レッスン1： ハローワールド（LED 点灯）

ほとんどのプログラム入門の最初のプログラムは、”Hello World” とコンピュータから挨拶の文字を表示させるのがしきたりのようですが、このレッスンでは、ターゲット上の0～7の番号がつけられている8個の LED のうち、0番の LED を点灯させて、挨拶がわりにしています。PIC にはたくさんの入出力ピン（I/O ピン）がありますが、このレッスンではその設定方法を学びます。

新しい命令

BSF	Bit Set
BCF	Bit Clear

各 LED は I/O ポート PORTD の各ピン RD0～RD7 に接続されています。これらの I/O ピンのひとつをハイに駆動する（0 V から5 V にする）と LED が点灯します。I/O ポートは1ピンごとに入力あるいは出力に設定することができます。電源投入スタート時には、強制的に入力に設定されています。この設定のために、PIC 内部の TRIS という Special Function Register (SFR) のビットを’0’にすれば出力、’1’にすれば入力になります。私たちはすべてデジタル出力にしたいので、設定は次のようになります。

##プログラムリスト1##

プログラム記述方法の説明

;	セミコロンではじまるテキスト行はコメントとして扱われます。コメントに限り、日本語（かな、漢字）を使用してもかまいません。
#include	PIC16F887 で設定可能なすべての SFR の定義をプログラムに融合させます。また、有効なメモリ領域の定義をします。これらの定義はデバイスのデータシート上の名称と一致しています。使用する PIC ごとに別の定義ファイルが用意されています。プログラムの記述上省略できません。
__CONFIG	頭のアンダーバーは2個連続させます。 コンフィギュレーション・ワードを定義します。詳細な内容はいずれ必要なときにするとして、ルールとして PIC16F887 では、このとおり記述します。
org 0	アセンブラに何番地からスタートするプログラムコードを生成させるかを伝えます。どこからスタートするようにしてもよいですが、PIC がリセットしてスタートする番地は、0番地ですから、ここでは0にしています。
Start:	プログラム行にはラベルをつけることができます。ここでのラベルは0番地と同じ意味を持っています。ここではあまり意味はありませんが、プログラムの実行順を変更する場合に、次に実行する番地をラベルで指定することができます。 なお、ラベルは自由につけられますが、PIC アセンブラで予約されたラベルがあり、これはラベルに使用できません。見た目にわかりにくいラベルはバグのもとになりますから、自分にとってわかりやすいラベルにしましょう。 ただし、日本語は使えません。
BCF TRISD,0	あるファイルレジスタ中の1ビットをクリアすることをプロセッサ（PIC)に伝える命令です。TRISD は PORTD の各ビットを入力にするか出力にするかを設定するレジスタです。このプログラムでは、RD0 に出力するのが目的ですから、0を設定します。ビットをクリアする、つまり、0に設定することになります。
BSF PORTD,0	あるファイルレジスタ中の1ビットを1または0に設定するための命令です。

この命令によりPORTDのビット0がハイ状態になり、0番のLEDが点灯します。

Goto $　　現在の命令の実行を継続するようにプロセッサに伝えます。結果的にプログラムの進行はその場で足踏み（停止）します。

レッスン2：　点滅（遅延ループ）

最初のレッスンは、ひとつのLEDをどのようにして点灯させるかを示しました。このレッスンはそのLEDを点滅させる方法を示します。このレッスンはレッスン1からささいな変更のように見えますが、さらにいくつかの命令を探求する機会を与えてくれます。

新しい命令

CLRF　　ファイルレジスタをクリアします
INCF　　ファイルレジスタをインクリメントします
DECF　　ファイルレジスタをデクリメントします
INCFSZ　　ファイルレジスタをインクリメントし、もし結果がゼロならば次の命令をスキップします
DECFSZ　　ファイルレジスタをデクリメントし、もし結果がゼロならば次の命令をスキップします
GOTO　　プログラム中のどこか新しい場所にジャンプします

