Компьютерные инструменты в образовании, 2022
№ 1: 46–56
УДК: 004.31:004.051
http://cte.eltech.ru
doi:10.32603/2071-2340-2022-1-46-56

ИССЛЕДОВАНИЕ ОПЕРАЦИИ СОВМЕЩЁННОГО УМНОЖЕНИЯ-СЛОЖЕНИЯ НА ПРОЦЕССОРЕ Baikal-T

Архипов И. С.[1], студент, ✉ arkhipov.iv99@mail.ru

[1]Санкт-Петербургский государственный университет,
Университетский пр., д. 28, Старый Петергоф, 198504, Санкт-Петербург, Россия

Аннотация

Данная статья посвящена исследованию эффективности команды совмещённого умножения–сложения на процессоре Baikal-T. Рассмотрены различные примеры использования команды, сделаны измерения и сформулированы выводы, в каких случаях применение совмещённого умножения-сложения даёт выигрыш в вычислениях и в каких ситуациях применение команды невыгодно с точки зрения скорости выполнения программы.

Ключевые слова: *MIPS, процессор Байкал, умножение-сложение, оптимизация, ассемблер.*

Цитирование: Архипов И. С. Исследование операции совмещённого умножения-сложения на процессоре Baikal-T // Компьютерные инструменты в образовании. 2022. № 1. С. 46–56. doi: 10.32603/2071-2340-2022-1-46-56

1. ВВЕДЕНИЕ

На протяжении десятилетий высокопроизводительные вычисления были и остаются неотъемлимой частью академических исследований и технологических инноваций в различных отраслях. Инженеры, учёные и исследователи используют высокопроизводительные системы для решения ресурсоёмких задач. Такие системы используются в самых разных отраслях: экономика, генетика, здравоохранение, нефтегазовая отрасль, промышленность и многие другие.

В рамках развития вычислений с высокой производительностью было разработано множество технологий, например SIMD, MIMD, многоядерные, многопроцессорные, параллельные и распределённые вычисления. Увеличение скорости вычислений ведётся и на программном уровне. Во многом развитие технологии компиляции обязано потребностью в высокой скорости работы программ. Также и на аппаратном уровне реализуются специальные команды для ускорения выполнения программы.

Одной из таких таких команд является совмещённое умножение-сложение (*multiply-accumulate* или *multiply-add*). Это распространённая операция, при которой два числа умножаются и складываются с аккумулятором. Совмещённое умножение-сложение может ускорить вычисление многих широко применяемых вычислений. К таким относятся скалярное произведение, умножение матриц, вычисление значения полинома, метод Ньютона и многие другие.

Операция включена в стандарт IEEE 754-2008 [1, с. 21]. Данная инструкция реализована, например, в выскопроизводительной системе NVIDIA GPU серий GeForce 200 (GTX 200), GeForce 300 и NVIDIA Tesla GPGPU C1060 & C2050 / C2070 [2, с. 8]. Реализовано совмещённое умножение-сложение и в архитектуре MIPS [3, с. 255], на базе которой реализован российский процессор Baikal-T[1].

При работе с отладочной платой BFK 3.1 было обнаружено, что использование команды совмещённого умножения-сложения в некоторых ситуациях не только не даёт выигрыша во времени исполнения программы, но и сильно замедляет вычисления. Например, оказалось, что при умножении матриц следует отказаться от использования данной инструкции. Из-за этого оптимизирующие компиляторы генерируют не самый оптимальный код программы. Исследованию эффективности операции совмещённого умножения-сложения и посвящена данная работа.

2. ОБЗОР

Ввиду новизны процессора Baikal-T на данный момент отсутствуют работы с исследованием его особенностей. Имеются несколько обзорных статей и патентов [7–9]. Остальные работы посвящены специфическим применениям и разработкам на процессоре Baikal-T. Например, программа конфигурации операционной системы для одноплатной рабочей станции на базе процессора Baikal-T [10], встроенное программное обеспечение загрузчика операционной системы для одноплатной рабочей станции на базе процессора Baikal-T [11] и другие свидетельства о регистрации программы.

3. ПРИМЕР: УМНОЖЕНИЕ МАТРИЦ

Рассмотрим довольно распространённую программу умножения матриц из Приложения 1. При работе этой программы бóльшая часть времени работы машины уходит на цикл в строках 28–31. Машинные команды данного цикла будут исполняться 8 миллиардов раз. Посмотрим, как транслировали в машинный код MIPS этот цикл различные трансляторы.

