# Повышение производительности микроконтроллеров за счёт использования типов данных с фиксированной точкой

**Павлов О.В.[1], Кейно П.П.[1]**
[1]Московский авиационный институт (Национальный исследовательский университет), Москва, Россия
pavlov@smiap.ru

***Аннотация***
*В данной статье рассматривается эффективность применения типов данных с фиксированной точкой, вместо типов данных с плавающей точкой стандарта IEEE-754, для повышения производительности микроконтроллеров.*
*Данный вопрос является актуальным так как большинство программного обеспечения для микроконтроллеров разрабатывается с помощью языков высокого уровня, в основном C/C++, которые изначально разрабатывались для работы с другими архитектурами, что приводит к заимствованию части программных инструментов, оптимальных для архитектур x86/x64, но неоптимальных и избыточных для многих архитектур микроконтроллеров.*
*Проведён анализ популярных микроконтроллеров на архитектурах AVR, AVR32, ARM, ESP-32 на предмет наличия аппаратной реализации операций с типами данных, стандартизированных в IEEE-754. Анализируются методы эмуляции сопроцессора для чисел с плавающей точкой в микроконтроллерах AVR. Приводится анализ скорости выполнения операций сложения, вычитания, умножения и деления на микроконтроллерах AVR для типов данных с фиксированной и плавающей точкой.*

***Ключевые слова**: оптимизация, микроконтроллеры, AVR, ARM, IEEE, ISO/IEC*

С каждым годом автоматизированные системы внедряются во всё большее количество областей производства и жизни людей, яркими примерами этих тенденций являются развитие концепций цифрового производства и индустрии 4.0, а также концепции умного города и дома. Ключевым элементом любых автоматизированных систем остаются микроконтроллеры, осуществляющие получение информации с датчиков, управление различными элементами и связь с различными сервисами и системами. Повышение производительности микроконтроллеров позволяет решать большее количество прикладных задач или повышать эффективность решения текущих задач.

На сегодняшний день большинство программного обеспечения для микроконтроллеров разрабатывается с помощью языков высокого уровня, в основном C/C++, которые изначально разрабатывались для принципиально других архитектур, что приводит к заимствованию части программных инструментов, оптимальных для архитектур x86/x64, но неоптимальных и избыточных для многих архитектур микроконтроллеров.

## 1. Хранение данных в памяти компьютера

На сегодняшний день в большинстве языков программирования и баз данных для хранения дробных чисел используются числа с плавающей запятой описанных в стандарте IEEE 754-2008. Для хранения значений число переводится в двоичную систему счисления, а затем приводится к нормальной экспоненциальной форме вида 1,0110111 × 2^101. После этого истинный порядок преобразуется в смещённый для хранения отрицательного смещения, и все данные записываются в память. Для типа данных binary32, занимающего 4(32 бита) байта памяти и именуемого float в большинстве языков программирования и СУБД, биты распределены следующим образом:

- Знак числа (1 бит)
- Смещённый порядок (8 бит)
- Мантисса (23 бита)

Данный подход к хранению данных позволяет хранить числа в промежутке от $-3{,}4\cdot10^{-38}$ до $3{,}4\cdot10^{38}$.

Также в данный тип включены представление положительного и отрицательного нуля, положительной и отрицательной бесконечностей, а также нечисла (англ. Not-a-Number, NaN). Для хранения и работы с данными специальными значениями требуется реализовывать дополнительную логику.

Проблема с представлением чисел в разных реализациях согласно IEEE-754 может приводить к ошибкам при проверках на равенство, так конструкция простого сравнения при очевидной простоте и правильности может не выполняться.

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

Ключевой проблемой является то 0,2 не имеет точного двоичного представления и будет округлено к ближайшему числу, в зависимости от компилятора константа 0.2 может быть представлена как типом binary32 так и другими типами, например binary64, что приведет к хранению различных значений в переменных.

Одним из вариантов решения этой проблемы является сравнение разницы между значениями с учетом допустимой абсолютной погрешности:

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

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.

Для решения этой проблемы предлагается множество способов, один из них - сравнение чисел с плавающей запятой преобразованием к целочисленной переменной подробно рассмотрен в статье Брюса Доусона [1], а также в других работах [2-6].

## 2. Анализируемые архитектуры микроконтроллеров

Типы данных, описанные в стандарте IEEE-754 [7], прошли долгий путь внедрения в архитектуру персональных и специализированных компьютеров и в итоге наличие сопроцессора для операций с плавающей точкой стало общим стандартом для компьютеров, однако во многих архитектурах микроконтроллеров FPU не реализуется.