インクリメントとは、1ずつ増加することです。（A=A+1） デクリメントはその逆で、1ずつ減少することです。（A=A-1)

##プログラムリスト2##

BCF命令を追加する一方、プログラムをループさせることにより、LEDは点滅します。ただ、そのままでは点滅があまりにも高速のため、点滅には見えず、ぼんやり光っているように見えます。このリストのループには4つの命令の実行時間しかかかりません。最初の命令でLEDが点灯します。2番目の命令でLEDが消灯します。GOTO命令には、2命令実行時間がかかりますが、それでも全体の25%にすぎません。PICマイクロコントローラは1秒間に100万回の命令を実行します。（内蔵クロック4 MHzで動作した場合）このレートで、点滅を目に見えるようにするためには、スローダウンする必要があります。それは遅延ループを用いることによって可能です。

インクリメントまたはデクリメント　ファイルレジスタ

INCFSZおよびDECFSZ命令は、ファイルレジスタの値から1、加算、あるいは減算して結果がゼロのときに次の命令をスキップします。遅延ループにおけるひとつの使用方法をプログラムリスト3に示します。

##プログラムリスト3##

“GOTO Loop”（プログラムリスト#3のShort Loop）はプログラムの進行をもどし、実行を繰り返します。このループには3命令実行時間つまりデクリメントのために1、そしてGOTOのために2、かかります。さらに、カウンタは強制的に256回ループを回しますから、トータル768命令実行時間（768 μS）かかります。

それでも、まだ、人間の目には早すぎます。そのため、この最初のループの外側に、さらに、2 番目のループを追加することによって、スローダウンすることができます。（プログラムリスト#3 の Long Loop)

外側のループの実行1回のために、内側のループ768 μS プラス3命令実行時間かかり、しかも、その外側のループが256回実行されるため、
(768+3)*256=197376μS=0.197s となります。

レッスン3： ローテイト（LED 点灯位置の移動）

レッスン1と2により、LED ひとつを点灯させ、遅延ループによって、点滅させる方法を示しましたが、このレッスンではローテーションを加えます。まず、8番目の LED を点灯させ、それを、7番目、6番目と1番目までシフトさせ、また、8番目にもどします。

新しい命令

MOVLW　　WREG に数値をロードする
MOVWF　　WREG の内容をファイルレジスタに移す
MOVF　　ファイルレジスタの内容を WREG に移すか、もとのファイルレジスタに戻す（注）

RRF　　ファイルレジスタを右にローテイト
RLF　　ファイルレジスタを左にローテイト

（注） ファイルレジスタ自身に内容を移動させるとは、一見、何も実行しないのと同じです。しかしながら、その値を反映したゼロフラグをセットできるという便利な副次的効果があります。言い換えれば、MOVF fileregister,f は、WREG の内容に影響することなく、その値がゼロかどうかテストする方法として便利なのです。

ローテイトプログラムフロー

・はじめに、I/O ポートと表示保存領域 Display を初期化する
・Display の値を I/O ポートにコピーする
・しばらく遅延させる
・表示をローテイトする

##ローテイトのプログラムフロー図##

ローテイト

ローテイト命令（RRF または RLF)はファイルレジスタのすべてのビットを、キャリービットを通して、右、または左に1だけ位置をシフトさせます。キャリービットはバイト（ファイルレジスタは8ビットのバイトデータです）にシフトインされ、バイトからシフトアウトしたビットはキャリーに受け取られます。表示バイトの中に望まないビットが生成されないように、キャリービットはローテイトする前にクリアしておくべきです。また、キャリービットは表示バイトが空の場合の標識にもなっています。空のときには、ビット7に1を再挿入します。

PIC マイクロコントローラは2つのローテイト命令を有しています:ローテイト・レフト（RLF)とローテイト・ライト（RRF)　これらの命令はファイルレジスタの内容とキャリービットを1だけ位置をローテイトします。なお、キャリービットは STATUS レジスタの中にあります。

