１５．乗除算と固定小数点演算

８ビット表現において，１０進数の「10」は２進数では

```
10D=00001010B
```

であり，また，「20」は２進数で

```
20D=00010100B
```

となる。この２つを見比べると「20」の表現は「10」の２進数表現を全体に左に１ビットだけずらした（「シフト（Shift）」という）ものとなっていることがわかる。すなわち，１ビット左シフトすれば「２倍」となるわけである。したがって，２ビット左シフトでは

```
40D=00101000B
```

で「４倍」，３ビット左シフトは「８倍」となる。要するにNビット左シフトは「2^N倍」となる。これは２進数が２の累乗で表されている結果である。この時，一番下位のビット（ＬＳＢ）には「０」が入って来ることに注意されたい。これは，１０進数で「10」を１０倍するには桁を左に１つずらして「100」とすることと同じである。

このように考えると，逆に右に１ビットシフトすれば「1/2 倍」となることも理解できるであろう。すなわち，

```
5D=00000101B
```

となる。さらに，これを右に１ビットシフトすると

```
00000010B=2D
```

となる。この時，「5」の半分は「2.5」であるが，小数部は表現できないので切り捨てとなり，誤差を生じる。また，一番上位ビット（ＭＳＢ）には「０」を入れることに注意されたい。このようにビットのシフトによって「２の累乗の乗除算」ができる。

いままでは正の数を考えたが，負の数ではどうだろうか。例えば，「-10」は２進数で

```
-10D=11110110B
```

である。これを左１ビットシフトすると

```
-20D=11101100B
```

で２倍となる。負の数においても「2^N倍」する時は正の数の場合と同じであり，ＬＳＢには「０」をつめている。それでは，「-10」を「1/2 倍」するために右に１ビットシフトすると

```
01111011B
```

となる。正の数と同じようにＭＳＢに「０」を入れると，これは「+123」となってしまうことがわかる。「-10」の「1/2 倍」は「-5」であり，これは

```
11111011B
```

の２進数表現である。すなわち，負の数の「（１／2^N）倍」もまた負の数となるので，ＭＳＢは「１」でなければならない。このことは正の数の場合と異なる。このようにして，さらに右１ビットシフトすると

```
11111101B
```

となり，これは「-3」である。実際には「-2.5」であり，正の数の場合とは異なった結果

となる。すなわち，ビットシフト操作で得られる整数値は BASIC における「INT（その値を越えない最大の整数値）」命令実行時と同じ結果である。これらをまとめると

「2^N」を得るには「Ｎビット左シフトしＬＳＢに「０」を入れる 「１／2^N」を得るには「Ｎビット右シフトし 扱う数値が正の時にはＭＳＢに「０」を入れる 扱う数値が正負表現の時はＭＳＢの値は不変とする

ということである。これらの操作を行うアセンブラ命令が８章で述べた「算術シフト命令（Arithmetic Shift）」であり，「SAL(SHL)および SAR」命令である。

「２の累乗の乗除算」はシフト操作によって簡単に結果を得ることができ，「乗除算命令（MUL,DIV 等）」を行うよりはるかに速く実行できるが，乗算（左シフト）の場合には扱う数値の大きさに注意が必要である。

２の累乗計算の場合には前述したような方法で求めることができるが，２の累乗でない乗算を行う場合には「MUL(IMUL)」命令を使用するか，あるいは２の累乗のシフト操作で構成したプログラム，例えば，「AX」の５倍の値を求める時は

$AX \times 2^2 + AX$

などのようにしなければならない。後述するＤＳＰでは「乗算命令」は１マシンサイクル（ＣＰＵが命令を実行するためのクロック周波数の１周期）で実行できるので，「乗算」に対してはそのまま「乗算命令」を利用しても演算時間の問題はない。ところが，ＤＳＰの命令には「除算」がないために除算用のプログラムを組まなければならない（通常３０ステップ程度）。したがって，乗算命令を利用して除算するプログラムテクニックによって実行時間の短縮を図る必要があり，ここでこれを説明する。ＤＳＰ命令は後述するので，ＣＰＵで行う場合を述べるが，この方法は実際の制御プログラムで頻繁に使用しているので，必ずマスターされたい。

「5」で除することは「0.2」を乗じることである。「0.2」は「２を乗じて１０で除する（あるいは４を乗じて２０で除するなどと考えてもよく，いくらでもそれらの値は考えられるが）」であるが，これでは乗数も除数も２の累乗ではないために，乗除算命令を使用しなくてはならず，意味がない。そこで，除数を２の累乗に選び，対応する乗数を求める。いま，除数を「64（$=2^6$）」に選ぶと乗数は「64／5=12.8」となるので，この値に近い「13」とする。この乗数を用いて乗算を行うと，例えば「AL=100」に対して

```
ＡＸ＝１００×１３
　＝１３００Ｄ
　＝００００　０１０１　０００１　０１００Ｂ
```