### 2.1. AVR

AVR — семейство восьмибитных микроконтроллеров, ранее выпускавшихся фирмой Atmel, затем - Microchip. Данные микроконтроллеры обладают малой стоимостью и популярны в создании электроники.

На популярность данной архитектуры во многом повлияло развитие и популярность платформы Arduino, многие платы которой базируются на микроконтроллерах AVR.

Микроконтроллеры AVR имеют гарвардскую архитектуру и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл.

Стандартные семейства:

tinyAVR (ATtinyxxx):

megaAVR (ATmegaxxx):

XMEGA AVR (ATxmegaxxx):

Ни в одном из семейств сопроцессор для переменных с плавающей точкой не предусмотрены.

### 2.2. AVR32

AVR32 — 32-битные микроконтроллеры архитектуры RISC, разработанные фирмой Atmel. Могут использоваться в КПК и других мобильных высокопроизводительных устройствах. Имеют 2 семейства: AVR32 AP и AVR32 UC3.

Несмотря на увеличенную разрядность системы микроконтроллеры архитектуры AVR32 используют схожий с AVR набор команд и не имеют аппаратную реализацию операций с плавающей точкой.

### 2.3. ESP-32

ESP32 — серия недорогих микроконтроллеров с низким энергопотреблением. Представляют собой систему на кристалле с интегрированным Wi-Fi и Bluetooth контроллерами и антеннами. Данные микроконтроллеры приобрели большую популярность за счёт наличия WiFi и Bluetooth, что позволило использовать их для создания IoT устройств.

В серии ESP32 используется микроконтроллерное ядро Tensilica Xtensa LX6 в вариантах с двумя и одним ядром. Серия является преемником микроконтроллеров ESP8266.

Архитектура набора команд Xtensa (ISA) — это новая ISA после RISC, ориентированная на встроенные, коммуникационные и потребительские товары.

Xtensa ISA позволяет конфигурировать отдельные функции ядра в соответствии с задачами, для которых будет использоваться контроллер. Рисунок 1 иллюстрирует общую организацию аппаратного обеспечения процессора, который реализован в Xtensa ISA.

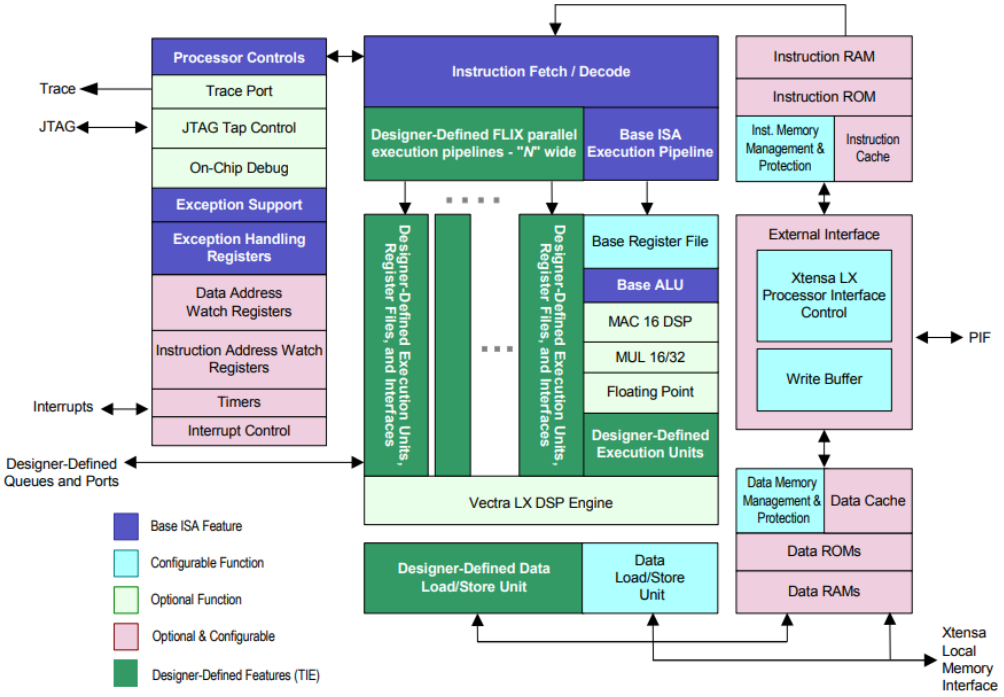


Рисунок 1. Структурная схема ядра Xtensa

Опция сопроцессора с плавающей запятой добавляет логические и архитектурные компоненты необходимые для реализации операций с плавающей запятой одинарной точности IEEE754. По умолчанию Xtensa ISA не включает в себя сопроцессор для обработки значений с плавающей точкой, но может быть добавлена заказчиком.

В микросхемах семейства ESP-8266 и ESP -32 сопроцессор для обработки значений с плавающей точкой не установлен.