Algorithm 1: Внутренний цикл после компиляции Clang

```
$BB0_12:
    addu $1, $9, $24      # вычисление адреса a[i][k]
    lw $25, 0($14)        # вырезка b[k][j]
    addiu $14, $14, 800   # инкремент индуцированной переменной
    addiu $24, $24, 4     # инкремент индуцированной переменной
    lw $1, 0($1)          # вырезка a[i][k]
    mul $1, $25, $1       # умножение
    bne $24, $6, $BB0_12  # переход
    addu $15, $15, $1     # сложение
```

Код цикла в строках 28–31, полученный в результате трансляции Clang [4] версии 6.0.0, представлен в Algorithm 1, а код, полученный после трансляции GGC [5] версии 7.5.0, — в Algorithm 2.

[1] Название процессора постоянно изменяется. Когда велось исследование, он назывался Baikal-T1. Сейчас в ходу двойное название: Baikal-T или BE-T1000. В данной статье было принято решение использовать наименование Baikal-T.

Algorithm 2: Внутренний цикл после компиляции GCC

```
$L8:
    addiu $2,$2,800  # инкремент индуцированной переменной
    lw $5,0($3)      # вырезка a[i][k]
    lw $4,-800($2)   # вырезка b[k][j]
    addiu $3,$3,4    # инкремент индуцированной переменной
    bne $6,$2,$L8    # переход
    madd $5,$4       # умножение и сложение
```

В коде, полученном в результате трансляции GGC, меньше машинных команд, поэтому логично предположить, что он исполняется быстрее кода, полученного после трансляции Clang. Однако измерения показывают, что это не так. Программы были запущены на плате BFK 3.1 с процессором Baikal-T с измерением времени с помощью утилиты *time* [6]. Результат представлен в таблице 1.

Таблица 1. Сравнение Clang и GCC

Транслятор	**Время, с**
Clang	61.55s
GCC	72.15s

Почему же это происходит, ведь варианты транслированного кода мало отличаются друг от друга? Возникает предположение, что такое возможно из-за использования команды *madd*. Это команда совмещённого умножения-сложения, её использование позволяет выиграть в количестве затраченных машинных команд при трансляции. Чтобы проверить данное предположение, был переписан код, полученный в результате трансляции Clang, с использованием команды *madd* (Algorithm 3), и замерено время. Результат представлен в таблице 2. Чтобы убедиться в результате, была переписана программа, полученная после трансляции GCC, без *madd* (Algorithm 4) и замерено время. Результат представлен в таблице 3.

Таблица 2. Время исполнения после трансляции Clang с *madd* и без использования *madd*

	Время, с
С *madd*	74.46
Без *madd*	61.55

Algorithm 3: Код после трансляции Clang с использованием *madd*

```
$BB0_12:
    addu $1, $9, $24      # вычисление адреса a[i][k]
    lw $1, 0($1)          # вырезка a[i][k]
    lw $25, 0($14)        # вырезка b[k][j]
    madd $1, $25          # умножение и сложение
    addiu $24, $24, 4     # инкремент индуцированной переменной
    bne $24, $6, $BB0_12  # переход
    addiu $14, $14, 800   # инкремент индуцированной переменной
```

Далее необходимо выяснить, работает ли программа с *madd* медленнее из-за какой-либо последовательности команд, включающей в себя *madd*, которая замедляет работу,

Таблица 3. Время исполнения после трансляции GCC с *madd* и без использования *madd*

	Время, с
С *madd*	72.15
Без *madd*	54.57

Algorithm 4: Код после трансляции GCC без использования *madd*

```
$L8:
    addiu $2,$2,800  # инкремент индуцированной переменной
    lw $5,0($3)      # вырезка a[i][k]
    lw $4,-800($2)   # вырезка b[k][j]
    addiu $3,$3,4    # инкремент индуцированной переменной
    mul $4, $4, $5   # умножение
    bne $6,$2,$L8    # переход
    add $16, $16, $4 # сложение
```

или дело всё-таки в самой команде. Для этого последовательно убирались команды из четырёх программ: Algorithm 1, Algorithm 2, Algorithm 3 и Algorithm 4.

