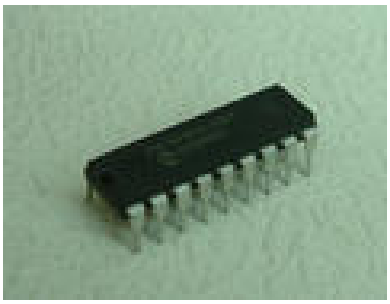
PICを使ってPICNIC！

著者　MORIO・KEI

MORIO 出版

【 Mission 1：PIC を知る① 】

PIC の説明を簡単にさせてもらいましょう。PIC とは、早い話「 超小型コンピュータ 」(マイコン という類のものです。) です。外見は ＜図①＞ のような形をしています。よく聞く「 IC 」と、形は一緒です。その中には、CPU、メモリ、プログラムメモリ（PIC を動かすソフトをしまっておくメモリです。)、周辺装置などが入っています (＜図②＞ 参照)。5 ㎝長の超小型パソコンと言えばもっとわかり易いでしょう。



＜ 図① ＞

＜ 図② ＞

PIC は先程の説明の通り、コンピュータの一種です。

PC でテキストエディタを起動して、編集し、結果を保存する。この一連の作業を、デジタルの CPU が一貫して行っていることは、皆さんご存知の通りですね。この PC のソフトは、当然 PC が無いと動かせません。ソフトの入った CD は、PC がなければ、ただの円盤です。それもご承知のことでしょう。

PIC も、同じように、チップの中にソフトが入っているし、電源につなぐとそのソフトを起動し、実行していきます。見かけは、PC となんら変わりはありません。細かく言うと専門的になりますが、OS が必要ない（存在しない…DOS 専用ソフトを動かす感じ)、IC のような用途として使われることが多いなどの違いがあるなどの理由から、PC とは一見かけ離れているように思えます。

しかし、プログラミング経験のある人は、プログラムを作り、デバックし、実行するという一連の作業に慣れていると思います。PIC にもとっつきやすいと思います。さぁ、がんばりましょう！

【 Mission 2：PIC を知る② 】

PIC は、皆さんと縁の無い物のように思われるでしょうが、実は、皆さんの周りにある冷蔵庫、テレビ、ラジオ、洗濯機など、電化製品と呼ばれるものの多くに使われているのです。例えば、テレビだったら赤外線の受信部と、チャンネル合わせの回路、洗濯機なら、すすぎ、脱水の制御、冷蔵庫なら温度調節回路などです。すごいでしょ。

しかし、20 年くらい前は PIC がない時代でした。今流行の（と言っても当たり前のものになりましたが）圧力電気釜で、全自動でおいしいご飯を、それも「 誰にでも 」作れるようになりました。これは、PIC が内部で温度調節をしている恩恵です。昔はこれがないので、温度設定やら、炊飯時間設定やらをやっても、今ほど充実したご飯は炊けなかったでしょう。この例からも PIC が実は皆さんの生活の一部となっていることがわかりますね。では、PIC がなぜここまで使われるようになったのか。これは、企業のコスト削減によるところが大きいです。

まず、テレビの例を挙げましょう。テレビは ＜図④＞ のような構造になっています。その中で、企業が赤外線受信（リモコンの）の部分のチップを新たに開発するとしたら、開発費だけで数百万円はかかってしまいます。また、チャンネル設定の回路もそうです。しかし、PIC なら、僕が使えるくらい安価で（最安値品は秋葉原最安価格で 150 円）開発キットも自作でき、開発環境は、PIC を作っているメーカーが無償提供しています。よって、製品の値段が安くなるわけです。例えば、10000￥のテレビが PIC 2 個の利用により、3000￥になったりするのです。これは決して単なる例ではありません。PIC による値段の低下は知らないうちになされているのです。

僕が 2000 円もっていれば、秋葉原の交通費と部品代が確保できるわけです。それくらい安価です。そして、チップを扱う点で、企業と同じ設備能力を備えたようなものです。個人の設備でチップの開発ができるように振舞います。それは、何でも作れることを意味します。僕が洗濯機を自作することもできるし、HDD を作ったりもできるわけです。それは又、企業にとって、もっと有益になるでしょう。さらに、PIC はいろんな周辺機器を備え、製品シリーズが豊富です。それにより選択範囲が広くなり、従って値段と機能の両立が、可能になるわけです。