## 2.4. ARM

Архитектура ARM разделена на 3 профиля:

- A (application) — для устройств, требующих высокой производительности (смартфоны, планшеты);
- R (real time) — для приложений, работающих в реальном времени;
- M (microcontroller) — для микроконтроллеров и недорогих встраиваемых устройств.

В данной работе рассматриваются только микроконтроллеры на ядрах Cortex архитектуры ARM M.

В зависимости от архитектуры в микроконтроллерах Cortex может присутствовать FPU одинарной или двойной точности [8, 9].

Таблица 1. Сравнительные характеристики архитектур Cortex

| | **Cortex M1** | **Cortex M3** | **Cortex M4** | **Cortex M7** | **Cortex M23** | **Cortex M33** | **Cortex M35P** | **Cortex M55** |
|---|---|---|---|---|---|---|---|---|
| ARM архитектура | ARMv6-M | ARMv7-M | ARMv7E-M | ARMv7E-M | ARMv8-M Baseline | ARMv8-M Mainline | ARMv8-M Mainline | Armv8.1-M |
| Архитектура | Фон Нейман | Гарвардская | Гарвардская | Гарвардская | Фон Нейман | Гарвардская | Гарвардская | Гарвардская |
| Операции с плавающей точкой одинарной точности (binary32) | Нет | Нет | Опционально | Опционально | Нет | Опционально | Опционально | Опционально |

| | Cortex M1 | Cortex M3 | Cortex M4 | Cortex M7 | Cortex M23 | Cortex M33 | Cortex M35P | Cortex M55 |
|---|---|---|---|---|---|---|---|---|
| Операции с плавающей точкой двойной точности (binary64) | Нет | Нет | Нет | Опционально | Нет | Нет | Нет | Опционально |

### 2.5. Сводная таблица

Итог рассмотрения вышеперечисленных архитектур представлен в таблице 2.

Таблица 2. Наличие FPU в рассматриваемых архитектурах

| Архитектура | Аппаратная реализация операций с плавающей точкой. (наличие FPU) |
|---|---|
| AVR | Отсутствует |
| AVR32 | Отсутствует |
| ESP-32 | Опционально в рамках архитектуры |
| ARM -M | Опционально в старших сериях |

Рассмотренные архитектуры микроконтроллеров позволяют сделать вывод о низком уровне внедрения FPU в архитектуры микроконтроллеров. Особенно эта проблема актуальна для архитектур малой разрядности, например 8-ми битной AVR.

## 3. Эмуляция FPU в микроконтроллерах AVR

В архитектурах без аппаратной реализации операций для типов данных с плавающей точкой их поддержка осуществляется за счёт эмуляции FPU имеющимися командами.

Исходный код на языке Си реализующий простое сложение двух переменных.

```
volatile float f1,f2,fr;
f1=0.5;
f2=0.25;
fr=f1+f2;
```

В связи с отсутствием аппаратной реализации сложения микроконтроллер использует имеющиеся аппаратные команды для эмуляции FPU и выполнения сложения [10, 11].

Ассемблерный код, сгенерированный компилятором GCC-AVR:

```
 fr=f1+f2;
 b2:    69 81       ldd   r22, Y+1    ; 0x01
 b4:    7a 81       ldd   r23, Y+2    ; 0x02
 b6:    8b 81       ldd   r24, Y+3    ; 0x03
 b8:    9c 81       ldd   r25, Y+4    ; 0x04
 ba:    2d 81       ldd   r18, Y+5    ; 0x05
 bc:    3e 81       ldd   r19, Y+6    ; 0x06
 be:    4f 81       ldd   r20, Y+7    ; 0x07
 c0:    58 85       ldd   r21, Y+8    ; 0x08
 c2:    10 d0       rcall .+32  ; 0xe4 <__addsf3>
 c4:    69 87       std   Y+9, r22    ; 0x09
 c6:    7a 87       std   Y+10, r23   ; 0x0a
 c8:    8b 87       std   Y+11, r24   ; 0x0b
 ca:    9c 87       std   Y+12, r25   ; 0x0c
000000e4 <__addsf3>:
 e4:    bb 27       eor   r27, r27
 e6:    aa 27       eor   r26, r26
 e8:    0e d0       rcall .+28  ; 0x106 <__addsf3x>
 ea:    77 c0       rjmp  .+238       ; 0x1da <__fp_round>
 ec:    68 d0       rcall .+208       ; 0x1be <__fp_pscA>
```

```
ee:    30 f0       brcs  .+12  ; 0xfc <__addsf3+0x18>
f0:    6d d0       rcall .+218       ; 0x1cc <__fp_pscB>
f2:    20 f0       brcs  .+8   ; 0xfc <__addsf3+0x18>
f4:    31 f4       brne  .+12  ; 0x102 <__addsf3+0x1e>
f6:    9f 3f       cpi   r25, 0xFF   ; 255
f8:    11 f4       brne  .+4   ; 0xfe <__addsf3+0x1a>
fa:    1e f4       brtc  .+6   ; 0x102 <__addsf3+0x1e>
fc:    5d c0       rjmp  .+186       ; 0x1b8 <__fp_nan>
fe:    0e f4       brtc  .+2   ; 0x102 <__addsf3+0x1e>
100:   e0 95       com   r30
102:   e7 fb       bst   r30, 7
104:   53 c0       rjmp  .+166       ; 0x1ac <__fp_inf>
```