##ローテイトのプログラムリスト##

レッスン４:　Analog-to-Digital

このレッスンは、ADC を設定し、ターゲット上のポテンショメータによってコントロールされる電圧を読み取って A/D 変換し、上位8ビットを表示させる方法を示します。

PIC16F887 には14チャンネルの10ビット分解能のアナログ・ツー・デジタル　コンバータ（ADC)が組み込まれています。そのコンバータは、PIC デバイスの電源電圧（VDD)または外部の基準電圧を参照することができます。ターゲットでは PICkit2 プログラマから供給される VDD を基準電圧として参照するようになっています。ADC からの読みと、基準電圧には、比率関係があります。
ADC = V/VREF*1023

ADC からの読みを V（ボルト）にもどすには換算が必要です。
V = ADC/1023*VREF
換算式右辺の3つのパラメータは定数で、あらかじめ計算しておくことができます。これにより、実際に割り算を実施する必要はなくなりますが、実行中に式を解くためには、固定小数点または浮動小数点の掛け算が依然として必要になります。

しかしながら、多くの場合、センサの読みのような場合、電圧の計算は最初のステップです。センサからの意味のあるデータを計算するためには、付加的な算術が必要です。例えば、サーミスタ（温度計）の読みの場合、電圧の計算は温度を得るための最初のステップにすぎません。

線形テーブルルックアップやピースワイズリニア補完を含め、ADC 値を変換するには、そのほかいくつかの方法があります。（前者については、今後、応用例でトライする予定です。後者は上位機種の PIC でないと実現がむずかしいと思われます。）

以下がこのレッスンでチェックするポイントです。
・PORTA をアナログ入力に設定　TRISA<0>=1,ANSEL<0>=1
・ADCON1 中の正規化方法（左詰め／右詰め）、および比較電圧ソースの選択
・ADCON0 中のクロック分周比とアナログチャンネルの選択

ADCON1

ADCON1 レジスタにはレジスタ ADRESL と ADRESH から読み出された16ビットの結果を10ビットの結果として割付ける方法を指定します。左詰めで正規化する設定では、ADRESH の読みである8ビットが上位となり、ADRESL の読みであるビット7とビット6が下位となります。ADCON1 には、さらに基準電圧源である VREF+および VREF-の選択も指定します。VREF-はゼロの電圧です。VREF+は最大値（1023）の電圧です。私たちはそれぞれについて、PIC16F887 の VSS と VDD を指定します。

ADCON0

ADCON0 は ADC のオペレーションをコントロールします。ビット0で ADC モジュールをオンにします。そして、ビット1でコンバージョンをスタートします。ビット＜7:6＞でプロセッサクロックとコンバージョンスピードの比率を選択します。ビット＜5:2＞でどのチャンネルで ADC をオペレートするか選択します。プロセッサクロックとコンバージョンスピードとの間の比率は重要です。なぜなら、ADC には少なくとも1ビットあたり1．6 μS 必要だからです。もし、クロックスピードが速すぎると精度が落ちます。プロセッサクロックスピードが上がると、1ビットあたり1．6 μS 以上のコンバージョンスピードをキープするために、ますます大きなディバイダが必要になります。4 MH zの場合、最低8:1以上の比率で最速のコンバージョンレートが得られます。このとき、1ビットあたり2 μS のコンバージョンスピードになります。

本稿の元になっている User's Guide では、データシート（DS41291)の Analog-to-Digital の項にある表 ”TAD VS. Device Operating Frequencies” を参照するように勧めていますが、ここでは、代わりに AD 変換の概念を図解します。こちらも参照して理解を深めてください。

##Ad 変換の概念図##

このレッスンのためには、ADC をオンにして、RA0 ピンの AN0 チャンネルを指定する必要があります。（ターゲットでは、ポテンショメータが RA0 に接続されているからです）
ADC には、チャンネルを切り換えてから、ADC サンプリングキャパシタが安定するまで、5 μS 必要です。最後に、私たちは ADCON0 中の GO ビットをセットしてコンバージョンをスタートすることができます。そのビットは DONE フラグとしても使用されます。すなわち、ADC は、コンバージョンが完了すると、その同じビットをクリアします。そして、答えは ADRESH:ADRESL に得られます。このレッスンでは、結果の上位8ビットを取り出して PORTD に接続されている表示用 LED にコピーします。

