（1） CPU の高速化

前回の紹介では、「並列化」、「アルゴリズムの工夫」の 2 点について、ご説明しましたが、「アルゴリズムの工夫」に関しましては、乗算のみのご紹介でした。今回はその続編ということで、その他の工夫についてご紹介いたします。

（2） 除算

割り算を筆算で行うときは、掛け算の九九を用いて、大体のあたりをつけて商をたて、除数とその商との積を求めてから、被除数から引いてみて、引きすぎたら、商を減らして、またトライするという具合に、「あたりを付ける」というようなところが実に人間的で、これが実はコンピュータには大変難しい処理なのです。では、コンピュータではどうするのか？

何回引けるかをカウントし、引けた数を商とする。というのがコンピュータの処理です。何とも「芸のない」というか何というか・・・と言わないでください。このような処理の中でも、工夫をしながら、少しでも速く処理する努力はあるのです。

何度も申し訳ありませんが、コンピュータは 2 進数で処理をしますので、この処理を 2 進数で実行します。今まで 2 進数としつこく言ってはいたのですが、2 進数について説明していないことがありました。それは正の数と負の数をどのように表すかということです。8 ビットの 2 進数と言っても、これまでは正の数として扱ってきていました。今回、除算を説明するに当たっては、いよいよ負の数としても扱わないといけませんので、ここで脇道にそれますが、説明をさせていただきます。

あまりビット数が大きいと説明も大変なので、4 ビットの 2 進数を考えます。今まで 4 ビットの 2 進数というと、0～15 を表せると理解してきました。しかし、ここでは 4 ビットで負の数までも表現する方法を説明します。1 を 4 ビットの 2 進数にすると、0001 です。では、-1 はどうでしょうか？1 と-1 を加算すると 0 になるはずです。この考えを使いましょう。つまり、0001 と足して 0000 になる 4 ビットの 2 進数は、1111 です。このように、足して 0 になる数字を、1111 は 0001 の「2 の補数」、という言い方をします。このような考え方で 4 ビットの 2 進数で正の数、負の数を表すと下の表のようになります。ちなみに「1 の補数」というのは、その数値の各ビットを反転させた数値です。「2 の補数」は「1 の補数」+1 で求めることができます。

2進数	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
正負の数	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7

ここでは 4 ビットなのですが、これが 8 ビットになると下のようになります。

2進数	10000000	10000001	～	11111110	11111111	00000000	00000001	～	01111110	01111111
正負の数	-128	-127	～	-2	-1	0	1	～	126	127

つまり､4 ビットでは-8～7 の正負の数､8 ビットでは-128～127 の正負の数が表せます。2 進数を見てみると、それぞれのビット数で、一番上のビットが 0 の時は正の数（ゼロも含む）、1 の時は負の数であることがわかります。そのため、1 番上のビットのことを符号ビットとみなします。

さて、そのような準備をもとにして、さっそく、下のような、01010101÷00000101　十進数で表すと 85÷5 ですから､商 17 余り 0 となるはずです｡つまり 2 進数では､商 00010001 余り 00000000 です。図で処理の様子をステップバイステップで見ていきましょう。ここでの割り算は、小数点以下までを求める割り算ではなく、整数部分の商を求めて、余りも求める除算を行います。処理を始める前にまず、被除数の上に除数と同じ桁数のゼロを置きます｡除数を引く前に上位桁に 0 を付加した被除数を左に 1 ビットシフトしてから除数を引きます。ゼロから引くので負の数になるはずです。00000101 の 2 の補数になっているのがお分かりになりますか？反転して 11111010 に 1 を加えた数 11111011 になっていますね。

```
  00000000 10101010
- 00000101
------------------
  11111011 10101010
```

負になったということは引けなかったということですので、商としては 0 となります。我々が筆算で割り算をする場合、このように部分剰余が負のときは、引き過ぎたということで、一回足し戻して正の部分剰余に戻すところですが、ここでは、足し戻さず、負のまま左に 1 ビットシフトします。そのとき、商の 0 を右からシフトインします。その位置では、先ほど足し戻さなかった代わりに除数を加算します。先ほどの桁位置で引き過ぎたということは-1 の重みですが､1 ビット部分剰余を左にシフトした位置関係で足し算すると､-1+1/2=-1/2、つまり、この桁で-1 しているのと同じ計算になります。足し戻して、この桁で引き算するのと、結果的には同じことです。