Операция сложения представлена отдельной функцией __addsf3, которая в свою очередь также вызывает несколько функций в зависимости от результата выполнения промежуточных действий.

Данная реализация позволяет выполнить сложную математическую операцию, но требует большого количества операций.

## 4. Типы данных с фиксированной точкой

Типы данных с фиксированной точкой хранят число с фиксированным смещением, что позволяет им иметь простую организацию и использовать целочисленные операции вместо специальных. Наиболее актуальным стандартом чисел с фиксированной точкой является ISO/IEC DTR 18037 “Programming languages, their environments and system software interfaces —Extensions for the programming language C to support embedded processors” [12]. Данный стандарт описывает реализацию типов данных с фиксированной точкой Fract и Accum для языка C/C++. Ввиду специфики прикладных задач, решаемых данными языками, именно для них были реализованы данные типы данных в компиляторе AVR-GCC.

### 4.1. ISO/IEC DTR 18037

Технический отчет охватывает три отдельных раздела: арифметика с фиксированной точкой, именованные адресные пространства и классы хранения именованных регистров, а также базовая аппаратная адресация ввода-вывода. Поскольку это технический отчет, реализации могут свободно выбирать функции, которые они хотят поддерживать. Технический отчет действительно поощряет реализации, которые выбирают один из разделов для полной поддержки всего раздела.

Первое издание технического отчета было опубликовано в 2004 году как ISO / IEC TR 18037: 2004. Второе издание технического отчета было впоследствии опубликовано в 2008 году как ISO / IEC TR 18037: 2008.

В техническом отчете представлены значения данных с фиксированной точкой, либо как дробные значения данных, либо как целая и дробная части. Данные с фиксированной точкой обозначаются новыми ключевыми словами _Fract и _Accum. Спецификатор _Sat может использоваться для определения насыщающего типа с фиксированной точкой. Удобная версия этих ключевых слов определена в <stdfix.h>. Заголовок также включает набор новых функций поддержки арифметических операций с фиксированной точкой.

### 4.2. Реализация в GCC-AVR

avr-gcc 4.8 [13] и выше частично поддерживает арифметику с фиксированной запятой в соответствии с ISO / IEC TR 18037 и реализует несколько типов данных (таблица 3).

Таблица 3. Типы данных Accum и Fract

| Тип | размер | беззнаковый | знаковый | Примечание |
|---|---|---|---|---|
| **_Fract** | | | | |
| short | 1 | 0,8 | ± 0,7 | |
| - | 2 | 0,16 | ± 0,15 | |
| long | 4 | 0,32 | ± 0,31 | |
| long long | 8 | 0,64 | ± 0,63 | Расширение GCC |
| **_Accum** | | | | |
| short | 2 | 8,8 | ± 8,7 | |
| - | 4 | 16,16 | ± 16,15 | |
| long | 8 | 32,32 | ± 32,31 | |
| long long | 8 | 16,48 | ± 16,47 | Расширение GCC |

Недостатком данной реализации можно считать различную точность у всех типов данных, что усложняет перенос значений между ними.

Ключевым недостатком данной реализации является выделение знакового бита из дробной части, это привело к появлению двух существенных проблем.

#### 4.2.1. Различная точность у знаковых и беззнаковых типов данных.

В связи с выделением бита дробной части под знаковый бит точность знаковых типов данных всегда на один разряд меньше, это может привести к потере точности при присваивании беззнакового значения в знаковую переменную, а также усложняет процесс выполнения математических операций между знаковыми и беззнаковыми переменными.

#### 4.2.2. Несовместимость знаковых типов данных с целыми знаковыми типами данных.

Выделение знакового бита из дробной части позволило не изменять размер целой части, что привело к её увеличению на один бит и получения 9-ти битового целого знакового в short _Accum, 17-ти битного целого в _Accum и long long _Accum а также 33-х битного целого в long _Accum. Все эти типы данных превышают размеры классических знаковых целых типов данных размером 8, 16 и 32 бита что затрудняет работу с ними.

