電子計算機 **2E**

3E　2003.5.28

本日の授業のテーマ

本日の事業のテーマは、以下のとおりです。

(1) 固定小数点表示
(2) 不動小数点表示
(3) プログラミング言語と表現方式

本日の授業のゴールは、以下のとおり。

- 固定小数点表示の意味が理解できる。
- 不動小数点表示の内容が理解できる。
- コンパイラー言語で変数の宣言が必要な理由がわかる。

0. 先週の復習

・負の数の代表的な表現方法には、①絶対値表現　②1 の補数による表現 ③2 の補数による表現があります。

・これらのうち、現在のコンピューター内部での負の数の表現は、2 の補数が使われています。 たんに、補数と言えば、2 の補数のことです。

・コンピューター内部では、負の数は 2 の補数による表現を行います。その手順は、以下の通りです。

① 負の数の絶対値を 2 進数で表現して、ビット反転する。
② +1 加算

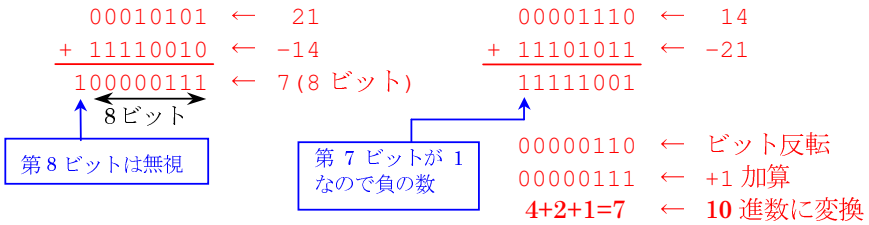
[例] $(-18)_{10}$ は、コンピューター内部、8 ビット表現では、$(11101110)_2$ と表されます(メモリーへの格納状態)。

$(-18)_{10}$	00010010	←	18 の 2 進数表現
	11101101	←	ビット反転
	11101110	←	+1 加算

・2 の補数を使うと、以下の有利な点があります。

- 負の数の加算が通常の加算器で出来る。
- 正の数の減算をする場合、① 2 の補数に変換して、② 加算器による加算で可能となる。減算器を作るより、この方が回路が簡単になる。

[例] $(21-14)_{10}$ と $(14-21)_{10}$ を加算器(8 ビット)を使って計算する。



・補数を求める手順　(①ビット反転 ②+1 加算) は、コンピューター内部表現では、×(-1) と同じです。

・コンピューター内部では、正の数は 2 進数でそのままの表現です。一方、負の数は 2 の補数を使います。正か負かの判断は、最上位のビットで判断します。最上位のビットが 0 ならば正、1 であれば負です。

・したがって、最上位のビットが符号を表すため、絶対値は残りのビットで表すことになります。したがって、8 ビットの場合、以下の範囲で整数が表現できます。

最大値　$(01111111)_2=(2^7-1)_{10}=(127)_{10}$
最小値　$(10000000)_2=(-2^8)_{10}=(-128)_{10}$

1 固定小数点表示

2 進数で、小数点の位置を固定した表現です。 一般的に、右端に小数点を固定します。したがって、整数のみを表わすることになります。 負の数は、2 の補数をつかいます。例えば、8 ビットで$(-107)_{10}$を表現すると、図 1 のようになります。計算機内部で、$(-107)_{10}$は(10010101)と表現されます。

