ホビープログラマの、2進数による乗算

（アセンブラ・プログラミングへの考え方）

１、はじめに

　コンピューターの内部では、計算はすべて２進数で行われているのは、皆さんのご承知のとおりです。しかし我々は日常の生活において２進数を使うことはまずないので、これについての必要性をあまり感じない処だと思います。
パソコン（以下ＰＣと略します）のプログラミングを行う場合でも、コンパイラー言語や、豊富な関数類のおかげで、２進数をさほど意識しないで済みます。
近年ではＰＩＣをはじめ、安価なマイコンが数多くリリースされ、ホビーユースでもこれらを利用した工作が盛んに行われて来るようになりました。
この場合でも優れた開発ツールのおかげで、２進数をさほど意識しないプログラミングを行うことが出来ますが、ＰＣのプログラミングと比較し２進数が活躍する場面が多いと言えそうです。
Ｈ８等の比較的規模の大きなＣＰＵの場合、乗算回路や除算回路が内部機能として組み込まれていたり、メモリの大容量化の恩恵で高級言語が利用しやすくなり、高度な演算を行う時でも２進数を強く意識しなくてもプログラミング出来るようになりました。
しかし、ＰＩＣなどのように、これらの機能がないか、または利用しづらい環境下でプログラミングを行う場合は、やはり２進数を意識せざるをえません。
メモリの小さなチップを用いた場合や、何でも自分の手で済ませないと気が済まないハードユーザーには、２進数は、もはや避けては通れない道だと思います。
　今回は、ＰＩＣなどのＣＰＵにプログラミングを行うことを念頭において、この２進数を用いた乗算のアルゴリズムについて、そのさわりを説明したいと思います。

２、１０進数と２進数

かけ算の説明に入る前に、もう一度１０進数と２進数の違いについて簡単におさらいをしようと思います。我々に最もなじみの深い１０進数の起源は、我々人類の手の指の数が両手で１０本ある事に由来していると言われています。
１０進数では桁が一つあがる事に、その数が１０倍（１０の乗数が１上る）になります。
1 と 10 とでは、その比は 10 倍であり、100 と 1000 でもその比は 10 倍になります。

１０進数の一例として 1457 という数を１０の乗数により因数に分解してみると、

$$1457 = 1*10^3 + 4*10^2 + 5*10^1 + 7*10^0$$
$$= 1*1000 + 4*100 + 5*10 + 7*1$$
$$= 1000 + 400 + 50 + 7 = 1457$$

と表現する事が出来ます。

２進数は、我々の生活として馴染みが薄いのですが、基数が１０か２かの違いだけであとは全く同じと考える事が出来ます。
２進数では桁が一つあがる事に、その数が２倍（２の乗数が１上る）になります。
1 と 10 とでは、その比は 2 倍であり、100 と 1000 でもその比は 2 倍になります。

２進数の一例として 1011 という数を２の乗数により因数に分解してみると、

$$1011 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$
$$= 1*1000(8) + 0*100(4) + 1*10(2) + 1*1$$
$$= 1000 + 0 + 10 + 1 = 1011$$

（カッコ内の数値は１０進数による表記です）

と表現する事が出来ます。

少し理解しにくいかも知れませんが、２進数も１０進数もその本質に大きな違いはなく、計算手順もまったく同じといえます。

３、１０進数における乗算

２進数による乗算を説明する前に、我々にもっとも馴染みの深い１０進数による乗算の手順について考察してみます。
例として 57 * 4763 について考えてみます。

57 * 4763　を因数として分解し、別の表現をすると

$$57 * 4763 = 57 * (4*10^3 + 7*10^2 + 6*10^1 + 3*10^0)$$
$$= 57 * 4*10^3 + 57 * 7*10^2 + 57 * 6*10^1 + 57 * 3*10^0$$
$$= 57*10^3 *4 + 57*10^2 *7 + 57*10^1 *6 + 57*10^0 *3$$
$$= 57*1000 *4 + 57*100 *7 + 57*10 *6 + 57*1*3$$
$$= \mathbf{57000} *4 + \mathbf{5700} *7 + \mathbf{570} *6 + \mathbf{57} *3$$