## 5. Тестирование

Для оценки скорости выполнения различных операций или алгоритмов используется симулятор IDE Microchip Studio. Данный симулятор позволяет полностью симулировать выполнение команд различными контроллерами архитектур AVR и AVR32 и измерить количество тактов необходимых для выполнения различных операций.

Измерение тактов позволяет получить наиболее точное значение времени выполнения, не зависящее от режимов работы микроконтроллеров и внешних факторов.

### 5.1. Выборка данных

Для сравнения времени выполнения отдельных математических операций ввиду особенностей типа данных float ключевым параметром, влияющим на скорость выполнения операций является разница между смещенными порядками аргументов, которая отображает количество разрядов.

Зависимость времени от разниц между разрядами представлена на рисунках 2 и 3.

На рисунке 2 время выполнения операции растёт с при увеличении разницы в размерности слагаемых. Уменьшение времени выполнения операции при достижении числа 64 обусловлено достижением разницы между разрядами значения 8, что позволяет использовать аппаратную реализацию операции сдвига на 8 порядков, что значительно ускоряет время выполнения операции.


Рисунок 2. Анализ операции сложения

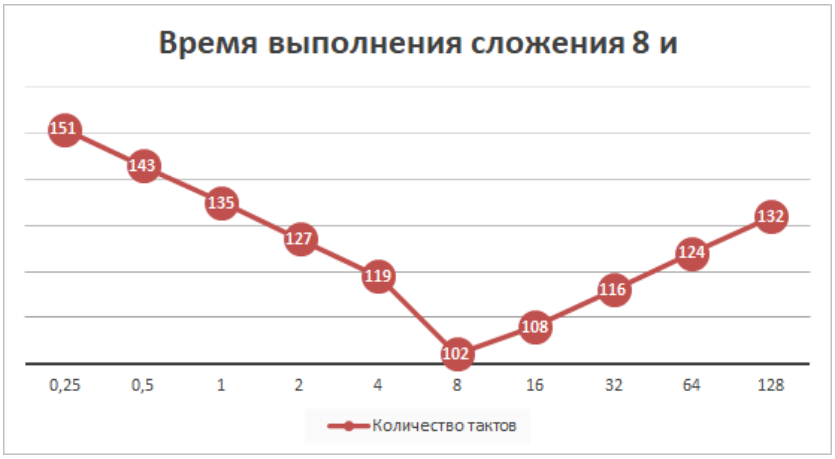


Рисунок 3. Анализ операции сложения

### 5.2. Базовые математические операции

Наибольший интерес для изучения производительности представляют простейшие математические операции. С помощью анализа базовых операций можно сделать выводы о целесообразности применения сложных алгоритмов исходя из количества базовых математических операций в них.

#### 5.2.1. Сложение

Операция сложения является самой простой математической операцией для аппаратной реализации и используется в большинстве сложных алгоритмов.

При сложении чисел с плавающей запятой результат определяется как сумма мантисс слагаемых с общим для слагаемых порядком. Для выравнивания порядков мантиссы сдвигаются вправо или влево на разницу между порядками слагаемых.

Время выполнения данной операции напрямую зависит от разности между порядками мантисс аргументов, так как операции бинарного сдвига в большинстве архитектур выполняются н 1 или 8 бит, что требует выполнения нескольких операций сдвига и организации циклов для корректной работы данной операции. Время выполнения операции сложения в тактах представлено в таблице 4.

Таблица 4. Время выполнения операции сложения для разных значений в формате binary32

| Add binary32 | 0,125 | 0,25 | 0,5 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0,125 | 102 | 108 | 127 | 124 | 132 | 151 | 159 | 167 | 124 | 133 | 130 |
| 0,25 | 119 | 102 | 108 | 127 | 135 | 143 | 151 | 159 | 167 | 124 | 133 |
| 0,5 | 127 | 108 | 102 | 119 | 127 | 135 | 143 | 151 | 159 | 167 | 124 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 135 | 116 | 108 | 102 | 119 | 127 | 135 | 143 | 151 | 159 | 167 |
| 2 | 143 | 124 | 116 | 108 | 102 | 119 | 127 | 135 | 143 | 151 | 159 |
| 4 | 151 | 132 | 124 | 116 | 108 | 102 | 119 | 127 | 135 | 143 | 151 |
| 8 | 159 | 140 | 132 | 124 | 116 | 108 | 102 | 119 | 127 | 135 | 143 |
| 16 | 167 | 148 | 140 | 132 | 124 | 116 | 108 | 102 | 119 | 127 | 135 |
| 32 | 124 | 156 | 148 | 140 | 132 | 124 | 116 | 108 | 102 | 119 | 127 |
| 64 | 133 | 113 | 156 | 148 | 140 | 132 | 124 | 116 | 108 | 102 | 119 |
| 128 | 141 | 122 | 113 | 156 | 148 | 140 | 132 | 124 | 116 | 108 | 102 |
| Среднее | 139,9 | 126,1 | 124,7 | 127,2 | 125,1 | 124,6 | 125,7 | 128,4 | 132,7 | 133,5 | 136 |
| Среднее общее | 129,7 | | | | | | | | | | |

