元のmicroEnginerring Labs社に直接お問い合わせ頂くことも可能です。その場合には、下記の要領でお問い合わせください。なおお問い合わせはすべて英語となります。

■メーカーサポートEメール：　support@melabs.com

上記サポートにメールを送る場合には、ご使用の環境及びPBPのバージョンなどサポートに必要と思われるできる限り詳しい情報を記述してください。また、ソースコードを添付する場合には、必ず拡張子をtxtに
変更して添付してください。メール本文にソースコードをコピーして貼り付けることはしないでください。
メーカーのサポートは迅速に問題解決のためのご案内をしております。

10-2、E-mail又はFAXサポート、アップグレードについて

当方ではPICBASIC PRO Compilerについてのお問い合わせを随時受け付けておりますが、次のような内容のご質問にはお答えいたしかねますのであらかじめご了承ください。

・お客様の作られたプログラム（以下、プログラムと表記）の内容に関するご質問。
・お作りになったプログラムが正しく動作しない場合の原因究明。（プログラムの検証）
・プログラム自体の不具合箇所の発見依頼や修正依頼。
・PICマイコン自体に関するご質問やハードウエアに関するご質問。
・他社製品が混在したプログラムに関するご質問。

当方ではお客様の記述されたコードの検証や問題解決に関するサポートは行っておりません。
当方ではPBPの基本的な使い方やインストール方法、ソフトウエアの設定方法のみのサポートを行っております。ご理解の上ご了承くださいますようお願い申しけ上げます。プログラミングに関する内容や技術的なご質問は、当方ではなく開発元であるmicroEnginerring Labs社にお問い合わせください。お問い合わせ方法は131ページをご参照ください。

インストール方法のサポートメール：　　support@microtechnica.net

混雑状況により回答が遅れる場合もございます。あらかじめご了承ください。

製品がアップグレードされた場合には当方のwebページにてご案内致しますので、定期的に当方webページをご覧頂けますようお願いいたします。
なお、アップグレードにかかる費用についてはその都度異なりますので、アップグレード版がリリースされた際に当方のwebページでご案内いたします。

アップグレードに際しては、製品版をお届けした際に添付されていたアクティベーションキーが必ず必要です。紛失された場合にはアップグレードができません。またアップグレードに際してはメーカー及び当方へユーザー登録をしている必要があります。必ずユーザー登録を行って頂けますようお願い致します。

10-3、免責事項

この日本語説明書は英文説明書を元に作成されたものですがすべての部分を日本語訳したわけではなく、必要と思われる部分を要約して翻訳したものです。よって本書の内容は英文説明書の内容とは一部異なる部分がありますことをご了承下さい。

本製品は開発言語用コンパイラという製品自体の性格上、サポートの対象範囲及び製品に関する保証及び責任範囲を限定させて頂いております。当方ではお客様の記述されたコードに関する責任及びサポートはサポート範囲外となります。プログラムが正しく動作するかについては、お客様の責任の下十分な検証を行って頂けますようお願いいたします。

当コンパイラが英語版説明書や日本語版説明書に記載された命令を用い、かつ構文ミスや表記ミス、設計ミスのないソースプログラムを正しくコンパイルしHEXファイルを生成するまでの過程のみを当方の責任の対象としております。ただし、プログラムには潜在的な欠陥（バグ）が含まれることがあり、開発元及び当方では、PICBASIC PRO Compiler並びに、その他の付随する製品についてのバグについては、その責を負わないものといたします。

当方及びコンパイラの製造元は本コンパイラ及びそのすべての付属品によって生じたいかなる被害や損害、懲罰的損傷、経営上の損失や損害についてもその責を一切負わないものとしますので、あらかじめご了承ください。

本製品に関する最新情報やアップデート情報は当方のホームページでご確認ください。

http://www.microtechnica.shop/

PICBASIC PRO Compiler Ver.3.1日本語説明書
2023年2月版
発行所／マイクロテクニカ
発行者／吉田靖幸

2023 (C)Copyrights Microtechnica All rights reserved.
発行所及び発行者の許可無く本書の一部又は全部を複写、転写、転載することはできません。