その結果の部分剰余は相変わらず、負の数です。ですから商としては 0 です。また同様に、0 をシフトインして左に 1 ビットシフトします。まだ負の数ですから引き過ぎているということで、次の演算も加算です。

```
    11110111 01010100
+   00000101
-----------------
    11111100 01010100
```

以降、このアルゴリズムで処理を進めると以下のようになります。

```
    11111000 10101000
+   00000101
-----------------
    11111101 10101000

    11111011 01010000
+   00000101
-----------------
    00000000 01010000
```

ここで、部分剰余がゼロとなりました。つまり、この桁では引けたということになりますので、商は 1 です。1 をシフトインとして左に 1 ビットシフトします。次の演算は引き過ぎを足し戻せている状態なので引き算となります。

```
    00000000 10100001
-   00000101
-----------------
    11111011 10100001
```

先ほどと同様にまた引き過ぎになりましたので、次は加算です。同様に繰り返すと、

```
    11110111 01000010
+   00000101
-----------------
    11111100 01000010
```

```
    11111000 10000100
+   00000101
-------------------
    11111101 10000100

    11111011 00001000
+   00000101
-------------------
    00000000 00001000
```

ここまでで、8 ビットの被除数が除数によって 1 ビットずつシフトされて引き算され、引けたら 1、引けなかったら 0 の商を求めたことになりました。最後は部分剰余がゼロとなりましたので、商として 1 となり、1 をシフトインとして左に 1 ビットシフトした結果、

```
00000000 00010001
```

となりました。この上位 8 ビットが余り、下位 8 ビットが商となっています。除算は乗算のように部分積を求めてそれを加算するような、演算の並列化ができないため、どうしても被除数のビット数分の演算が必要になりますが、引き過ぎたら次の桁で足すという具合に少しでも演算回数を減らす工夫がされていることがおわかりいただけますか？

（3） 十進二進変換

さて、だんだんと細かい話になってきました。ですが、人間はやはりいつも 10 進数を使って生活をしていますし、かといって、コンピュータは二進数で数字を処理しますので、十進数と二進数の変換という処理がどうしても必要になってきます。これも高速に処理するに越したことはありません。

処理の方式を説明する前に、コンピュータの中で 10 進数がどのように表現されているかを説明します。と言っても実際の数値は 2 進数ですので、BCD というコードで表すのが一般的です。BCD とは、Binary Coded Decimal といい、二進化十進法と言います。具体的には 2 進数 4 ケタで 10 進数 1 ケタを表します。下に対応表を示します。

2進数	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
10進数	0	1	2	3	4	5	6	7	8	9

1010～1111 はどうなるのか？という疑問があるでしょうが、BCD としてはこの 1010～1111 は不正な十進数として、エラーとするような決まりになっています。10 進数を扱う命令がありますが、このような値があると 10 進数データエラーというようなエラーとなってしまいます。ですが、コンピュータとしては、データが 10 進数なのかということはわかりませんので、単純に ALU で加算などをすると、2 進数の加算として処理されてしまいますので、10 進数として加算するためには工夫が必要になります。ここでは 10 進数の加算の方法については踏み込みませんので、簡単に考え方だけ触れますと、各桁（4 ビットごと）の加算結果が 1010 以上になる場合は 6 を加算して補正する。とだけ言っておきます。実現方法はご自分で考えてみてください。頭の体操になりますよ。

10 進数の加算はさておきまして、十進二進変換に話を戻しますが、コンピュータで行う変換はこの BCD で表された 10 進数を 2 進数に変換するという処理となります。

処理方法は大きく 2 種類あります。

その一つは、10 進数の各桁は 10 のべき乗の重み、つまり 1 の位は 10 のゼロ乗、10 の位は 10 の 1 乗、100 の位は 10 の 2 乗、という具合ですが、その計算を 2 進数として実施すれば、2 進数になるはずです。4 ケタの 10 進数で、処理方法を例示してみます。rstu という 4 ケタの 10 進数とします。

r×10 の 3 乗＋s×10 の 2 乗＋t×10＋u

これの処理手順は、以下の処理を 2 進数として実施します。2 進数として実施というのは難しいことではなく、ALU や乗算器（前回に紹介した 4 ビット×4 ビットの積を ROM に持っておく）をそのまま使えば、2 進数として演算したことになります。