これは、いままで、さんざん学習してきたので、これ以上の説明はしません。

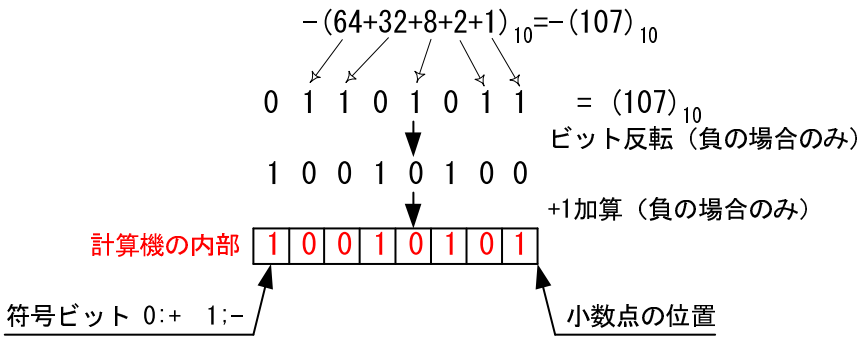


図 1 $(-107)_{10}$の固定小数点表示

2 浮動小数点表示

2.1 浮動小数点表示とは

浮動小数点表示とは、指数化（例えば、-0.123 ~1^{-2}）して数値を表現します。これは非常に便利な方法で、自然科学では多くの場合、このように数値を表現します。コンピューターでも同様で、データが整数と指定されない限りこの浮動小数点が用いられます。実際、この仮数部の (-0.123)と指数の(-2)をメモリーに格納します。この方法の長所と短所は、以下の通りです。

長所　決められたビット数内で、非常に小さな数値から大きな数値まで表現可能になる。
短所　桁落ち誤差が発生する場合がある。

浮動小数点表示を学習するために、必要な言葉の意味は、図 2 の通りです。1 年生の数学の授業で学習したはずです。

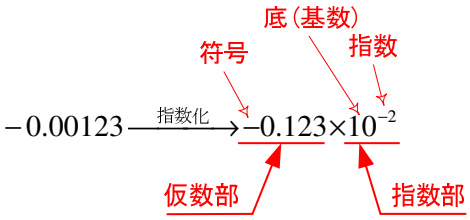


図 2 指数表現の名称

ここでは、教科書に示されている 2 通りの方法と、秋田高専の計算機サーバー(な

まはげ君)での C 言語[1]での固定小数点表示を示します。

2.2 オフセットバイナリを使う方法(教科書)

教科書にはきっちりと仕様が書かれていません。そこで、補数は使わないで、絶対数表現ということで話を進めます。教科書の不足分を補っているだけで、教科書とは矛盾はしません。

まず例として、

$$(-0.0390625)_{10} = (-0.0000101)_2 = (-0.0a)_{16} = (-0.a\times10^{-1})_{16}$$

を浮動小数点表示することを考えます。10 進数のを 16 進数で正規化しています。正規化というのは、仮数部の整数部をゼロにして、小数点一桁目を 1～F までの数字で表すことを言います。

正規化後、以下の操作をすれば、浮動小数点表示ができます。

① 負の数なので符号ビットは、1 です。
② 指数$(-1)_2$は、10 進数で表しても、$(-1)_{10}$です。これを教科書の表 2.2 にあるオフセットバイナリ-方式で、2 進数へ変換します。オフセット値は 64 なので、それを加算して、$(-1+64)_{10} = (111111)_2$となります。指数は、7 ビットで表すため、$(011111)_2$と表現します。
③ 次に仮数部ですが、$(0.a)_{16} = (0.1010)_2$です。残りのビットはすべてゼロです。即ち、$(0.1010)_2 = (0.10100000000000000000000)_2$です。

以上ですべてのビットが決まったので、コンピューター内部では、図 3 のように表現されます。

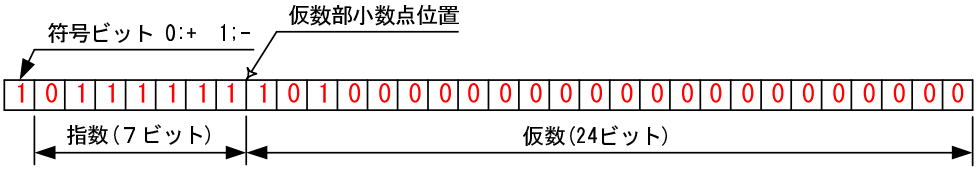


図 3 $(-0.0390625)_{10}$をオフセットバイナリ-を使った浮動小数表現

2.3 指数符号ビットを使う方法(教科書)