＜ 図④ ＞　　＜ 図⑤ ＞

さて、PIC は CPU 内臓小型コンピュータだとい言いましたが、そもそも、CPU とは何なのでしょうか。

ここで、PIC を例にあげましょう。PIC の中の CPU 部分をさらに詳しく見ると、＜図⑥＞ のような構造をしています。

＜ 図⑥ ＞

わかりましたか？CPU とは、ある機能（ブロックとしましょうか）の集まりなのです。例えば、5+3 などをする機能（全加算器)、8－85 をする機能（全減算器)、2 進数 10010010 を右に 3 つ動かす機能（シフト機能）などの集まりです。そして、CPU の機能のうち、何を使うかを決めていくのがプログラミングに他ならないのです。機能の数はそれぞれの CPU に特有です。それ以上も以内もありません。CPU は神様です。プログラマーは CPU を信じることからすべてが始まるのです。そして、それを具現化し、CPU と人間を 1 対 1 対応させたのが「 アセンブラ言語 」です。

【 Mission 3：ハードに触れる① 】

さて、少し難しい話になりますが、皆さんに「 54+32 」を計算してもらいましょう。えっ？ばかにするな、まぁまぁ、そう言わずに。これがすべての始まりですから。

54+32＝86

何も難しいことはないですね。では、変化をつけて。これを 2 進数でやってみてください。2 進数とは、0 と 1、つまり、0 1 2 3 4 5 6 7 8 9 10 11 12 13…を 0 1 00 01 10 11 100 …と 2 つの数（0 と 1）と桁だけで表したものです。例をあげると、55 は 110111、68 は 1000100 になります。10 進数（普段のお金の数え方です）を 2 進数にするには、2 で割ったあまりを列挙し、2 で割ったのを切捨てていきます。

55 なら、55÷2＝27…1　→　26÷2＝13…0　→　…と続けていくと、元の値が最終的に 0 になります。そこまで割っていくと、答えが出ます。

これを踏まえて、54＋32 をやっていきましょう。

54（110110）+　32（100000）＝　86（1010110）

こうなります。では、CPU はこの計算をどうやって実現しているのでしょうか。

実は、デジタル CPU は、同時に 2 つの動作をするために、最低 2 つの装置を必要にします。54+32 は、2 進数の 1 桁目の計算、2 桁目の計算、3 桁目の計算…を同時にするために、1 桁の和を求める装置（半加算器）を桁数分並べて計算しています。図⑦のようにです。

＜図⑦＞

ここで、1 桁の計算をする「 半加算器 」を説明しましょう。

例として、２進数の「1」と「1」の和をとってみましょう。1+1＝10 ですね。

さて、２数の和の計算で重要なのは 2 点。まずは、1 桁の和。

1+1 は 10（桁上がりで 1 桁目は 0）、1+0 は 1、0+1 は 1、0+0 は 0 です。

これは、2 つの数が両方とも同じときだけ、0 になる、それ以外は 1 になる性質をも持っています。この演算を「 EX－OR（XO r ）演算 」と言います。「 N　XOR　M 」のように書きます。これを使うと、1 桁の和をとった後の 1 桁目の状態を表すことができます。これは、＜図⑧＞のような回路で実現できます。

次に、皆さんが当たり前のようにやる「 桁送り 」、つまり 1+1 の 1 桁目が繰り上がったのを 2 桁目に足しますね。これを実現するには、1＋1 のときの特別な状態、そう、2 つの数ともに 1 である性質を利用します。これは、2 つの数が両方とも 1 のときだけ、1 になる、それ以外は 0 になる「 AND 演算 」と言うものを利用します。「 N　AND　M 」のように書きます。これを使うと、1 桁の和をとった後の桁送りの有無を表すことができます。これは、＜図⑨＞のような回路で実現できます。

