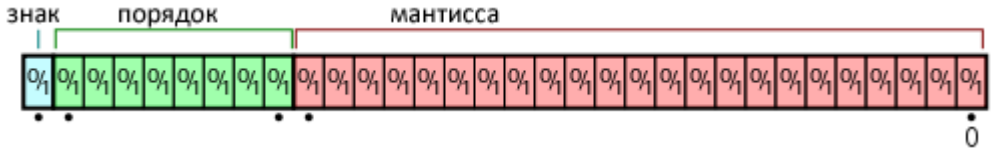
Реализация математической библиотеки для ARM архитектуры.

Мкртчян Тигран Мушегович (рук. Пушков Роман Львович).
Кафедра «Компьютерные системы управления».
2492460@gmail.com

Представление чисел в формате с плавающей запятой.

Число с плавающей запятой состоит из набора отдельных разрядов, условно разделенных на знак, экспоненту порядок и мантиссу. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:



На сегодняшний день существует стандарт IEEE754. Числа с плавающей запятой в нем представлены в виде знака (s), мантиссы (M) и порядка (E) :

Хочу добавить, что существование одного стандарта не означает , что любая программа будет выдавать один и тот же результат на разных компьютерах. На портабельность и соответствие стандарту влияет компилятор и опции оптимизации.

Проблемы с точностью.

1) **Сравнение.**

Очень распространенная ошибка при работе с float-ами возникает при проверке на равенство. Например,

```
float fValue = 0.2;
if (fValue == 0.2) DoStuff();
```

Ошибка здесь, во-первых, в том, что 0,2 не имеет точного двоичного представления, а во-вторых 0,2 – это константа двойной точности, а переменная fValue – одинарной, и никакой гарантии о поведении этого сравнения нет.
Лучший, но все равно ошибочный способ, это сравнивать разницу с допустимой абсолютной погрешностью:

```
if (fabs(fValue – fExpected) < 0.0001) DoStuff(); // fValue=fExpected?
```

Недостаток такого подхода в том, что погрешность представления числа увеличивается с ростом самого этого числа. Так, если программа ожидает «10000», то приведенное равенство не будет выполняться для ближайшего соседнего числа (10000,000977). Это особенно актуально, если в программе имеется преобразование из одинарной точности в двойную.

2) **Округление.**

С ошибками из-за погрешностей округления в современной арифметике с плавающей запятой встретиться сложно, особенно если использовать двойную точность. Правило округления в стандарте IEEE754 говорит о том, что результат любой арифметической операции должен быть таким, как если бы он был выполнен над точными значениями и округлен до ближайшего числа, представимого в этом формате. Это требует от АЛУ дополнительных усилий и некоторые опции компилятора (такие как «-ffast-math» в gcc) могут отключить такое поведение. Особенности округления в IEEE754:
-Округление до ближайшего в стандарте сделано не так как мы привыкли. Математически показано, что если 0,5 округлять до 1 (в большую сторону), то существует набор операций, при

которых ошибка округления будет возрастать до бесконечности. Поэтому в IEEE754 применяется правило округления до четного. Так, 12,5 будет округлено до 12, а 13,5 – до 14.
-Самая опасная операция с точки зрения округления в арифметике с плавающей запятой — это вычитание. При вычитании близких чисел значимые разряды могут потеряться, что может в разы увеличить относительную погрешность.
-Для многих широко распространенных математических формул математики разработали специальную форму, которая позволяет значительно уменьшить погрешность при округлении. Например, расчет формулы «x2-y2» лучше вычислять используя формулу «(x-y)(x+y)».

3) Числовые константы.