レッスン5： バリアブル・スピード・ローテイト

レッスン5は、ローテイションスピードをコントロールするのに、アナログ・ツー・デジタル・コンバータ（ADC)を用いることによって、レッスン3とレッスン4を結合したものです。

新しい命令

BTFSS　　ファイルレジスタのビットテストをし、もし、1にセットされていたら、次をスキップ
BTFSC　　ファイルレジスタのビットテストをし、もし、0クリアされていたら、次をスキップ

コンバージョンはメインループのすべてのパスを通して動きます。結果が外側のループの長さをコントロールします。

##バリアブル・スピード・ローテイトのリスト

##バリアブル・スピード・ローテイトのプログラムフロー

レッスン6： スイッチの振動抑制

メカニカルスイッチは、ほとんどのコンピュータ、マイクロプロセッサ、そして、マイクロコントローラの応用上、実用面から重要で広範な役割を担います。メカニカルスイッチは安価で単純でかつ信頼

できるものです。しかしながら、スイッチは電気的に、たいへんノイズが多いと言わざるを得ません。開閉には明らかなノイズがともない、クリーンな電気的伝達になることはまれです。最終的にスイッチが安定状態になるまでの間、接続には多くの、たぶん、数百回の開閉を引き起こします。

この問題はスイッチ振動として知られています。多くの断続動作はスイッチの接点がお互いに弾き合うことで起こります。ビリヤードの玉がお互いに弾く様を思い描いてください。硬くて弾力がない材質は、動きの運動エネルギーを吸収しません。代わりに、そのエネルギーは、ビリヤードボールが、お互いに反発しあう動作における、摩擦にくりかえし消費されます。硬い金属スイッチの接点は同じ原理で反発します。しかも、スイッチの接点は完全になめらかではありません。接点が互いに動くとき、その表面の不完全さと不純物のために、電気的接続が妨げられます。その結果がスイッチの振動です。
不確かなスイッチ振動の帰結としては、ほんの困った程度から、破滅的なものまで幅広いにちがいありません。例えば、TV チャンネルを切り換えるときを想像してください。次のチャンネルに切り換えるつもりが、ひとつ、ふたつ飛び越して選択されてしまうようなことです。これは設計者が回避努力すべき場面です。
スイッチ振動は初期のコンピュータ以前からの問題のままです。伝統的な解決方法には、抵抗－コンデンサ回路を通過させる、あるいは、リセッタブル・シフト・レジスタを通過させるというようなフィルタがありました。（＃＃図＃＃） これらの手法は今でも有効ですが、それらには付加的部品の費用やボードの場所が必要となります。ソフトウェアによる振動抑制であれば、これらの付加的費用発生をおさえられます。
スイッチ振動抑制の最も簡単な方法のひとつが、信号が安定するまで、あるいは、信号がもう振動を検出しなくなるまでサンプルし続ける方法で、スイッチをサンプルすることです。どの程度サンプリングし続ければよいかは、少し調査が必要です。しかしながら、5msは通常十分な長さで、利用者がそれを認識できないほど十分に速い反応速度です。
レッスン6では、多くの連続的な状態変化を、1msの率で待ちながらそのラインをサンプルする方法を示しています。その状態変化を単純に5回数えて、毎回もとの変化のない状態である限り、カウンタをリセットします。
ターゲットのスイッチはそれほどは振動しませんが、この方法なら、どんなシステムのスイッチであっても、その振動抑制に、応用がききます。