という結果が「AX」レジスタに求まる（バイナリ表現はわかり易いように４ビットづつ区切って記述する）。そこで，これを「64」で除するために「AX」を６ビット算術右シフトを行うと

```
ＡＸ＝０００００　０００００　０００１　０１００Ｂ
```

となり，これは「20」である。すなわち，シフトを行う前の「AX」の内容を

```
ＡＸ＝０００００　０１０１　００．０１　０１００Ｂ
```

のように，あたかも「６ビットの位置に小数点がある」と考えた時の整数部が答えとして得られている。このような考え方が「固定小数点演算」と呼ばれる演算法の基本である。小数点の右１ビット目は「2^{-1}」すなわち「0.5」を表し，右２ビットは「2^{-2}」すなわち「0.25」，Ｎビット目は「2^{-N}」を表している。したがって，上記「AX」は「20.3125」を表現している。この値は当然「100×13／64=20.3125」に等しくなる。

さらに，除数の値を大きくして（除数を大きくすると演算誤差が小さくなる）「256（=2^{8}）」とすると，乗数は「256／5=51.2」ととなるので「51」として，乗算すると

```
ＡＸ＝１００×５１
　　＝５１００Ｄ
　　＝０００１　００１１　１１１０　１１００Ｂ
```

となる。この時の小数点の位置は

```
ＡＸ＝０００１　００１１．１１１０　１１００Ｂ
```

である。ここで，８ビット算術右シフトを行うが，これは「AH」レジスタそのものであり，シフトを行わなくても結果が得られている。ただし，この場合には答えが「19」となっているが，これは乗数「51.2」を「51」と小さい値を用いたためである。小数部を含めると「19.921875」であり，先の乗数を「64」とした時の「20.3125」より「20」に近い値である。正確な値に近い整数値を得るには得られた１６ビットデータを「四捨五入」する必要があり，これは固定小数点データに「0.5」を加えて得られる。「0.5」は８ビット小数点表現では

```
０００００　０００００．１０００　００００Ｂ
```

であり，これを先のデータに加算すれば

```
ＡＸ＝０００１　０１００．０１１０　１１００Ｂ
```

となり，整数部は「20」が得られる（この時の AX の値は 20.421875 を表している）。実際の制御プログラムで扱っているデータは小数部桁が大きので四捨五入が行っていないが，より正確な結果を得るにはこの操作が必要である。

このように除数を８ビットでは「256」，１６ビットでは「65536」に選べば，結果はそれぞれ８ビットでは「AH」レジスタ，１６ビットでは「DX」レジスタに得られる。これは例えば８ビットでは「1」を乗じる場合に「256」を乗じて上位８ビットを取るということであり，単に「1」を「256」に対応させて考えるだけである。扱う数値が正負表現のデータでも同じである。

このような方法は通常「１以上の値で除する，すなわち１以下の値を乗じる」場合には問題がない。しかしながら，時によってはあるデータを「1.3」したいということがあるが，この時には扱う数値の大きさを考慮していないとオーバーフローする場合があるので，注意を要する。「1.3」倍の時には被乗数８ビットでは乗数が「333」となり，８ビットで表現できないので１６ビットに符号拡張して同様に計算すればよいが，被乗数１６ビットで

は「85197」となり，計算不可能となる。このような時には次のように演算を工夫する。すなわち，「1.3」倍を

1.3＝1＋0.3

とし，小数部を上述した方法で計算し，その結果に被乗数を加える（１倍は被乗数そのものである）。これは

AX×1.3＝AX×1＋AX×0.3

である。ただし，小数部の演算結果は上位（この場合は「DX」レジスタ）であるので，「AX×1」の部分は上位に加算する。これのプログラミング例は次のようである（被乗数は「AX」レジスタ，結果は「DX」レジスタ）。

```
MOV  BX,AX        ;AX の内容を BX に保存
MOV  CX,19661     ;乗数（0.3×65536）
MUL  CX
ADD  AX,8000H     :小数部に 0.5 加算（四捨五入）
ADC  DX,BX        上位に AX×1 を加算
```

扱う数値が正の数だけであれば全ての計算はこれでよいが，正負表現の時には不都合が起こる場合がある。それは乗数の小数部が「0.5」を越える時である（0.5 の時には被乗数を１ビット右シフトを行えばよい）。この場合には１６ビットデータでは乗数が「32767」より大きくなり，正負表現ではこれを越えると負の値となる。例えば，「1.8」の場合には乗数は「52429」となり，これは上記と同じ計算では「-0.2」になるので，結果は被乗数の「-0.8」倍となる。そこで，このように小数部が「0.5」を越える時には