＜ 図⑧ ＞　　＜ 図⑨ ＞

この 2 つの回路をつかうと、1 桁同士の和をとった後の 1 桁目の状態と、桁送りの有無を表示する装置ができます。これを並べると、下の桁の桁送りを足して和を取る、つまり、あたかも複数の桁の演算を（一回で）実行する回路の様に振舞うわけで、CPU の大事な機能のひとつになります。（和をとる機能）

【 Mission 4：ハードに触れる② 】

Mission4 では、電気物理について紹介しましょう。皆さん、よく乾電池を使うでしょう（使わない、まぁそうおっしゃらずに…）。ご存知の通り、乾電池には＋と－があります。＋と－を間違えるとショートしたり、なんてことをよく耳にすると思います。

では、＋と－はそもそも何を意味するのか、それを紹介します。これは、電気回路のすべての基本ですから、しっかり理解しましょう。

＜図⑩＞を見てください。「 電子 」という粒が乾電池から放出されます。

＜図⑩＞

この電子、実はいっぱい電子があるところより、少ないところを好んで進んでいきます。乾電池の＋は、電子が少なく、－はその逆。－から＋へ電子が進んでいきます。それを結ぶ線があるとしますと、その途中に部品をつけると、その部品にも電子が流れます。この性質を利用して、あらゆる電気回路は作られています。「 すべての基本 」といったのは、これによります。

簡単に言うと、この 2 箇所の電子のある数の多さの差を数字で表したのが「 電位差 」で、「 電圧 」とも言います。電圧のほうはよく聞くでしょう。

【Mission 5：PIC を覗く① 】

いよいよ PIC に触れてみます。PIC の「 キモ 」は、PIC への接続の仕方（回路）と、プログラムです。PIC は、CPU 内臓コンピュータです。それも、IC 的な使い方もできる万能コンピュータです。PIC で PC を作った方もいるくらいですから、まさに「 小は大をかねる 」ですね。PIC は、プログラム次第で携帯電話のチップにも、ライトの点滅機にもなります。そこで、うまいプログラミングと回路の作成方法を学びましょう。※　尚、ここまでが、簡単な説明の限界で、ここからは、ある程度のハードウェア知識と思考が必要になります。あしからず…。ページ数の問題もありますしね。

PIC の簡易アーキテクチャを＜図⑪＞に示します。「 アーキテクチャ 」とは、ハードウェア内部の構造を図にしたものです。図を見るとメモリと CPU とプログラムメモリと周辺機能、そして、データの入出力を実現するための PORT と呼ばれる（IC のピン）部分でできています。

＜図⑪＞

まず、電源につなぐと、プログラムカウンタと言うものの位置を 0 にします。（CPU は、使用する CPU の機能＝命令＝の列挙…つまりプログラムです。…を上から順番に実行していきます。プログラムカウンタは、今実行している命令の場所を保持しています。これが無いと CPU は何もできません。）今回使う PIC のシリーズのひとつ「 PIC16F84A 」は、プログラムメモリが　1024*1WORD（14Bit）あるので（つまり、命令を 1024 個保存できます。）、0 にした位置から終了命令(END です。…というより、PIC の機能を終了する直前のプログラムカウンタの位置が、そこで最後になるような位置に挿入することで、終了できます…)までを実行していきます。

PIC の重要な機能（？…コンピュータにはあって当たり前かもしれませんが）にメモリ（レジスタ）があります。これは、あるデータ（普通は 2 進数の集まり＝Bit です。）を保持する機能です。例えば、56 という値を保持したりです。

このレジスタは、PIC を動作させるためのデータ（入力出力の有無など）を保持する「 SFR 」と「 汎用レジスタ 」の 2 種あります。SFR は、スペシャルファンクションレジスタ、つまり、特別な用途に使うレジスタで、前述の通りです。汎用レジスタは、文字通り汎用なレジスタで、自由に使えます。PIC16F84A では、汎用レジスタは 68 Byte あります。この 2 つを合わせ、ファイルレジスタと呼びます。1 つ 8Bit の容量です。