Первый замер был произведён с неизменёнными программами. Во втором замере из Algorithm 1 была убрана вторая команда, из Algorithm 2 вторая команда, из Algorithm 3 третья команда и из Algorithm 4 вторая команда. В третьем замере дополнительно были убраны из Algorithm 1 первая и пятая команды, из Algorithm 2 третья команда, из Algorithm 3 первая и вторая команды и из Algorithm 4 третья команда. В последнем, четвёртом замере, были убраны из Algorithm 1 третья команда, из Algorithm 2 четвёртая команда, из Algorithm 3 седьмая команда, а из Algorithm 4 четвёртая команда. Результаты представлены на графике на рисунке 1.

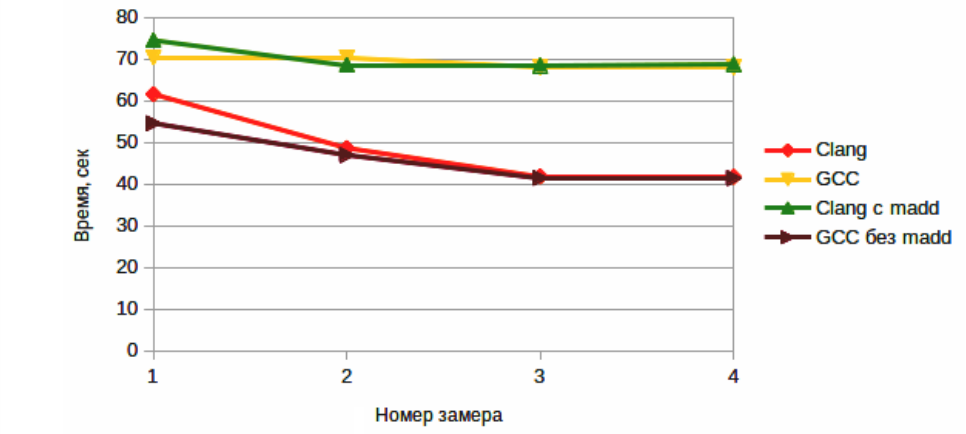

Рис. 1. Время работы программ

Команды из программ убирались так, чтобы были случаи и когда *madd* стоит в слоте задержки команды перехода, и когда *madd* стоит в окружении других арифметических команд, и когда слот задержки вообще не заполнен. Однако всегда программы с *madd* работали медленнее своих аналогов без *madd*. На основании этого можно сделать вывод, что медленная работа программы умножения матриц — не результат какого-то специфического сочетания команд, дело в самой команде *madd*.

4. ИЗУЧЕНИЕ РАБОТЫ КОМАНДЫ *madd*

4.1. Измерение количества тактов

Для изучения свойств команды *madd* нужен более тонкий инструмент, чем измерение времени. В дальнейшем будут измеряться такты процессора, которые тратятся на исполнения команд. Для подобных целей в архитектуре MIPS предусмотрена специальная команда *rdhwr* [3], которая позволяет получить значение счётчика тактов процессора. Была написана программа на ассемблере (фрагмент представлен в Algorithm 5), которая в цикле получает значение счётчика тактов процессора, исполняет последовательность команд, ещё раз получает значение счётчика тактов процессора и выводит в консоль количество тактов, необходимых для выполнения последовательности команд. Цикл исполняется 100 раз, чтобы получить среднее значение. Результат вывода на консоль записывается в файл, а потом обрабатывается программой на Python, которая выводит среднее значение тактов, необходимых для выполнения последовательности команд.

Algorithm 5: Подсчёт числа тактов

```
$L3:
    rdhwr $2, $2   # сохранение счётчика в начале
    # команды, для которых необходимо измерить такты
    ...
    rdhwr $3, $2   # сохранение счётчика второй раз
    subu $2,$3,$2  # вычитание
    # вывод результат в консоль
    ...
    # переход на метку $L3 в цикле
    ...
```

4.2. Эффективность изолированной команды madd

Сначала необходимо проверить, не является ли команда *madd* «плохой» с точки зрения времени исполнения сама по себе или это проявляется только в паре с командой условного перехода. Для этого было проведено измерение количества тактов, необходимых для выполнения от 1 до 20 команд *madd* подряд, и сделано сравнение с количеством тактов, необходимых для выполнения от 1 до 20 пар команд *mul* + *addu*. Результат представлен на графике на рисунке 2.