1.8＝2－0.2

のように計算し，常に小数部が「0.5」を越えないように行う。「0.2」は「13107（=65536-52429）」であるから，小数部演算に対して乗数に「13017」を用いると２倍した整数部からの減算，乗数に「52429」を用いると２倍した整数部への加算となる。

これらの演算法は既知の定数で除算する場合に適用できるが，未知数での除算は「DIV(IDIV)」命令を使用しなければならない。

コンピュータで扱う数値は整数であるが，これを整数とみなすか実数とみなすかはプログラマの考え方の違いであり，それによってプログラミングが異なる。整数値とする時，小数点はＬＳＢの右側，すなわち１６ビットでは

```
ＸＸＸＸ　ＸＸＸＸ　ＸＸＸＸ　ＸＸＸＸ．Ｂ
```

であると考える。また，前述したように８ビット目に小数点があると考えた場合には

```
ＸＸＸＸ　ＸＸＸＸ．ＸＸＸＸ　ＸＸＸＸＢ
```

いずれの表現でも「固定小数点表示（Fixed-Point Representation）」であるが，後者の場合には小数部も表現されているので，これを特に「分数固定小数点表示（Fractional Fixed-Point Representation）」という。これらの表現を使用した演算が「固定小数点演算（Fixed-Point Arithmetic）」であり，制御プログラムではこれを使用している。

一方，実数を仮数部と指数部に分けて表示した場合を「浮動小数点表示（Floating-Point Representation）」という。これは例えば，１６ビットの内，１２ビットを仮数部，４ビットを指数部に分けて表示し，

（仮数部）×２$^{（指数部）}$

として表す。この表現による演算は行っていないので，詳細はマニュアル等を参考にされたい。

固定小数点について，実際に数値を使って説明するが，わかり易いように電圧，電流と抵抗を使ってオームの法則を計算する。固定小数点の場合，まず，それぞれの値に対して小数点の位置を何処に置くか（小数部のビット数をいくらにするか）を決めなければならない。これにより，表現できる値の大きさと分解能が決まる。ここでは，各値を１６ビットで表現することにし，電流を小数部８ビットとする。すなわち

```
（整数部）　（小数部）
XXXX XXXX.XXXX XXXXB
```

正負表現のため，ＭＳＢは符号を表すので，このデータの最大値は

```
0111 1111.1111 1111B = 127.99609375 (A)
```

であり，単位は一応「Ａ」とする。また，最小値は

```
1000 0000.0000 0000B = -128 (A)
```

となり，この範囲の値を表現できることになる。この時の分解能はＬＳＢが「2^{-8}」を表すので

```
1 LSB = 0.00390625 (A)
```

となり，この精度で表現可能である。これは当然のことながら

```
128 - 0.00390625 = 127.9960375 (A)
```

である。実際に実験で流れる電流値は高々２０Ａ程度であるから，電流の表現は小数部８ビットで十分である。小数部ビットを大きくすれば精度はよくなるが，それほど精度を上げる必要はない。

抵抗値も電流と同じ精度としておく（単位は「Ω」）。

電圧は実際には３００Ｖ以上の値を取り得るので，小数部８ビットでは表現できないため，精度を落として小数部４ビットとすると，その最大値は

```
0111 1111 1111.1111B = 2047.9375 (V)
```

であり，単位は一応「Ｖ」とする。また，最小値は

```
1000 0000 0000.0000B = -2048 (V)
```

となり，分解能は

```
1 LSB = 0.0625 (V)
```

である。以上のように小数部ビット数を決める。

では，電流 I=5.5(A)，抵抗 R=6.1(Ω)の場合の電圧 V を求める。実際の計算では，得られる電圧は

```
V = IR = 5.5×6.1 = 33.55 (V)
```

である。電流と抵抗の固定小数点表示は

```
電流 I = 1408D = 0580H = 0000 0101.1000 0000 = 5.5 (A)
抵抗 R = 1562D = 061AH = 0000 0110.0001 1010 = 6.1015625 (Ω)
```

である。ただし，抵抗は6.1(Ω)に最も近い値を使う。これを「IMUL」命令で積を計算すると，その結果「DX:AX」の３２ビットデータは

```
00218F00H = 00000000 00100001.10001111 00000000
```