簡潔なスイッチ振動抑制プログラムのフロー

レッスン7： リバーシブル・バリアブル・スピード・ローテイト

レッスン7はレッスン5と6を結合したものです。ボタンが押されたときローテイション方向が逆転し、ポテンショメータの調節でローテイション速度をコントロールします。

プログラムはローテイション方向を追跡する必要があり、逆方向へのローテイションのために、新しいコードが必要です。

レッスン5は右にローテイトし、1がキャリービットに入るのをチェックして、シーケンスをリスタートする時期を決定しています。レッスン7では、私たちは左へのローテイトと、display のビット7に1が入るのをチェックする必要があります。1が display のビット7に現れたら、1をビット0の場所に挿入します。

リバーシブル・バリアブル・スピード・ローテイトのプログラムリスト##

レッスン8： 関数コール

レッスン8はリバーシブル LED を示しますが遅延ループは、関数として新たに書き直します。

新しい命令

CALL	関数あるいはサブルーチンを呼び出す
RETURN	関数あるいはサブルーチンを終了する
RETLW	関数あるいはサブルーチンを終了する（W レジスタに関数の戻り値）

関数あるいはサブルーチンは CALL 命令で呼び出され、RETURN あるいは RETLW 命令で終了します。RETURN はもとのプログラムの CALL の次の場所へ戻します。RETLW も呼び出されたプログラムに戻しますが、WREG に定数を持ってもどります。

Mid-range PIC マイクロコントローラ・デバイスの CALL スタックは8つのリターンアドレスを保持することができます。
もし、9個めの CALL がされたならば、それが最初のひとつに上書きされ、プログラムは、すべての道を RETURN できなくなります。

引数の受け渡し

サブルーチンへの引数はいろいろな方法で受け渡せます。WREG は1バイトを渡す場所として便利です。また、もし他の目的で使われていなければ、FSR で別のバイトを渡すことができます。もし、もっと多くのデータを渡さなければならないならば、バッファを確保しなければなりません。（レッスン11で説明されています）

遅延機能がサブルーチンに引き継がれるとき、ADC 結果は WREG に移しておいて、CALL 命令で Delay サブルーチンに制御を渡します。RETURN 命令は CALL の次の命令である MOVLW に制御を渡します。

ファンクションコールのリスト

レッスン9： タイマー0（TMR0)

TMR0 はプロセッサに組み込まれたカウンタのひとつです。それは、プロセッサ・クロック・サイクルや、外部のイベントをカウントするのに使えるでしょう。レッスン9では、TMR 0を、インストラクションサイクルのカウントと、それが、あふれたときに、フラグをセットするように設定します。これにより、プロセッサを、単純な、遅延のためのサイクルカウントから開放して、もっと意味のある仕事をさせることができます。

TMR 0はオプションのプリスケーラを持つ、8ビットのカウンタです。プリスケーラは TMR 0の前にあり、256分周まで設定できます。

##タイマー0の簡略図##

TMR0 はスペシャル・ファンクション・レジスタ（SFR)で、プログラムからリードでき、また、書き換えできます。プリスケーラは SFR ではありませんので、プログラムからリードも書き換えもできません。た

だ、TMR0 に書き込むとプリスケーラはクリアされます。

TMR0 はプロセッサをドライブする同じクロックにつなぐこともできるし、外部のイベントにつなぐこともできます。プロセッサクロックによってドライブされるときには、すべてのインストラクション・サイクルに1回、インクリメントします。（プリスケーラを使用している場合には、プリスケーラがカウントされるだけですから、注意してください）
プロセッサが、遅延ループのために無駄に使われるよりも、本来の問題処理に使えるので、TMR0 の利用は時間を知る方法として重宝するし、遅延ループよりも良い方法です。
プリスケーラは OPTION_REG を通じて、設定できます。図を参照してください。

OPTION_REG を通じてプリスケーラを設定する図

レッスン9では TMR0 の最大の遅延になるように、プリスケーラを含めて TMR0 を設定します。そうすると、TMR0 のフラグは65536 μS（0．0000001 Sec*256*256）ごとに、あるいは、約1秒に15回、セットされます。メインプログラムはループの中にじっとしていてロールオーバを待ち、そうなったときに、表示を1インクリメントして、また、ループにもどります。（ここでロールオーバとは、設定した最大時間になって、フラグがセットされたときのことです）