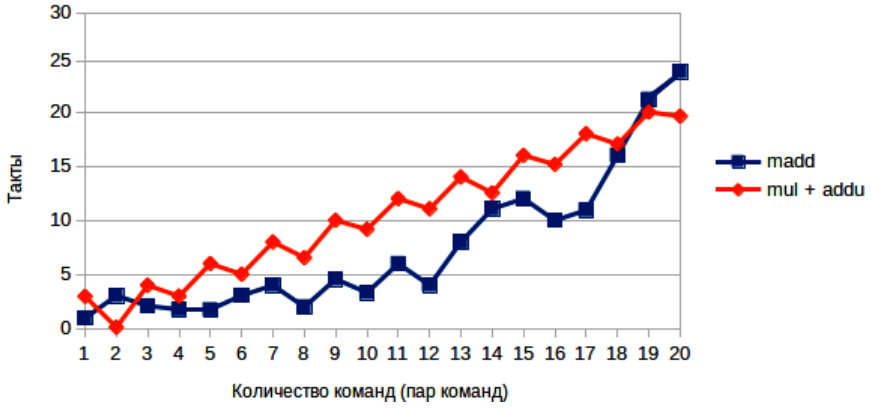

Рис. 2. Количество тактов

Из графика видно, что в большинстве случаев команда *madd* работает лучше связки команд *mul* + *addu*. Также стоит отметить, что команда *madd* не нарушает работу конвейера, так как количество тактов изменяется каждый раз на разную величину.

4.3. Эффективность команды madd в последовательных переходах

Далее необходимо изучить работу команды *madd* совместно с командой условного перехода. Рассмотрим программы Algorithm 6 и Algorithm 7. Algorithm 6 представляет из себя набор блоков, состоящих из команды условного перехода и *madd* в слоте задержки и выполняющихся последовательно. Аналогичным образом построена и программа Algorithm 7, только в слоте задержки находится *addu*, а перед условным переходом находится команда *mul*. Блоков в программе от 1 до 10. Результат замера количества тактов представлен на рисунке 3.

Algorithm 6: *madd* + *bne*

```
    bne $12, $13, $A1
    madd $10, $11
$A1:
    bne $12, $13, $A2
    madd $10, $11
$A2:
    ...
```

Algorithm 7: *mul* + *bne* + *addu*

```
    mul $11, $10, $11
    bne $12, $13, $A1
    addu $14, $11, $14
$A1:
    mul $11, $10, $11
    bne $12, $13, $A2
   addu $14, $11, $14
$A2:
    ...
```

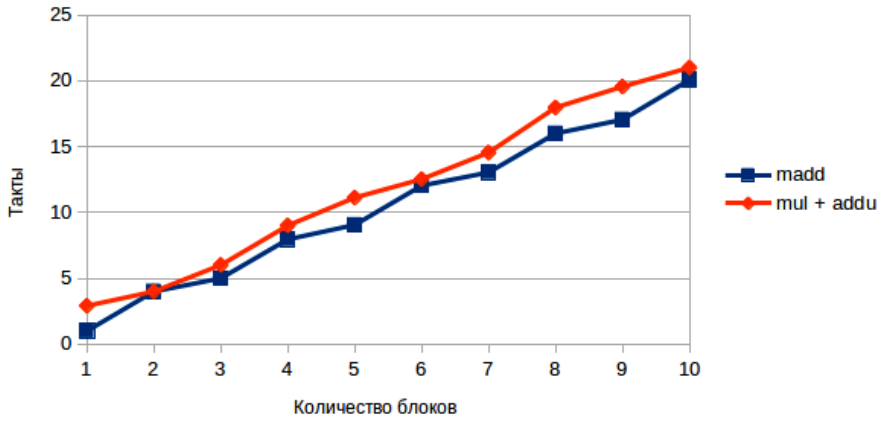

Рис. 3. Количество тактов на блоки

Из графика следует, что использование *madd* выгоднее, чем использование *mul* и *addu*. Получается, что и сама по себе, и в связке с командой условного перехода *madd* работает быстрее, чем в аналогичных условиях *mul* + *addu*.

4.4. Эффективность команды madd в цикле

Совершенно иная картина предстаёт при использовании *madd* в цикле. Сначала было рассмотрено небольшое количество итераций цикла, от 1 до 20. Код циклов с *madd* и без неё представлен в Algorithm 8 и Algorithm 9 соответственно. Результат замера тактов представлен на рисунке 4. Ситуация кардинально отличается от предыдущих. Вариант с *madd* работает сильно хуже аналогичного ему варианта без *madd*.