他に、「 ワーキングレジスタ 」というものもあります。これは、ファイルレジスタとは独立に存在し、8 Bit のデータを保持します。例えば 2 数の和を取る場合、足されるほうに足すほうの値を足す際に、一時的に演算結果を保持する必要がありますので、ワーキングレジスタが必要になってくるわけです。（別に独立させないで、ファイルレジスタの一部にしてもいいのですが、そうなると、常に使えないレジスタが出来て、PIC の操作も複雑化しますし、ワーキングレジスタの回路も演算装置の一時記憶装置としての位置付けなので、独立しているのです。）

PC（Pentium4 など搭載の PC）の場合、ファイルレジスタのデータを入出力するピン(内部配線)数が 32Bit、つまり、32 本あり、一度に転送できるデータは 32Bit です。また、命令のサイズも 32Bit です。この 2 つを共有のピンで転送しています。このような、メモリとプログラムメモリのピンが共有な構造を「 ノイマンアーキテクチャ 」といいます。＜図⑫＞がそうです。ENIAC を作製したノイマンさんが考えた構造です。これは、メモリを外付けにするのを前提にするような場合、データの Bit 数を優先する構造で、命令 Bit 数も慣例的に、8 の倍数になってしまいます。

さて、PIC の場合、メモリは 8Bit、プログラムメモリは 12～16Bit（シリーズにより違いますが、同じシリーズ内は互換です。）になっていて、それぞれ独立しています。これを「 ハーバードアーキテクチャ 」といいます。＜図⑬＞がそうです。利点は、プログラムメモリの値（命令）を読むのと並行して、レジスタの値を転送できるので、CPU を無駄なく（待ち時間なしで、例えば、演算命令と同時にレジスタの値も転送でき、その時間の間に次の命令を、先読みしておく…命令フェッチといいます…なんてこともできます。）使用できます。内臓コンピュータである PIC は、プログラムメモリも内蔵なので（外付けしにくい構造です。）、2 つを分ける必要が出来てきます。

＜ 図⑫ ＞　　＜ 図⑬ ＞

PIC（というより CPU 全般で）は、「 クロック 」というものを時間の基準にして動作します。これは、＜図⑭＞のようにある基準値以上の電流の行き来 1 回で 1 クロックとし、この電流の検出でプログラムメモリから命令を読み取り、実行します。

前述の通り、PIC は、ハーバードアーキテクチャです。

まず PIC は、①　最初のクロックで現在のプログラムカウンタの値の位置（アドレスとも言います。）にあるプログラムメモリの値を読み込み、②　２個目のクロックでデコーダー（プログラムメモリのデータを解読し、命令を転送する装置）にデータを転送し、③　３クロック目でデコーダーから CPU へ何を実行するかを転送し、④　４クロック目で CPU が命令を実行します。この並行動作は、ハーバードアーキテクチャだから出来るのです。

＜ 図⑭ ＞

これくらいで PIC の動作の概要は解説しました。もっと詳しく読みたい人は、インターネット上に転がっているフリーの PDF 資料があるので、それを御参照ください。

【Mission 6：PIC を覗く② 】

PIC16F84A の外見は、＜図⑮＞のようになっています。基本的に、＜図⑯＞のように接続します。ここで、PB0～PB7、PA0～PA4 の入出力設定により、ここの回路が変わります。尚、＜図⑯＞中の「 GND 」とは、電位が 0V の場所で、ピンから GND に電流が流れ込み、その途中に部品を挿入することで、いろいろな回路が実現できます。

＜ 図⑮ ＞　　＜ 図⑯ ＞

あとは、プログラム次第で、あなたの長年の夢がかないます！！

…少し無責任な解説になってしまいますので、ソフトを作るための手段「 アセンブリ言語 」をまず習得しましょう。

アセンブリ言語とは、先ほど紹介したとおり、CPU と 1 対 1 対応している言語で、CPU 内部の機能（命令、強いてはデータ）をそれぞれ英単語のニックネーム（ニモニックといいます。）で区別し、人間にわかりやすくした言語です。当然、CPU の持っている命令以外の命令語は扱えません。（あとで出てくる「 擬似命令 」は例外ですが。）