В зависимости от разницы между разрядами мантиссы увеличение времени выполнения операции относительно минимального может достигать 64%, а в среднем достигает 27%, что делает сложно предсказуемым скорость выполнения данной операции при работе с различными наборами данных.

В отличии от типов данных с фиксированной точкой типы данных с фиксированной точкой обладают стабильным временем выполнения операций, для операции сложения двух 4-х байтовых переменных время выполнения составляет 27 тактов.

В среднем операция сложения для значения с плавающей точкой выполняется на 480% или на 103 такта медленнее относительно операции сложения для значения с фиксированной точкой.

### 5.2.2. Вычитание

Операция вычитания является развитием операции сложения, при которой отрицательное число переводится в дополнительный код и производится сложение.

Время выполнения данной операции напрямую зависит от разности между порядками мантисс аргументов, так как операции бинарного сдвига в большинстве архитектур выполняются н 1 или 8 бит, что требует выполнения нескольких операций сдвига и организации циклов для корректной работы данной операции. Время выполнения операции сложения в тактах представлено в таблице 5.

Таблица 5. Время выполнения операции вычитания для разных значений в формате binary32

| Sub binary32 | 0,125 | 0,25 | 0,5 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0,125 | 90 | 118 | 126 | 134 | 142 | 150 | 158 | 166 | 123 | 132 | 140 |
| 0,25 | 129 | 90 | 118 | 126 | 134 | 142 | 150 | 158 | 166 | 123 | 132 |
| 0,5 | 137 | 129 | 90 | 118 | 126 | 134 | 142 | 150 | 158 | 166 | 123 |
| 1 | 145 | 137 | 129 | 90 | 118 | 126 | 134 | 142 | 150 | 158 | 166 |
| 2 | 153 | 145 | 137 | 129 | 90 | 118 | 126 | 134 | 142 | 150 | 158 |
| 4 | 161 | 153 | 145 | 137 | 129 | 90 | 118 | 126 | 134 | 142 | 150 |
| 8 | 169 | 161 | 153 | 145 | 137 | 129 | 90 | 118 | 126 | 134 | 142 |
| 16 | 177 | 169 | 161 | 153 | 145 | 137 | 129 | 90 | 118 | 126 | 134 |
| 32 | 134 | 177 | 169 | 161 | 153 | 145 | 137 | 129 | 90 | 118 | 126 |

| Sub binary32 | 0,125 | 0,25 | 0,5 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | 143 | 134 | 177 | 169 | 161 | 153 | 145 | 137 | 129 | 90 | 118 |
| 128 | 151 | 143 | 134 | 177 | 169 | 161 | 153 | 145 | 137 | 129 | 90 |
| Среднее | 149,9 | 143,8 | 141,3 | 140,5 | 136,2 | 133,5 | 132,4 | 132,9 | 135 | 133,6 | 133,9 |
| Среднее общее | 137,3 | | | | | | | | | | |

В среднем операция вычитания для значения с плавающей точкой выполняется на 508% или на 110 тактов медленнее относительно операции вычитания для значения с фиксированной точкой.

### 5.2.3. Умножение

Результат умножения чисел с плавающей запятой определяется как произведение мантисс сомножителей чисел и сумма порядков чисел. При умножении разрядность сомножителей не выравнивается, анализ знакового разряда не производится.

Данный алгоритм обладает достаточно простой реализацией и время его выполнения не зависит от значения сомножителей и составляет 155 тактов.

При реализации умножения для чисел с фиксированной точкой алгоритм умножения обладает несколько большей сложностью чем для реализации сложения, так как операция умножения для операндов размеров больше 8 бит не реализована, и в отличие от сложения требует более сложного алгоритма.

Однако несмотря на это операция умножения занимает 93 такта, что на 67% быстрее чем операция умножения для чисел с плавающей точкой.

Дополнительным преимуществом операций с фиксированной точкой является возможность программной оптимизации при умножении на константу равную степени двойки за счёт замены операции умножения на бинарный сдвиг.

Это позволяет серьезно сократить время выполнения операций при умножении по сравнению с любым из вариантов реализации выполнения умножения (таблица 6).