Algorithm 8: Цикл *madd* + *bne*

```
$A1:
    addiu $12, $12, 1
    bne $12, $13, $A1
   madd $10, $11
```

Algorithm 9: Цикл *mul* + *bne* + *addu*

```
$A1:
    addiu $12, $12, 1
    mul $11, $10, $11
    bne $12, $13, $A1
    addu $14, $11, $14
```

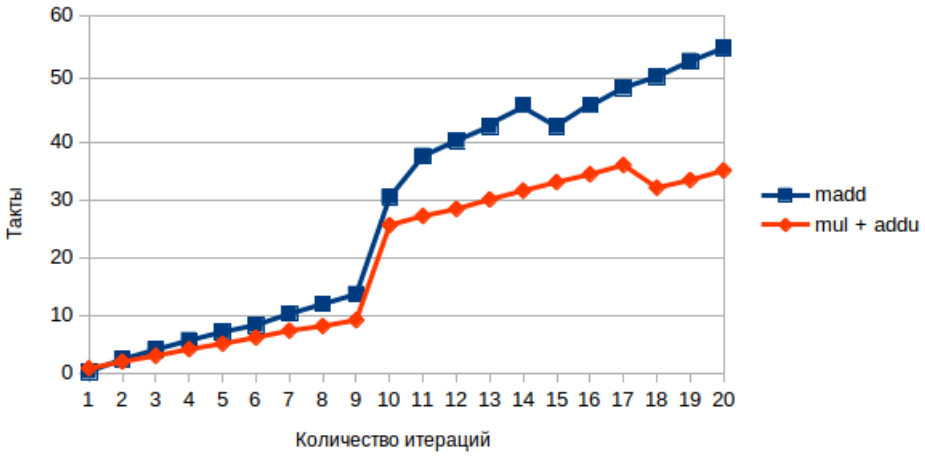

Рис. 4. Время работы программ

Далее были сделаны замеры для развёрнутого цикла с количеством итераций от 1 до 20. Код программ с использованием инструкции *madd* и без неё представлены в Algorithm 10 и Algorithm 11 соотвественно. Результат представлен на рисунке 5. Ситуация со временем работы программ изменилась на противоположную.

Algorithm 10: *madd + bne*

```
    addiu $12, $12, 1
    bne $12, $13, $A1
    madd $10, $11
$A1:
    addiu $12, $12, 1
    bne $12, $13, $A2
    madd $10, $11
$A2:
    ...
```

Algorithm 11: *mul + bne + addu*

```
    addiu $12, $12, 1
    mul $11, $10, $11
    bne $12, $13, $A1
    addu $14, $11, $14
$A1:
    addiu $12, $12, 1
    mul $11, $10, $11
    bne $12, $13, $A1
    addu $14, $11, $14
$A2:
    ...
```

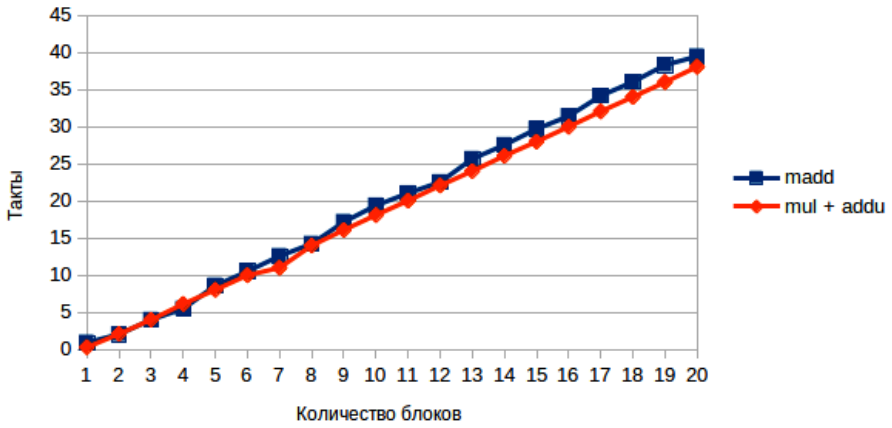

Рис. 5. Время работы программ

В конце были сделаны измерения количества тактов, необходимых для выполнения циклов с бóльшим количеством итерациий. Код циклов с *madd* и без неё представлен в Algorithm 8 и Algorithm 9 соответственно. Замеры были сделаны каждые 20 итераций с количеством итераций от 20 до 200. Результат представлен на рисунке 6.