例を出しましょう。例えば、A+B をする命令を「 ADD 」とし、A－B をする命令を「 SUB 」とします。ADD が 00100010、SUB が 00100011 というデータだと仮定すると、ADD と SUB はデータ（命令）を意味しますが、人間には「 ADD は加算、SUB は減算 」と一目でわかるので、プログラム開発が容易になります。そして、それぞれの命令は何秒で実行できるか決まっているものです。なので、時間にシビアなソフトの開発をするにはもってこいです。なんせ、ニックネームを、見たまま順番に数字に戻せばいいのですから。

【Mission 7：アセンブリ言語を学ぶ①】

まず、最近作った連射速度測定器を紹介します。PIC に慣れれば、これ位の物ならすぐ作れるようになりますので、がんばってください。又、＜図⑰＞はその外見です。僕の記事の最後に、そのプログラムを載せておきました。

＜ 図⑰ ＞

PIC16F84A の命令数は 35 個です。この数が多いほど、多機能になりますが、価格もあがる傾向になります。そして複雑になってしまします。最新機種は、命令数 75 個もあります。その点、初めての人にとって、35 個のニックネームを覚えるだけでいい PIC16F84A は、うれしい限りです。僕もすぐとっつけました。

アセンブリは、基本的に「 A　b, c 」の形で命令を記述します。A が ADDWF（加算命令）なら、ｂにｃを足すという命令になります。たったこれだけで加算が出来ます。この基本さえ覚えれば、後は A の種類を覚えるだけです。

A に入る命令には、それぞれ「 実行時間（命令を実行するのにかかる時間)」というものが決まっており、20MHｚで PIC16F84A を動かした場合、A は 200ns または 400ns です。2 つの種類があるのは、簡単に言うと 1 サイクル（1 つの命令を実行する単位です。PIC の場合、4 クロックで 1 サイクルです。）の命令と 2 サイクルの命令があるからです。尚、この HZ とは、1 秒間に PIC に入力されるクロック信号の数(クロック周波数といいます。)です。先ほど述べたように、PIC はクロックを 4 つ使用してひとつの命令を実行します。20MHｚなら、1 秒間に 20000000Hz ですので、1 秒間に 5000000 個の 1 サイクル命令を実行できるわけです。又、PIC16F84A は、3 種類のクロック周波数（4・10・20）で動かせます。これは、オプションです。

【Mission 8：アセンブリ言語を学ぶ②】

アセンブラの説明をしましょう。アセンブラとは、アセンブリ言語（の文法）で作ったプログラムソースを実際のプログラムに変換するソフトです。文字を打って作ったプログラムソースは、文字の羅列ですが、文字と命令はデータの構造が違うので、プログラムにはなりません。例えば ADD という命令を 10011101 とすると、プログラムソース中の＝ADD＝という 3 文字のデータ自体は 10011101 ではないので、これを命令として翻訳する必要があります。この作業自体はアセンブルといいます。

ちなみに、あとで出てくる「 擬似命令 」とは、アセンブラが特別に用意した命令で、プログラム中に書いてもエラーにはならず、ある決まった動作（アセンブル時にアセンブラ側がソースに手を加える処理）をします。

又、CPU ごとにアセンブラで使える命令の種類や数は違います。さらに、アセンブラによっては、擬似命令の種類も違ってくることがあります。

※ 尚、今回の解説のアセンブラには、PIC を作っているメーカーが無償公開している「 MPASM 」を使用しています。簡易の説明ですので、詳しい MPASM の説明や、アセンブリ言語の基本は、専門書を参考してください。MPASM は、原則、擬似命令を行の左端に書き、それ以外の命令は、それよりも 1 マス以上空白をあけて記述します。

さてさて、PIC の重要な命令をいくつか紹介しておきます。

※ ちなみに、この先の「 F 」は、ファイルレジスタ名で、後述の EQU で定義して使います。

～ EQU A … これは、擬似命令の 1 つです。

ファイルレジスタ F があるとすると、F のある位置（0 から数えて N byte めのレジスタ F をさします。＝N 番目＝のことをアドレス N といいます。）を A の場所に数字で書き、～の場所にニックネームをつけることで、プログラムソース中でのファイルレジスタ F の存在が、「 アドレス A 」という数字ではなく、意味のある「 ニックネーム ～」により明示的になり、プログラムの開発効率があがります。