となる。これは，このままの小数点位置であれば

```
33.55859375 (V)
```

を表している。ここで，小数部８ビット表現同士の積を行うと，得られる結果は小数部１６ビット表現となることに注意されたい。電圧は小数部４ビットとするので，この３２ビットから小数部４ビットを含む１６ビットデータを取り出すと（１２ビット右シフトして下位１６ビットを得るか，あるいは４ビット左シフトして上位１６ビットを得る）

```
00218F00H = 00000000 00100001.10001111 00000000
                          ↓
                0000 00100001.1000 = 0218H
```

となる。この値は「33.5(V)」であり，実際値と「0.05(V)」の誤差を生じる。これは演算結果の小数部８ビット表現を小数部４ビットとしたためであり，分解能による。誤差を小さくするためには小数部のビット数を大きくしなければならない。乗算では小数部Ｎビットと Mビットの積は小数部（Ｎ＋Ｍ）ビット表現の数値となる。

次に，電圧 V=123.7(V)，抵抗 R=8.5(Ω)の場合について電流を計算する。電流の解は

```
I = V／R = 123.7／8.5 = 14.55294118 (A)
```

である。電圧と抵抗の固定小数点表示は

```
電圧 V = 1979D = 07BBH = 0000 0111 1011.1011 = 123.6875 (V)
抵抗 R = 2176D = 0880H = 0000 0110.1000 0000 = 8.5 (Ω)
```

である。ただし，電圧は 123.7(V)に最も近い値を使う。これを計算するために「IDIV」命令を使うが，抵抗は１６ビット表現なので，電圧を３２ビット表現に符号拡張して行うと，

```
0000 07BBH／0880H = 0 (A)
```

となり，正しい結果が得られない。これは小数桁ビットの対応を考慮していないためである。すなわち，乗算とは逆に，除算の場合には被除数が小数部Ｎビットで除数が小数部Mビットでは，商の小数部は（Ｎ－Ｍ）ビットとなるためである。上の計算では商の小数部ビット数は「－４」となっている。したがって，小数部８ビットの商を得るためには，あらかじめ被乗数を１２ビット左シフトして小数部１６ビット表現としておく必要がある。すなわち，電圧は３２ビット表現で

```
      0000 0111 1011.1011
                ↓
0000 0000 0111 1011.1011 0000 0000 0000 = 007B B000H
```

とする。これで演算を行うと

```
007B B000H／0880H = 0E8BH
```

であり，これは小数部８ビット表現となっているので

```
0000 1110.1000 1101 = 14.55078125 (A)
```

となる。結果の誤差は電圧の表現誤差によるものである。除算の場合には商として得る値

の小数部ビット数を考慮して演算する必要がある。

固定小数点演算においては演算誤差すなわち有効桁数に注意しなければならない。

上述の演算をもう少しわかり易く記述すると以下のようである。電流と抵抗は小数部８ビットとしており，これらの１６ビット値は単に

「1」を「２５６＝2^8」に対応（実際値の２５６倍）

させ，電圧は

「1」を「１６＝2^4」に対応（実際値の１６倍）

させている。乗算では

```
I = 5.5 (A) = 5.5×256 = 1408
R = 6.1(Ω) = 6.1×256 = 1561.6 → 1562
```

とし

```
IR = (1408/256)×(1562/256) = (1408×1562)／(256×256)
                           = 2199296／(256×256)
                           = 00218F00H／65536
```

であり，結果は実際値の「６５５３６」倍となる。１２ビット右シフトは「1／2^{12}」倍すなわち「1／４０９６」倍することであるから

```
V = 2199296／4096 = 536.9375 → 536 = 0218H
```

となり，実際値の１６倍の結果が得られる。

一方，除算では

```
V = 123.7 (V) = 123.7×16 = 1979.2 → 1979
R = 8.5(Ω) = 8.5×256 = 2176
```

とし，電圧は１２ビット左算術シフトするので４０９６倍とし

```
V = 1979×4096
```

となり。この時点で電圧は実際値の「2^{4+12}」倍となっている。これを２５６倍の抵抗で除するので

```
V／R = (1979×4096)／2176 = 3725.176471 → 3725 = 0E8DH
```

小数部ビット数で表現すれば，乗算では

```
(I×2^8)×(R×2^8) = IR×2^16
```

これを１２ビット右シフトして

```
IR×2^(16−12) = IR×2^4
```

となる。除算では，まず電圧を１２ビット左シフトするので

```
V×2^(4+12) = V×2^16
```

であり，

```
I = (V×2^16)／(R×2^8) = V／R×2^(16−8) = V／R×2^8
```

となる。

このような演算において，乗算ではオーバーフローは起こらないが，除算では商が１６ビットに納まるよう，扱う数値の大きさに注意しなければならない。