5. ЗАКЛЮЧЕНИЕ

В статье были исследованы особенности работы инструкции совмещённого умножения–сложения *madd* архитектуры MIPS в процессоре Baikal-T. Был рассмот-

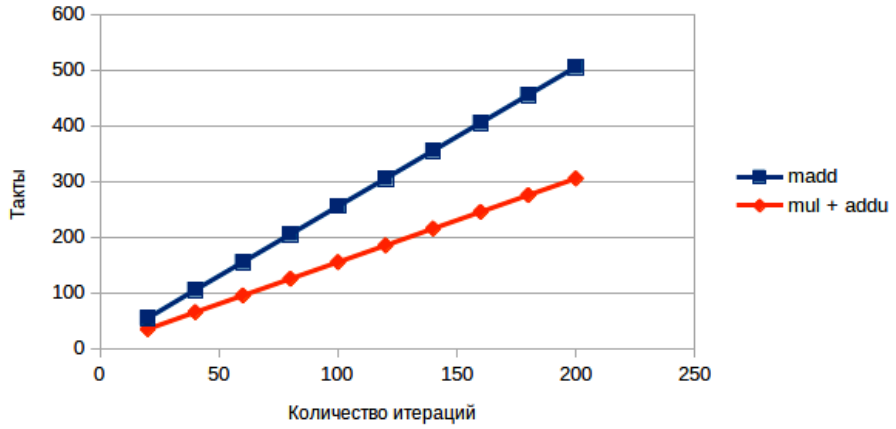

Рис. 6. Время работы программ

рен пример умножения матриц, а также изучены различные случаи использования команды *madd*. Результаты исследования показали, что *madd* крайне неэффективна в циклах, однако в остальных случаях её использование оправдано.

Приложение 1

```
void main() {
  int a[200][200], b[200][200], c[200][200];
  register int i, j, k, v;

  for (i = 0; i < 200; ++i)
  {
     for (j = 0; j < 200; ++j)
     {
        a[i][j] = i * j;
     }
  }

  for (i = 0; i < 200; ++i)
  {
     for (j = 0; j < 200; ++j) {

        b[i][j] = i + j;
     }
  }

  for (v = 0; v < 2000; ++v)
  {
     for(i = 0; i < 200; ++i)
     {
        for(j = 0; j < 200; ++j)
        {
           register int cij = 0;
           for(k = 0; k < 200; ++k)
           {
              cij += a[i][k] * b[k][j];
           }
           c[i][j] = cij;
        }
     }
  }
  printf("%i\n", c[0][0]);
}
```

Список литературы

1. IEEE Standard for Floating-Point Arithmetic // IEEE Std 754-2019 (Revision of IEEE 754-2008). 22.06.2019. 2019. P. 1–84, doi: 10.1109/IEEESTD.2019.8766229
2. Whitepaper NVIDIA's Next Generation CUDATM Compute Architecture: Fermi. 2009. https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf (дата обращения: 24.03.2022).
3. MIPS Architecture for Programmers. Volume II-A: The MIPS32. Instruction Set Manual. Document Number: MD00086, Revision 6.06. 2016. https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf (дата обращения: 24.03.2022).
4. *Ставцев Р.* Процессор Байкал-Т1. Программное и аппаратное окру- жение // Пятнадцатая конференция разработчиков свободных программ, 2018. С. 86–88.
5. *Ивлев А. А., Осипенко А. С., Лебедев М. А., Терентьев Ю. И., Хренов Г. Ю., Брюхова Ю. В., Зенин Е. Ю.* Процессор Байкал-Т1. Номер патента: RU 2016630090, 2016.
6. *Егоров А. А.* «Baikal electronics» Высокопроизводительные энерго-эффективные процессоры (обзор) // Автоматизация и IT в нефтегазовой области. 2021. № 4 (46). С. 48–59.
7. *Байков А. С. Ивановская Е. В.* Программа конфигурации операционной системы для одноплатной рабочей станции на базе процессора «Байкал-Т1». Свидетельство о регистрации программы для ЭВМ RU 2019611270, 2019.
8. *Новиков И. М.* Программа конфигурации операционной системы для одноплатной рабочей станции на базе процессора «Байкал-Т1». Свидетельство о регистрации программы для ЭВМ RU 2018665110, 2018.
9. Официальный сайт LLVM. URL: https://llvm.org/ (дата обращения: 24.03.2022).
10. Официальный сайт GCC. URL: https://gcc.gnu.org/ (дата обращения: 24.03.2022).
11. time(1) Linux manual page. URL: https://man7.org/linux/man-pages/man1/time.1.html (дата обращения: 24.03.2022).