これも、教科書にはきっちりと仕様が書かれていませんので、補数は使わないで、絶対数表現ということで話を進めます。

ほとんど、先ほどのオフセットバイナリ-を使う方法と同じです。異なるのは、指数部がオフセットバイナリではなく、絶対値表現になっていることだけです。先ほどと、同じ問題、$(-0.0390625)_{10}$の表現を考えます。先ほどの操作の①と③は全く同じです。②は、

② 指数-1 は、負の数なので指数符号ビットは 1 になります。その絶対値の$(1)_{16}=(1)_2$です。これを 6 ビットで表すと、$(000001)_2$です。

[1] IEEE (Institute of Electrical and Electronic Engineers)の定める標準形式になっている。その double 型の内部表現をここでは示します。

となります。これで、すべてのビットが決まったので、コンピューターの内部の表現は、図 4 のようになります。

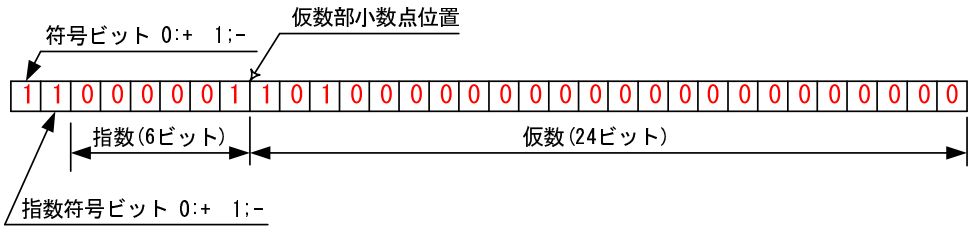


図 4 $(-0.0390625)_{10}$ を指数符号ビットを使った浮動小数表現

2.4 C 言語の倍精度実数の表現

IEEE の規格の C 言語の倍精度実数型の double の表現について説明します。まず、浮動小数点表示のための正規化を図 5 に示します。当然、仮数部、指数部とも 2 進数表現です。仮数部は、符号と 1.XXXX の表現にします。

$$(-0.0007696151733398438)_{10} = (-1.100100111\times10^{-1011})_2$$

仮数部　　指数部

図 5 IEEE 規格表現のための規格化。

つぎに、これを IEEE 規格の浮動小数点に表すことを考えます。まずその規格の仕様は、以下のようになっています。

- 64 ビット(第 0 ビット～第 63 ビット)で、浮動小数を表します。各ビットの構成は、図 6 の通りです。
- 最上位の第 63 ビットが仮数部の符号ビットです。正の場合ゼロで、負の場合 1 になります。
- 指数は 11 ビットでオフセットバイナリ方式で表します。11 ビットで 0～2047 の値になります。ただし、指数部 11 ビットの値 0 と 2047 は例外処理のために予約されています。11 ビットで表現される値からオフセット値 1023 を引くことにより指数の値が -1022～1023 の範囲になるように定められています。
- 仮数部は 52 ビットです。小数点以下を、絶対値で表現します。規格化のための整数部は 1 と分かっているので、このためのビットは割り当てられていません。