タイマ0のリスト

レッスン10： インタラプト

新しい命令

RETFIE　　インタラプトからリターン
SWAPF　　ファイルレジスタの桁の入れ換え

インタラプト
プログラムとは、ある入力をあたえて、それに処理を加えて、結果を出力するための手続きを記述するものだと定義しました。その処理を実行する場合、入力が入ってくる順番どおりに処理していけばよい場合、その処理にかかる時間をうまく割り振れば、順に処理していっても、必要な時間内にすべての処理が完了するでしょう。ただ、この方法だと、順番に入ってくるとは言っても、それぞれの入力があるかどうか、プログラムでモニタしながら待ち、入力があったら、その処理に移るという動作になり、そのモニタする部分で、単に入力を待つだけになる無駄な時間が発生します。

しかし、処理の中には、周期的に実行しなければならず、ちょうど、その周期にあたるときに、別の処理をしているために、実行を待たされるようなことがあると、困る場合もあります。もちろん、ひとつ、ひとつの処理の実行時間をできるだけ短くして、待ち時間が発生しないようにすることが肝心です。ただ、あらかじめ各処理の実行時間を設計どおりに作ったり、実行順も優先度に応じてスケジュールするのは、なかなか困難です。それよりも、いつ処理要求が到着しても、緊急性が高ければ、先に実行できるようにしたほうが効率的な場合もあります。その場合、現在、別の処理をしている PIC に割り込んで、その優先すべき処理要求を知らせる必要があります。それが、インタラプトです。

インタラプトとは、日本語で、割り込みです。文字通り、割り込みのことです。

PIC には、

① 割り込みを受け付ける仕組み（同時に禁止する仕組みでもあります）
② 割り込みがあったら、今実行している処理を中断して、割り込みを優先して処理できるようにする仕組み
③ 割り込み処理が終わったら、もとの処理に戻す仕組み
の3つが組み込まれています。なお、割り込み要因は1種類ではなく、PIC でも複数の割り込みを受け付けられるようになっています。たとえば、ADC の完了、タイマ0のタイムアップ、スイッチが押された場合などです。しかし、今回選択したレベルの PIC では、割り込み処理中に別の割り込みが発生しても、その割り込みは無視されます。したがって、割り込み処理は、できるだけ短時間に終了できるようなプログラムにしなければ、多くの割り込みに対応できません。また、複数の割り込み要求はすべて同一の優先順位で扱われ、ひとつの割り込み処理を起動できるだけですから、割り込み処理の中で、どの割り込みなのかを判定する必要もあります。

現在のコンテキストの保存
割り込み処理（Interrupt Service Routine=ISR)で最初にしなければならないのは、現在のプロセッサのコンテキストを保存することです。当然、メインプログラムに戻る前に、保存したコンテキストを復帰する必要があります。ISR の中で使われる SFR は保存しなければなりません。たとえば、WREG と STATUS レジスタは最低、保存しなければなりません。WREG は、はじめに保存するように心がけてください。他の SFR が WREG に移動されると、保存する必要のある値がこわされてしまうからです。PIC16F887 の各ファイルレジスタのページの最後の16バイトはバンクなしの場所ですからコンテキストを保存する場所として最適です。というのは、バンクを気にせず、どのページからもアクセスできるからです。なお、バンクなしのレジスタの場所は、部品ごと（PIC ごと）に異なります。使う部品（PIC)のバンクなしの場所は、そのレジスタマップで確認してください。

割り込み要求しているイベントの識別

次に ISR が実行しなければならないことは、何がインタラプトをかけたかを抽出することです。インタラプトの要因を判定するためには、インタラプト・フラグをチェックしなければなりません。割り込みの要因がわかってはじめて、周辺装置にサービスできるのです。