Поступила в редакцию 08.03.2022, окончательный вариант — 24.03.2022.

Архипов Иван Сергеевич, студент образовательной программы магистратуры «Математическое обеспечение и администрирование информационных систем» СПБГУ 1 года обучения, ✉ arkhipov.iv99@mail.ru

Computer tools in education, 2022
№ 1: 46–56
http://cte.eltech.ru
doi:10.32603/2071-2340-2022-1-46-56

Investigation of the Multiply-add Operation on the Baikal-T Processor

Arkhipov I. S.[1], Graduate Student, ✉ arkhipov.iv99@mail.ru

[1]Saint Petersburg State University, 28, Universitetskiy pr., 198504, Saint Petersburg, Russia

Abstract

This article is devoted to the study of the efficiency of the multiply-add operation instruction on the Baikal-T processor. Various examples of using the command are considered, measurements are made and conclusions are formulated in which cases the use of multiply-

add operation gives a gain in calculations and in which situations the use of the command is unprofitable in terms of program execution speed.

Keywords: *MIPS, Baikal processor, multiply–addition, optimization, assembler.*

Citation: I. S. Arkhipov, “Investigation of the Multiply-add Operation on the Baikal-T Processor,” *Computer tools in education*, no. 1, pp. 46–56, 2022 (in Russian); doi: 10.32603/2071-2340-2022-1-46-56

References

1. IEEE “IEEE Standard for Floating-Point Arithmetic,” in *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 22 Jul. 2019; doi: 10.1109/IEEESTD.2019.8766229
2. Nvidia Corporation, “Whitepaper NVIDIA’s Next Generation CUDATM Compute Architecture: Fermi,” in *www.nvidia.com*, [Online]. Available: https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
3. Wave Computing, Inc., “MIPS Architecture for Programmers. Volume II-A: The MIPS32. Instruction Set Manual. Document Number: MD00086, Revision 6.06,” in *www.wavecomp.ai*, 2016. [Online]. Available: https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf
4. R. Stavtsev, “Protsessor Baikal-T1. Programmnoe i apparatnoe okruzhenie,” in Proc. *XV konferentsiya razrabotchikov svobodnykh programm*, 2018, pp. 86–88 (in Russian).
5. A. A. Ivlev, A. S. Osipenko, M. A. Lebedev, et al., *Protsessor Baikal-T1*, Patent: RU 2016630090, 07 Jun. 2016, 2016 (in Russian).
6. A. A. Egorov, “Baikal electronics” high performance energy efficient processors (Review),” *Avtomstiatizatsiya i IT v neftegazovoi oblasti*, no. 4 (46), pp. 48–59, 2021 (in Russian).
7. A. S. Baikov and E. V. Ivanovskaya, “Programma Konfiguratsii Operatsionnoi Sistemy Dlya Odnoplatnoi Rabochei Stantsii Na Baze Protsessora "Baikal-T1” *Certificate of registration of the computer program RU 2019611270, 29 Dec. 2018*, 2019 (in Russian).
8. I. M. Novikov “Vstroennoe programmnoe obespechenie zagruzchika operatsionnoi sistemy dlya odnoplatnoi rabochei stantsii na baze protsessora "Baikal-T1” *Certificate of registration of the computer program RU 2018665110, 02 Nov. 2018*, 2018 (in Russian).
9. V. Adve, Ch. Lattner, at al., “The LLVM Compiler Infrastructure LLVM,” in *https://llvm.org/*. [Online]. Available: https://llvm.org/
10. GCC Development, “GCC, the GNU Compiler Collection,” in *gcc.gnu.org*. [Online]. Available: https://llvm.org/
11. M. Kerrisk “time(1) — Linux manual page,” in *man7.org*, [Online]. Available: https://man7.org/linux/man-pages/man1/time.1.html

Received 08-03-2022, the final version — 24-03-2022.

Ivan Arkhipov, Graduate Student of the educational program “Mathematical support and administration of information systems” SPbSU 1 year of study, ✉ arkhipov.iv99@mail.ru