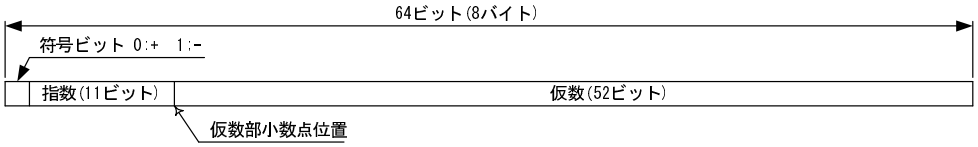


図 6 IEEE 規格 (C 言語の倍精度実数) 表現のビットの内訳。

以上の仕様をもとに、図 5 で規格化された数を浮動小数点表示します。ほとんどの部分は規格化で分かりますが、指数のみ計算が必要です。指数は、オフセットバイナリーで計算するために、まず 10 進数で表します。

$$(-1011)_2 = (-8-2-1)_{10} = (-11)_{10} \qquad (1)$$

不動小数表示の指数は、この式(1)の実際の値に、1023 を加算して求めます。すると、

$$(-11+1023)_{10} = (1012)_{10} = (1111110100)_2 \qquad (2)$$

となります。

これで、すべて準備が整いました。不動小数点表示は、図 7 のようになります。実際のコンピューターには、この 64 ビットは、８ビット(1 バイト)毎アドレスが割り当てられたメモリーに格納します。したがって、メモリーの 8 番地分のデータ領域が必要になります。

$$(-0.0007696151733398438)_{10} = (-1.100100111\times10^{-1011})_2$$

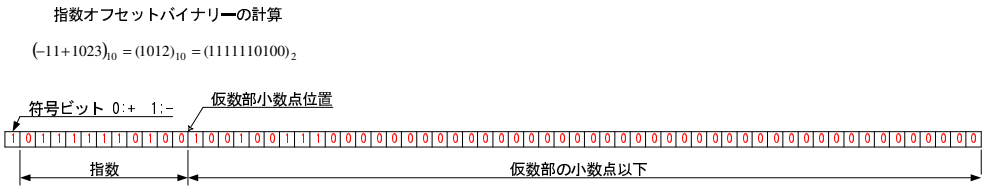


図 7 IEEE 規格の浮動小数点表示の例。

3 プログラム言語と表現方式

初心者がプログラミング言語を学び始めると、変数の宣言に戸惑います。数学を勉強するとき、変数に宣言など使ったことがないからです。また、宣言がなくても、アルゴリズムに不都合がないからです。変数の宣言が無くても、問題がないように思ってしまいます。

変数の宣言がなくても、アルゴリズム上、問題はありません。しかし、コンパイラーにとっては、大きな問題です。コンパイラーがソースプログラムをマシン語に翻訳する場合、その変数を表現するための領域の大きさ(バイト数)と表現方法を決める必要があります。

できるだけ大きな数字が表現できる型、例えば FORTRAN の 4 倍精度実数型に統一する方法もあります。しかし、そうすると以下のような問題が生じます。

- 整数どうしの加算や減算、乗算が整数にならない場合が生じる。
- 大きなメモリー領域が必要になり、資源の無駄になる場合が多い。

これらを、コンパイラー言語では、型宣言によって避けます。効率的なマシン語を作成するために、変数の宣言は絶対に必要となるのです。参考のために、表 1, 2 に FORTRAN と C 言語の代表的な宣言を示します。

一方、インタープリンター方式の言語の場合、型の宣言が無い場合があります。例えば、BASIC や Perl のような場合[2]、型宣言は不要です。あらかじめ、データ領域を確保する必要が無く、必要になったら、適当に領域を確保しているためだと思います。詳細は、良くわかりません。

表 1 FORTRAN の場合の変数の型表現。

型	宣言	バイト数	表現方式
整数	`INTEGER`	4	固定小数点
実数	`REAL`	4	浮動小数点
倍精度実数	`REAL*8`	8	浮動小数点
4 倍精度実数	`REAL*16`	16	浮動小数点

表 2 C 言語の場合の変数の型表現(秋田高専の計算サーバー)。

型	宣言	バイト数	表現方式
整数	`int`	4	固定小数点
実数	`float`	4	浮動小数点
倍精度実数	`double`	8	浮動小数点
拡張倍精度実数	`long double`	8	浮動小数点

[2] しかし、近頃、型宣言を行うようにもなってきています。昔は、不要でした。