コンテキストの復帰

周辺装置のサービス（割り込み処理）が完了したならば、コンテキストを復帰して、メインプログラムにもどる必要があります。コンテキストとは、インタラプトが発生したときの SFR の状態のことです。はじめは、コンテキストの復帰は見かけより少しむずかしいかもしれません。普通のやり方は通用しないからです。なぜなら、MOVF W_Temp,w は、前の命令で復帰された Z フラグに影響するかもしれないからです。代わりに、一組の SWAPF 命令を使えば、STATUS レジスタ中のフラグに影響せずに、WREG を復帰することができます。SWAPF は上位と下位のニブルを交換します。（ニブルとは、バイトの上位4ビットと、下位4ビットのことです）最初の SWAPF でファイルレジスタのニブルを交換して、2 番目の SWAPF で再度ニブルを交換して WREG に結果を入れます。

コンテキスト復帰のリスト

最後に、RETFIE でもとのプログラムに制御をもどし、GIE ビットをセットしてふたたび、インタラプトを許可します。

レッスン11：　インダイレクト・データ・アドレシング

FSR(File Select Register)は、ある決められたファイル・レジスタによってアドレス指定できるようにします。INDF(Indirect File Register)に対するリードあるいはライトは FSR でアドレスされたファイル・レジスタを参照します。
これは、移動平均フィルタの組み込みに利用できます。移動平均値は最新の一連のnの値と平均値をいっしょに保持します。フィルタは2つのパート：巡回式キューと平均を計算する関数　を必要とします。

移動平均の説明図

ミッドレンジ PIC マイクロコントローラで平均値を計算するときに、次の値はどこに挿入するのかを、追跡するのに、FSR を用いて実現するのが最適です。これは、最も古い値が常に最新の値に上書きするのを確実にし、メモリ内で値を移動させる時間を無駄にしません。

ファイル・セレクト・レジスタを用いたプログラム例

補足：データ・メモリ・ブロック

レッスン11では、cblock ディレクティブにより、データ・メモリ・ブロックにワークエリアを確保しています。

```
        cblock    0x20
Queue:8                 ; 最新の8個のデータを保持する場所（8バイト）
RunningSum:2            ; 平均前の8個のデータの合計値（2バイト）
Round:2                 ; 割り算の途中結果と、平均値の最終結果が入る場所（2バイト）
        endc
```

ここで記述したラベルで、データ・メモリ・ブロック中の番地が定義されます。(;)コロンの後ろの数字が、バイト数を意味しています。

補足：間接アドレス指定の説明

temp に、AD コンバートした最新の値を一時保存します。（一時保存場所だから temp と命名したのでしょう）
QueuePointer は、移動平均の対象となる、最近の8つの値を保存する領域の先頭アドレスです。この先頭アドレスを FSR に格納します。
MOVWF INDF を実行すると、temp に一時保存した AD コンバートの最新値が、W レジスタを経由して、FSR が示す場所に保存されます。完了後に QueuePinter の値（アドレス）を 1 インクリメントします。レッスン11のプログラムでは、このポインタが8に到達したところで、また、最初の先頭の値に戻し、循環するようにしています。このポインタが示す場所には、常に8つの中で最新の値が格納されています。

図解すると、

間接アドレス指定による、テーブルへの値の格納の図

レッスン11はレッスン4のアナログ・ツー・デジタルのコードに移動平均フィルタを付加したものです。ポテンショメータを回転すると、アナログ・ツー・デジタルコンバータの読みが変化します。その平均値はLED表示に送られます。平均フィルタは0．2秒ごとに動いて表示の変化をスローダウンして目に見えるようにします。表示は古いポテンショメータの位置のカウントから、新しい位置のカウントに変化します。
フィルタは最新の8つの読みを平均します。多くのサンプルに対する2のべき乗を選択することにより、実際の割り算を用いる代わりに、単純なローテイトによる割り算で済ませることができます。
さらに、毎回数値列を合計するよりもキューの中の最も古い値を引いて、最新の値を加えたほうが、より高速に走りながら合計し続けられます。

