IEEE754 準拠マシンでの丸めモード制御

桂田祐史

2000 年 12 月 26 日

1　丸めモードの制御法

1.1　round.h

```
/*
 * round.h -- 丸めモードの指定
 *  大石進一、精度保証付き数値計算、コロナ社（2000）を参考にした
 */

#ifndef ROUND_ALREADY_INCLUDED

#define ROUND_ALREADY_INCLUDED

#if defined(__FreeBSD__)

/*
 * FreeBSD は Linux と違って、浮動小数点数の標準精度が 53 bits なので
 * 大石先生の本にある CW の値を採用すると、矛盾が生じることに注意。
 * ここでは FreeBSD で元から用意されている手順を採用する。
 */

#include <machine/floatingpoint.h>
#define Near() fpsetround(FP_RN)
#define Up() fpsetround(FP_RP)
#define Down() fpsetround(FP_RM)

#elif defined(sparc)

/*
 * これは大石先生の本にあるものを採用した。
 */

/* #define QUICK */

#ifdef QUICK

/* これは大石先生の本にも載っているが、Bias でもこうなっている */
static int _RoundNear = 0x00000000L;
static int _RoundUp = 0x80000000L;
static int _RoundDown = 0xC0000000L;
#define Near() asm volatile("ld %0,%%fsr" : : "g" (_RoundNear))
#define Up() asm volatile("ld %0,%%fsr" : : "g" (_RoundUp))
#define Down() asm volatile("ld %0,%%fsr" : : "g" (_RoundDown))
```

```
#else

#ifdef FALSE /* 大石先生の本にはこういうのが使えると書いてあるが使えない */
#include <floatingpoint.h>
#define Near() ieee_flags("set", "direction", "nearest", &out)
#define Up() ieee_flags("set", "direction", "positive", &out)
#define Down() ieee_flags("set", "direction", "negative", &out)
#endif

/* これはオンライン・マニュアルから */
#include <ieeefp.h>
#define Near() fpsetround(FP_RN)
#define Up() fpsetround(FP_RP)
#define Down() fpsetround(FP_RM)

#endif

#else
 /* FreeBSD でも Sparc でもなければ、とりあえず Bias まかせ */
 /* Cygwin の場合は FreeBSD のときと同様で良いかな？ */

#include <BiasInt.h>
#define Near _BiasRoundNear
#define Up _BiasRoundUp
#define Down _BiasRoundDown

#endif

#endif
```

2 実験

2.1 丸めモードを up にするとアンダーフローしない！

浮動小数点数のシステムは、絶対値が大きな数や絶対値が小さな数は全く表現できないという性質がある。

計算の結果生じるはずの値の絶対値が小さくなって浮動小数点数では表現できなくなったとき「アンダーフローが生じる」と言い、例外を発生したり、値を 0 に置き換えて計算を続行したりする。

ところで丸めモードを up にしておくと、アンダーフローが起こらない。それを見るのが次の実験である。1 を 2 で割る操作を続けて行って 0 になってしまうかどうか調べている。

以下のプログラムでは、 x を 2 で割ってから、x の値が 0 に等しいかどうかチェックする間に、「意味のない」実行文をはさんでいることに注意してもらいたい。これをしないと、x は CPU の register に載ったままになり、register 上で表現可能な値の範囲は long double のそれと (ほぼ) 一致してしまい[1]、簡単にはアンダーフローしなくなる (long double で表現可能な値の範囲を越えて初めてアンダーフローするので)。