Таблица 6. Время выполнения операции умножения для разных значений в формате fixed24_8 с помощью бинарного сдвига относительно классической реализации умножения

| Mul const/non const | 0,125 | 0,25 | 0,5 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| fixed24_8 | 40% | 25% | 20% | 16% | 20% | 25% | 40% | 47% | 55% | 62% | 70% |
| binary32 | 24% | 15% | 12% | 10% | 12% | 15% | 24% | 28% | 33% | 37% | 42% |

### 5.2.4. Деление

При делении чисел, представленных в форме с плавающей запятой, результат определяется в два этапа: порядок - как разность порядков делимого и делителя, деление мантисс чисел производится по тем же правилам, что и деление чисел с фиксированной запятой. Время выполнения операции для типа данных binary32 составляет 458 тактов.

При реализации деления для чисел с фиксированной точкой из-за отсутствия аппаратной реализации операции деления происходит деление с помощью вычитания, что приводит к серьезному увеличению времени выполнения операции относительно остальных базовых операций. Время выполнения деления вне зависимости от значения аргументов составляет 615 тактов.

За счёт особенностей представления данных в форматах IEEE-754 реализация операции деления, несмотря на существенно больший размер кода, 368 байт против 69 байт для фиксированной точки, реализующий операцию деления, занимает меньшее количество времени по сравнению с делением целых чисел.

Однако операция деления на константу, так же, как и операция умножения, может быть оптимизирована заменой на операцию бинарного сдвига, что серьезно повысит производительность отдельных операций относительно классических реализаций (таблица 7).

Таблица 7. Время выполнения операции деления для разных значений в формате fixed24_8 с помощью бинарного сдвига относительно классической реализации умножения

| Div const/non const | 0,125 | 0,25 | 0,5 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| fixed24_8 | 6% | 4% | 3% | 2% | 3% | 4% | 6% | 7% | 8% | 9% | 11% |
| binary32 | 8% | 5% | 4% | 3% | 4% | 5% | 8% | 10% | 11% | 13% | 14% |

Оптимизация за счёт замены бинарным сдвигом позволяет получить серьезный прирост производительности и обеспечить экономию памяти микроконтроллера по сравнению с любым из вариантов реализации деления.

### 5.3. Сравнение с типом данных с фиксированной точкой

Исходя из полученных результатов можно построить результирующие графики в абсолютных (рисунок 4) и относительных (рисунок 5) величинах.