EQU は、それを実現するもので、EQU を記述すると、～ が A と同じ物になります。

MACRO　　…　擬似命令です。これは、マクロというものを定義します。

「 MACRO 〜 引数 1, 引数2, 引数 3, …引数 N 」の形で記述します。

マクロとは、アセンブラの場合、ある命令を組み合わせ、まとまった命令群を作っておき、それをあたかも１命令で実行するかのように見せかける手法です。マクロの 〜 部分に命令の名前を付けておき、引数（マクロ中に使うレジスタやアドレスの値など）を記述しておき、別の場所でマクロ（マクロ命令といいます。）を呼び出すことで、メインプログラムとマクロ（つまり、見かけは特別な機能）が見やすくなり、開発効率があがります。

ENDM　　…　擬似命令です。これは、マクロの定義をこの命令の手前で終了する命令です。

〜（ラベル）…　これは命令ではなく、ソース中に見出しをつけるようなもので、 〜 にラベルというニックネームをつけると、その後ろの命令のプログラムメモリのアドレスが擬似的にラベルに保持されます。擬似命令と同様に行の左端に書きます。

MOVLW　L　…　これは、定数（ソース中に固定値として記述する数） Ｌをワーキングレジスタに転送する命令です。1 サイクルで動作します。

MOVWF　F　…　これは、ワーキングレジスタ F のデータを指定したファイルレジスタ F に転送する命令です。1 サイクルで動作します。

DECF　F　…　これは、ファイルレジスタ F のデータを－1 する命令です。1 サイクルで動作します。

lNCF　F　…　これは、ファイルレジスタ F のデータを＋1 する命令です。1 サイクルで動作します。

ANDWF　F　…　これは、ファイルレジスタ F のデータとワーキングレジスタのデータの AND を取る命令です。1 サイクルで動作します。

XORWF　F　…　これは、ファイルレジスタ F のデータとワーキングレジスタのデータの XOR を取る命令です。1 サイクルで動作します。

lORWF　F　…　これは、ファイルレジスタ F のデータとワーキングレジスタのデータの OR を取る命令です。1 サイクルで動作します。

GOTO ラベル…　これは、プログラムカウンタをラベルの位置にする命令です。2 サイクルで動作します。

CALL ラベル…　これは、プログラムカウンタをラベルの位置にし、現在のプログラムカウンタの値を 「 スタック 」 と呼ばれる特別なメモリに保存する命令です。後述の RETURN を呼ぶと、現在の位置に戻って来ます。GOTO はそれが不可能です。2 サイクルで動作します。

RETURN　　…　これは、プログラムカウンタをスタックの値にする命令です。2 サイクルで動作します。

BSF　F, A　…　これは、ファイルレジスタ F の A ビット目を 1 にする命令です。1 サイクルで動作します。

BCF　F, A　…　これは、ファイルレジスタ F の A ビット目を 0 にする命令です。1 サイクルで動作します。

NOP　　…　これは、何もしない命令です。時間（クロック）だけが過ぎていくので、タイミングを取ったりするときに使用します。

※　この他にも、命令や擬似命令はありますが、これくらいあれば、はじめのうちは大丈夫でしょう。

【Mission 9：実習　LED を光らす】

いよいよ実習です。ここまでの内容は、知識をつける内容でしたが、Mission9 をみているあなたは、格段に PIC を知った人になっているはずです…よね？ですので、一気に実習を進めていきますので、覚悟してくださいね！！（笑）

さて、実習では「 LED 」を光らしてみます。LED とは、おそらく皆さんご存知だろうと思いますが、早い話、スーパー豆電球だと思ってください。何がスーパーなのかと言うと、①明るさがすごく、②長寿命で、③効率がいい、点です。これで、小さいものでは直径１㎜ぐらいのサイズながら、豆電球より数倍明るく、電池の減りも穏やかな回路が作れるわけです。

電子回路に詳しい方から、LED を光らすぐらい乾電池で光らせればいいのに…。などと厳しい言葉が聞こえてくるかもしれませんが、まぁ、そうおっしゃらずに。なんせ、LED は光っているのが目に見えることが特徴なのですから、まず PIC の内部を目に見えるようにする、実習として最高の部品なのです。