Помните, что не все десятичные числа имеют двоичное представление с плавающей запятой. Например, число «0,2» будет представлено как «0,200000003» в одинарной точности. Соответственно, «0,2 + 0,2 ≈ 0,4». Абсолютная погрешность в отдельном случае может и не высока, но если использовать такую константу в цикле, можем получить накопленную погрешность.
Этими тремя пунктами проблемы floating point не ограничиваются.
В идеале нам бы хотелось иметь точность до 1 мкм. А также, чтобы производить перемещение(для крупных станков) при резании на 1-10м.
Также , следует отметить, что возникают некоторые проблемы при работе с другими архитектурами, так как есть вероятность того, что на них не будет поддержки чисел с плавающей запятой, либо она будет аппаратной, что в своё время может означать финансовые затраты, как например архитектура ARM.
Архитектура ARM (Advanced RISC Machine, Acorn RISC Machine, усовершенствованная RISC-машина) — семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited.
Архитектура ARM поддерживается Unix и Unix-подобными ОС GNU/Linux, BSD, QNX, Plan 9, Inferno, Solaris, MacOS, iPhone OS, WebOS и Android.
Поддержка других операционных систем
Операционные системы, которые работают на ARM: ReactOS[48][49], FreeRTOS, Nucleus, Symbian OS, Windows CE, RISC OS, Windows 8[50], .NET Micro Framework.
По состоянию на 2009 на процессоры ARM приходится до 90 % всех встроенных 32-разрядных процессоров. Процессоры ARM широко используются в потребительской электронике, но нас, как разработчиков систем ЧПУ интересуют ПЛК , которые используют ARM архитектуру.
Данные процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и доминируют на рынке. Архитектура ARM имеет решения для чисел с плавающей запятой, но все они аппаратные.
Технология VFP (Vector Floating Point, вектора чисел с плавающей запятой) — расширение сопроцессора в архитектуре ARM. Она производит низкозатратные вычисления над числами с плавающей запятой одинарной/двойной точности, в полной мере соответствующие стандарту ANSI/IEEE Std 754—1985 Standard for Binary Floating-Point Arithmetic. Существуют и другие сопроцессоры, поддерживающие работу над числами с плавающей запятой для архитектуры ARM.
К сожалению расширение для чисел с плавающей запятой есть не во всех моделях ARM, более того наличие данного расширения является редкостью на микропроцессорах ARM.
В результате обеспечить нужную точность чисел, при написании программ для ARM без аппаратной поддержки становится задачей отнюдь не тривиальной.

Решение проблемы путём разработки математического аппарата.

Идея заключается в том, чтобы создать класс, а в будущем, возможно, и библиотеку математических функций, то есть разработки математического аппарата с фиксированной запятой и заданной точностью, которая будет полностью основана на типе int64. Тип выбран с учётом того, что числа должны обладать определённой точностью (На данный момент нас интересует точность 7 знаков после запятой, точность в 1мкм). На сегодняшний день таких решений нет, а если они и есть, то математический аппарат слишком слаб для того чтобы использовать его при разработке в системах ЧПУ. Математическая библиотека должна иметь реализацию тригонометрических функций , логарифмов. Тригонометрические функции нужны для расчёта кривых линий.

Примеры реализации.

Здесь я хотел бы продемострировать методы для численного вычисления чисел с плавающей запятой, так как они будут нужны для перехода от int64 в число с фиксированной запятой.

Стандартные операции.

Сложение и вычитание чисел с фиксированной запятой — это обычные сложение и вычитание: $(x \pm y)' = x' \pm y'$.

Умножение и деление отличаются от целочисленных на константу.

$$(x \cdot y)' = [x' \cdot y' \cdot z] = \left[\frac{x' \cdot y'}{u}\right]$$

$$\left(\frac{x}{y}\right)' = \left[\frac{x'}{z \cdot y'}\right] = \left[\frac{x' \cdot u}{y'}\right],$$

где [] — операция округления до целого.

Для других операций, помимо обычных рядов Тейлора и итерационных методов, широко применяются вычисления по таблице.

Если операнды и результат имеют разную цену (вес) младшего разряда, формулы более сложны — но иногда такое приходится делать из-за большой разницы в порядке величин.

Для решения наших задач U берём равным 7 и пишем программный код.

Как говорилось выше для математические функции можно описать с помощью численных методов представления, как например, разложить синус или косинус в ряд Тейлора :

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}, x \in \mathbb{C}$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}, x \in \mathbb{C}$$

Здесь также мы встречаем проблемы связанные с оптимизацией вычислений, так как добиться надобной точности при численных методах вычисления за маленькое время не так-то легко. Также ,хочу добавить , что используя численные методы мы можем подсчитать логарифмы, арксинусы, квадратный корень.

Вывод.

В результате мы получаем программную реализацию чисел с фиксированной запятой вместе с математическим аппаратом. Видим существенное повышение производительности на системах без FPU при небольшой потере времени на некоторых вычислениях. В то же время мы получаем повышение точности вычислений до удовлетворяющего нас знака и к тому же математический аппарат, удовлетворяющий программированию систем ЧПУ.