[1]実験に用いたマシンの OS は FreeBSD である。「FreeBSD では、CPU の計算精度は通常 53 bits に設定されているので、精度 は double 相当である」と説明されることが多い。しかし、指数部 の長さには変化がないので、register 上で表現可能な値の範囲は、long double 相当になっている。

test-round0.c

```
/*
 * test-round0.c
 *
     数列 2^{-n} を計算すると、大きな n に対してアンダーフローして 0 に
     なってしまう、というのが常識であるが、丸めモードが up であると
     そうはならない。
 */

#include "round.h"
#include <stdio.h>

void dummy() {}

void try()
{
  int i, n;
  double x, oldx;
  n = 10000;
  x = 1.0;
  for (i = 1; i <= n; i++) {
    oldx = x;
    x /= 2;
    /* 何か挟まないと x がレジスターに載っていて、
       long double のレンジを越えない限りアンダーフローしない */
    dummy();
    if (x == 0.0) {
      printf("2^{-%d} が 0 になりました。\n", i);
      printf("2^{-%d+1}=%g\n", i, oldx);
      break;
    }
  }
  if (i > n) {
    printf("%d 回 2 で割っても 0 にはなりませんでした。x=%g\n", n, x);
  }
}

int main()
{
  Near(); printf("Near\n"); try();
  Down(); printf("Down\n"); try();
  Up();   printf("Up\n");   try();
  return 0;
}
```

test-round0 の実行結果

```
Near
2^{-1075} が 0 になりました。
2^{-1075+1}=4.94066e-324
Down
2^{-1075} が 0 になりました。
2^{-1075+1}=4.94066e-324
Up
10000 回 2 で割っても 0 にはなりませんでした。x=4.94066e-324
```

2.2 計算を繰り返すと丸め誤差が蓄積される

test-round1.c

```
/*
 * test-round1.c --- 繰り返し計算をすると丸め誤差がたまることの確認
 */

#include "round.h"
#include <stdio.h>

int main()
{
  int i, n;
  double s;
  n = 10000;

  printf("多分、小数点以下 11 桁目あたりで差が生じる\n");

  Down(); printf("Down\n");
  s = 0.0; for (i = 1; i <= n; i++) s += 1.0 / i;
  printf("s=%25.16f\n", s);
  Near(); printf("Near\n");
  s = 0.0; for (i = 1; i <= n; i++) s += 1.0 / i;
  printf("s=%25.16f\n", s);
  Up(); printf("Up\n");
  s = 0.0; for (i = 1; i <= n; i++) s += 1.0 / i;
  printf("s=%25.16f\n", s);

  return 0;
}
```

test-round1 の実行結果

```
多分、小数点以下 11 桁目あたりで差が生じる
Down
s=       9.7876060360364807
Near
s=       9.7876060360443482
Up
s=       9.7876060360527308
```

ちなみに Mathematica で計算すると、.....

2.3 計算機イプシロンを求める定跡手順をやってみると

test-round2.c

```
/*
 * test-round2.c --- 有名な計算機ε を求めるコードを丸めモードを変えて実行
 */

#include "round.h"
#include <stdio.h>

void find_eps()
{
  int i;
  double eps;
  eps = 1.0;
  for (i = 1; i <= 10000; i++) {
     eps /= 2.0;
     /* ここで eps == 2^{-i} である (アンダーフローしない限り) */
     if (1.0 + eps == 1.0) {
       printf("eps=%g=2^{-%d}\n", eps, i);
       printf("eps_M=%g\n", 2*eps);
       return;
     }
  }
  printf("計算機ε は求まりませんでした。eps=%g に対して 1+eps>1\n", eps);
}

int main()
{
  printf("有名な計算機ε を求めるコードを三つの丸めモードで動かす\n");
  printf("\nrounding to the nearest even\n"); find_eps();
  printf("\nrounding toward the minus infinity\n"); Down(); find_eps();
  printf("\nrounding toward the plus infinity\n");   Up();   find_eps();

  printf("... 上への丸めでは計算機ε は求まらない\n");
  return 0;
}
```

test-round2 の実行結果

```
有名な計算機ε を求めるコードを三つの丸めモードで動かす

rounding to the nearest even
eps=1.11022e-16=2^{-53}
eps_M=2.22045e-16

rounding toward the minus infinity
eps=1.11022e-16=2^{-53}
eps_M=2.22045e-16

rounding toward the plus infinity
計算機ε は求まりませんでした。eps=4.94066e-324 に対して 1+eps>1
... 上への丸めでは計算機ε は求まらない
```