LED を光らすには、LED に 0.7V 以上（3.5V ぐらいまで流せます。）の電圧をかけ、15～20mA ぐらいの電流を流せば光ります。PIC も電源につなぐと、本体のピンから電源電圧で 25mA の電流が流れます。LED を光らすにはちょうど良いわけです。ただ、PIC は 2.5V くらいから 6V くらいまでの電源電圧で駆動しますから、電源電圧が 5V の時に LED などをつなぐと LED が壊れてしまいますのでご注意を。

PIC は＜図⑱＞のような構造になっています。「 PORT 」という部分で電流の出入りをします。PORT への入力を「 INPUT 」 PORT からの出力を「 OUTPUT 」といいます。

さて、例えば PIC の内部にある変数の状態を目で確認したいなら、PORT からピンを 2 進数の各桁に対応させて、電流を出力（制御）すればいいのです。PIC には「 PORTB 」という SFR（8Bit）があります。これは、PORTB のピンごとの出力・入力の状態を示します。従って、その値を制御すると、ピンから出力される電流もその通りに決定されます。

＜ 図⑱ ＞

これを踏まえて、A , B , C の 3 つの変数を用意し、A＋B＋C の結果を PORTB から出力するプログラムを作ってみましょう。

まず、先程も述べたように PORTB のピンの状態は PORTB レジスタに記憶されています。このピン 1 本 1 本を、入力にするか出力にするかは、「 TRISB 」という SFR で制御します。TRISB は、各ビットが、ピンの入出力設定に対応していて、0 なら出力、1 なら入力になります。

ピンを 8 本（8Bit）全て出力にするには、TRISB を 11111111 にすればよいのです。まずは、この操作をします。これは、以下のように記述できます。

```
#INCLUDE   ＜p16f84a.inc＞
  BSF            STATUS, RP0
  MOVLW          B‘ 11111111 ’
  MOVWF          TRISB
  BCF            STATUS, RP0
```

これで、PORTB の値が、ピンからの出力に対応する回路として振舞います。

※　尚、「 #INCLUDE 」とは、今編集中のファイルの外部から、マクロや定義などを参照する場合に使います。SFR もファイルレジスタですので、EQU でアドレスを明示的にしておくのが上手い方法です。それを、外部ファイルに分けることで、「 基本設定 」としてのファイルと「 動作設定 」としてのプロジェクトソースとを分離でき、プログラムソースがわかりやすくなりますので、今回も #INCLUDE することにします。

次に、A＋B＋C を実現するプログラムを書きます。

※　A，B，C は例えば、8Bit のファイルレジスタで構成します。

```
#INCLUDE　＜p16f84a.inc＞

  A             EQU     0Ch
  B             EQU     0Dh
  C             EQU     0Eh
  RET           EQU     0Fh

  CLRF          RET

  MOVLW         ～
  MOVWF         A
  MOVFW         A
  ADDWF         RET

  MOVLW         ～
  MOVWF         B
  MOVFW         B
  ADDWF         RET

  MOVLW         ～
  MOVWF         C
  MOVFW         C
  ADDWF         RET

  MOVFW         RET
  MOVWF         PORTB
```

これと、前頁のプログラムをあわせると、A＋B＋C を出力し、LED を光らせることが出来ます。このプログラムを、あとは PIC 本体に転送し、必要な回路を作るだけです。この転送を実現する装置を俗に「 PIC ライタ 」と呼びますが、これは今回著作権の関係で解説できませんので、あしからず…。専門書をご参照ください。

実習の回路図を＜図⑲＞に示します。

＜ 図⑲ ＞

【Mission おまけ：PIC16F84A の詳細データ】

このページでは、僕の記事を読んでくれた皆さんにプレゼントをあげちゃいます。

僕の記事を見て PIC を触りたくなった人、ますますプログラミングに興味をもった人、いろいろな方がいると思いますが、将来 PIC に触れる方のために、今回実習で使用した PIC16F84A の詳細なデータを載せておきます。