移動平均の図

補足： 右ローテイト(1/2)を3回で、1／8
レッスン11では、8個のA2D値の合計を8で割り算して平均するのに、右ローテイトを3回繰り返しています。図を参考に正しく演算できていることを確認してください。

右ローテイト(1/2)を3回で、1／8の説明図

レッスン12： ルックアップ・テーブル(ROMアレイ)

レッスン8ではファンクション・コール(関数呼出)を紹介しました。レッスン12では、ルックアップテーブルが、関数呼出とプログラム・カウンタの意図的修飾に、どのような方法で、適用できるかを示します。(ルックアップ・テーブルのプログラムリストを参照してください)

ある値から別の値に変換するためにテーブルを適用すると、便利なことがあります。
高級言語で表現すると、このようになります。
y＝*function(X)*;
この関数では、すべてのXの値に対して、対応するYの値が返されます。

ルックアップ・テーブルは、入力データを意味のあるデータに変換する方法として高速です。なぜなら、変換関数は、進行中に計算するのではなく、あらかじめ計算しておいて参照するからです。

PICマイクロコントローラでは、このよう方法をプログラム・カウンタ(PC)の直接修飾に適用します。一例としてヘキサデシマル(16進値)をASCIIの相当の値に変換することがあります。それぞれの桁の独立したニブルは数値として扱われ、ルックアップ・テーブルのインデックスとして用いられます。そのインデックスはPCに送られ、対応するRETLW命令が、WREGに、該当する定数をロードして、呼び出し側のプログラムに返します。

ルックアップ・テーブルのリスト

もしも、そのテーブルが256バイトのプログラム・メモリの境界をまたぐようならば、あるいは、何らかの理由で、ルックアップ・テーブルが境界を越えるインデックス値で呼び出されるならば、それは、テーブル外の場所にジャンプしてしまいます。

よいプログラミングのために、もういくつかの命令を練習しましょう。はじめは、テーブルがただの16の入り口しか持っていないならば、16以上の値を通さないように確認しなければなりません。

これを実行するもっとも簡単な方法はPCLを修飾する前にWREGの値と論理ANDをとることです。:ANDLW 0X0F　さらに複雑なエラー回避策はアプリケーションに応じて、適切に行ってください。
さらに付け加えて、テーブルが256ワード境界を越えるかどうか認識しようとするとき、微妙な注意点があります。プログラム・カウンタは13ビット幅ですが、下位の8ビットだけがPCLに存在します。（図3－12参照）
残りの5ビットは、PCLATHに格納されます。しかし、下位8ビットのオーバフローは自動的にはPCLATHに桁あふれされません。そのかわりに、そういうケースでは、コードの中で、扱いの確認を確実にしてください。

命令の到達点としてのPCローディング

このレッスンでは、バイナリをGrayコードに変換する方法を組み込むために、ルックアップ・テーブルを使用します。Grayコードは、あるシーケンスから次のシーケンスに移るとき、1ビットだけが変化するような、バイナリコードです。それは、エンコーダのアプリケーションにおいて、状態間の無秩序なジャンプを避けるために、よく用いられます。バイナリ・エンコーダは光センサによりセンスするスロットがある透明のディスクのために組み込まれるものとして典型的なものです。異なるビット列のスレシホールド・レベルが異なるために、ビット列がわずかに時間的影響をうけやすい瞬間に誤った結果に変化してしまうかもしれません。Grayコードはこれを防止できます。なぜなら、ひとつのシーケンスから次のシーケンスまでの間に、1ビットだけが変化するからです。現在のコードは次のコードに変化するまで、正しいのです。
バイナリからGrayコードへの変換アルゴリズムはかなり複雑です。少ないビット数においては、テーブル・ルックアップが小さくて、高速です。
このレッスンはアナログ・ツー・デジタル値の上位ニブルとこれをGrayコードに変換して最初の4つのLEDに表示するものです。コードは、ポテンショメータを回転させる範囲内で、1回に1ビット変化します。

Grayコードコンバータ

バイナリ Grayコード変換 プログラムリスト