となります。

ここで最も重要で着目すべき点は、**57000***a + **5700***b ・・・のように
掛けられるべき数（ここでは 57）が、左に３桁シフト*a + 左に２桁シフト*b・・
というように、規則的な変化をしているという点です。

この計算方法は、私たちが普段行っている手計算でも、意識こそしていませんが
同じ事をしています。

５７×４７６３　を手計算で行う。

```
          ５７
×      ４７６３
―――――――――――
        １７１    ←５７×３を計算している
      ３４２０    ←０は書かないが、５７０×６を計算している
    ３９９００    ←０は書かないが、５７００×７を計算している
  ２２８０００    ←０は書かないが、５７０００×４を計算している
―――――――――――
  ２７１４９１    ←計算結果の集計をしている
```

４、２進数における乗算

いよいよ本命の２進数による乗算を説明します。
ここでは例として　1101 * 0101　について考えます。
考え方は１０進数も２進数も全く同じと言えます。

この式を因数として分解し、展開すると、

$$1101 * 0101 = 1101 *(0*2^3 + 1*2^2 + 0*2^1 + 1*2^0)$$
$$= 1101 *0*2^3 + 1101 *1*2^2 + 1101 *0*2^1 + 1101 *1*2^0$$
$$= 1101*2^3 *0 + 1101*2^2 *1 + 1101*2^1 *0 + 1101*2^0 *1$$
$$= 1101*1000 *0 + 1101*100 *1 + 1101*10 *0 + 1101*1 *1$$
$$= \mathbf{1101000} *0 + \mathbf{110100} *1 + \mathbf{11010} *0 + \mathbf{1101} *1$$

となります。

ここでも重要で着目すべき点は、**1101000***a* + **110100***b* ・・・のように
掛けられるべき数（ここでは **1101**）が、左に３桁シフト*a + 左に２桁シフト*b・・
というように、２進数でも１０進数の計算と同様の規則的な変化をしている事が理解
出来ると思います。

アセンブリ言語でこれらの演算をする場合は上記の式を、ほぼ、そのまま当てはめる
ことが出来ます。

アセンブリ言語には、ビットシフト命令や加算命令、条件判断命令があります。
上記の式で、左に１桁シフトとありますが、これはビットシフトに他なりません。
また、**1101000** *0　のように、 ～*0 となっている項目は、条件判断命令で、加算をしなければ良いのです。

以上の乗算の内容をアセンブラ的にまとめると、

1101 を左に 3bit シフトして **1101000** とするが、乗ずる値の最上位ビットが
　0101 で **0** なので何もしない。
1101 を左に 2bit シフトして **110100** とし、乗ずる値の上から 2 ビット目が
　0**1**01 で **1** なので加算する。
1101 を左に 1bit シフトして **11010** とするが、乗ずる値の上 3 ビット目が
　01**0**1 で **0** なので何もしない。
1101 を左に 0bit シフトして **1101** とし、乗ずる値の最下位ビットが
　010**1** で **1** なので加算する。

とすれば、乗算が可能となります。

実際のプログラミングではテクニック上の問題として、メモリやステップの節約の為、加算してからビットシフトを行ったりと、多少の差異はありますが乗算としての理論は上記と同じです。

５、まとめ

以上のように、１０進数も２進数もまったく同じ手法が使えるという事がご理解頂けたかと思います。２進数にしてしまえば、それをアセンブラ・プログラミングに置き換えるのは比較的容易です。
ただ我々は２進数については実生活として馴染みが薄いのと、１０進数でも無意識のうちに計算している事が多いのですが、手順をよく分析して、それをプログラミング可能なように追ってゆけば、アセンブラ・プログラミングによる数値計算もさほど難しくないと実感して頂けるかと思います。

文章　JJ1LXH ／成田隆司