① r×10 を計算します。

② s を加算し、その結果に 10 をかけます。

③ t を加算し、その結果に 10 をかけます。

④ u を加算します。この結果が最終結果、つまり変換した 2 進数です。

どうでしょうか？結構高速にできそうですね。

この方法は、このような十進二進変換だけではなく、多項式の計算にも応用できます。わかりますか？ax3＋bx2＋cx+d（x の 3 次式）は、a×x を計算し、その結果に b を足して、x をかけ、その結果に c を足して x をかけ、d を足します。要は、掛け算と足し算を繰り返すだけです。このような演算を積和演算と言いますが、これを高速に処理するプロセッサとして、DSP（デジタルシグナルプロセッサ）というものがあります。デジタル信号処理ではこの積和演算の高速化が非常に重要です。

もうひとつのやり方は、我々が筆算で行う方法を使います。つまり、10 進数を 2 で割っていき、その余りを取り出していくという方法です。最初に 2 で割った余りは、一番下の桁(ビット)となり、その商を 2 で割った余りは、その上の桁となります。さらにその商を 2 で割って、その余りがその上の桁、という具合に、商が 1 になるまで繰り返して求めます。具体例で示しますと、例えば、1234 という 4 ケタの 10 進数を 2 進数に変換します。下の図をご覧ください。これにより、10011010010 という 2 進数となります。

```
2 ) 1234
2 ) 617   ---------- 0
2 ) 308   ---------- 1
2 ) 154   ---------- 0
2 ) 77    ---------- 0
2 ) 38    ---------- 1
2 ) 19    ---------- 0
2 ) 9     ---------- 1
2 ) 4     ---------- 1
2 ) 2     ---------- 0
    1     ---------- 0
```

これをコンピュータにやらせるにはどうしたらよいでしょうか？2 で割るというのは、その数を 1 ビット右にシフトすればよいことは前回も学びました。しかも、その時に一番右から出てくるビットが余りになります。いかにもコンピュータの回路で実現しやすそうな方法ではないでしょうか！さっそく絵に描いてみましょう。今回も 10 進数 1234 を使ってみます。BCD で表した 1234 を右に 1 ビットシフトすると絵のように、

0000 1001 0001 1010 0

となりますが、ここで少し考えてください。BCD の上から 2 桁目と 4 桁目はその上の桁から 1 が入ってきました。この 1 について考えてみます。BCD の桁で考えた上位桁ですので、10 の重みをもって入ってきているはずですので、その 2 分の 1 の 5 として入ってこなければなりません。しかし 2 進数として処理されているので、8 となってしまいます。つまり、上の桁から 1 が下りてきたら、3 引いてやる補正が必要です。1 ビットずつ右にシフトし、補正をするという処理を繰り返したものが下図です。

	0001 0010 0011 0100　1234		
	0000 1001 0001 1010 0　091A		
補正	0000 0011 0000 0011　を引く		
	0000 0110 0001 0111　0617		
	0000 0011 0000 1011 1　030B		
補正	0000 0000 0000 0011　を引く		
	0000 0011 0000 1000　0308		
	0000 0001 1000 0100 0　0184		
補正	0000 0000 0011 0000　を引く		0000 0000 0011 1000　0038
	0000 0001 0101 0100　0154		0000 0000 0001 1100 0　001C
	0000 0000 1010 1010 0　00AA	補正	0000 0000 0000 0011　を引く
補正	0000 0000 0011 0011　を引く		0000 0000 0001 1001　0019
	0000 0000 0111 0111　0077		0000 0000 0000 1100 1　000C
	0000 0000 0011 1011 1　003B	補正	0000 0000 0000 0011　を引く
補正	0000 0000 0000 0011　を引く		0000 0000 0000 1001　0009

最後一番下の桁が9になりました。これは4ビットの2進数にすでになっていますので、これを踏まえて、2進数を取り出しますと、

10011010010

となりました。こちらの処理と前の10をかける方法とどちらが高速かというと、回路の量を増やすことができるのであれば乗算のほうが分があると思いますが、実際の選択は高速化と回路の量とで判断することになります。

以上のようにCPUの高速化には、いろいろな要素があり、それぞれの判断基準は、回路の量との見比べによる判断となります。最近、コストパフォーマンスという言葉が携帯のCMなどでも使われ一般的になってきていますが、CPUの開発はまさにコストパフォーマンスの検討に終始することをご理解いただけますでしょうか。

以上をもちまして、CPUのご紹介は終了させていただきたいと思います。