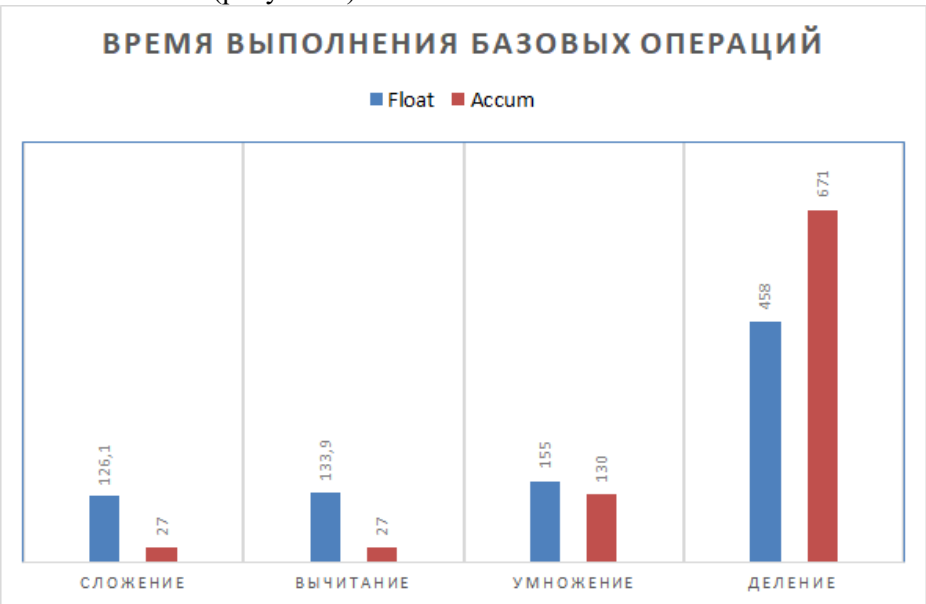


Рисунок 4. Время выполнения базовых математических операций

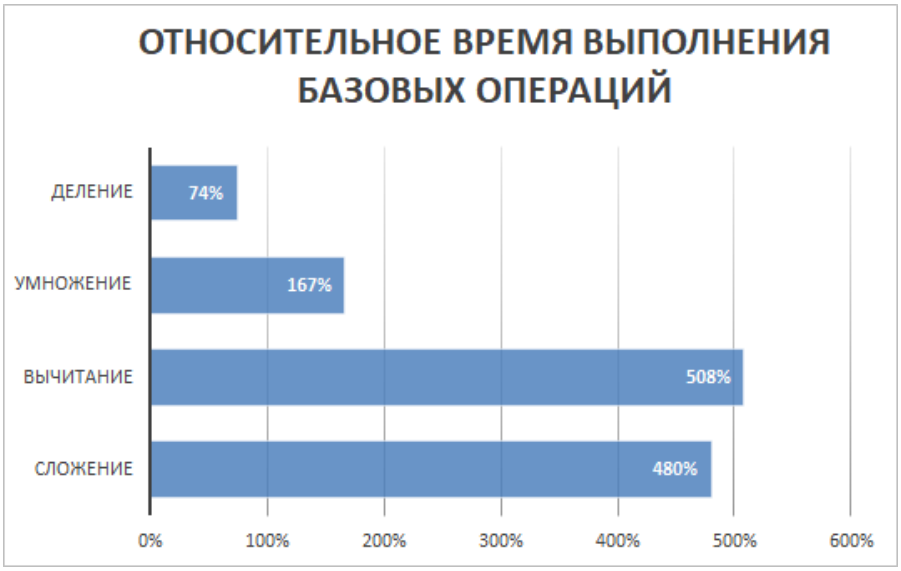


Рисунок 5. Время выполнения базовых математических операций в формате binary32 относительно fixed24_8

## 6. Выводы

Математические операции сложения и вычитания для типа binary32 при эмуляции FPU выполняется на 380% и 408% медленнее относительно fixed24_8 за счёт необходимости реализовывать сложные математические алгоритмы.

Математическая операция умножения для типа binary32 при эмуляции FPU выполняется на 20% медленнее относительно fixed24_8. Столь малое различие, относительно сложения и вычитания, связано с отсутствием аппаратной реализации операции умножения в микроконтроллерах AVR, данная проблема актуальна для многих архитектур, например для Cortex M1 не реализована операция умножения для 32-битных операндов, а для Cortex M23 реализовано только умножение для 32-х битных операндов с 32-х битным результатом.

Математическая операция деления для типа binary32 при эмуляции FPU выполняется на 26% быстрее относительно Fixed. Это связано с отсутствием аппаратной реализации операции деления в архитектуре AVR, что приводит к необходимости программно реализовывать деление для чисел с фиксированной точкой и возможности получить большую производительность деления при использовании типов данных согласно IEEE-754 за счёт иного алгоритма деления.

Использование типов данных с фиксированной точкой, вместо типов данных с плавающей точкой, при написании программ для микроконтроллеров без FPU, позволяет получить существенный прирост в производительности ПО.

## СПИСОК ЛИТЕРАТУРЫ


1. Bruce Dawson «Comparing Floating Point Numbers» [Электронный ресурс]. – URL: https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/
2. Р.А. Ющенко, А.К. Ющенко Особенности арифметики с плавающей запятой в современных компьютерах// Компьютерная математика. 2016, № 1.
3. *Молчанов И.Н.* Машинная математика. Проблемы и перспективы // Кибернетика и системный анализ. – 2004. – № 6. – С. 65 – 72.
4. Goldberg D. What Every Computer Scientist Should Know About Floating-Point Arithmetic //ACM Computing Surveys. – 1991. – Vol. 1, N 23. – P. 5 – 48.
5. Bjorndalen J.M., Anshus O.J. Trusting floating point benchmarks - are your benchmarks really data independent?, Proceedings of the 8th international conference on Applied parallel computing: state of the art in scientific computing, June 18 – 21, 2006, Umea, Sweden.
6. Higham N. Accuracy and Stability of Numerical Algorithms. – Philadelphia: 2002, SIAM. –P. 680.
7. *IEEE* 754-2008 Standard for Floating-Point Arithmetic.
8. Джозеф Ю. Ядро CORTEX-M3 компании ARM. Полное руководство. Пер. с англ. А.В. Евстифеева. - М.: Додэка-XXI, 2012. - 552 с.
9. Мартин Т. Микроконтроллеры ARM7 семейств LPC2300/2400. Вводный курс разработчика М.: Додэка-XXI, 2010. - 336 с.
10. Астраханцев А.В. Система команд микроконтроллеров семейства AVR СевНТУ, 2003. – 56 с.
11. Ревич Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера 2011 г. — 351 с.
12. ISO/IEC DTR 18037 Programming languages, their environments and system software interfaces — Extensions for the programming language C to support embedded processors.
13. Документация компилятора avr-gcc [Электронный ресурс]. – URL: https://gcc.gnu.org/wiki/avr-gcc#Fixed-Point_Support