Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б09-мм

Создание симулятора процессора

ВАСИЛЬЕВА Екатерина Евгеньевна

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н. Луцив Д. В.

Санкт-Петербург
2024

Оглавление

Введение

Процессор важнейшая часть компьютеров и других электронных устройств. Существует множество различных архитектур процессоров. CISC, появившаяся раньше остальных, имеет большое количество сложных команд. Позже, набор команд в архитектурах стали упрощать и минимизировать с целью увеличения производительности и уменьшения стоимости процессоров. Так появилась RISC архитектура, которая сейчас используется наравне с CISC.

При этом количество команд в RISC процессорах всё ещё достаточно большое, например, только в базовом наборе инструкций RISC-V 40 команд. Очевидно, что возможно обойтись меньшим количеством. Существуют архитектуры содержащие единственную команду, например, в URISC [3] это переход, если результат вычитания отрицательный. URISC создана преимущественно для учебных целей, так как писать программы используя одну инструкцию слишком сложно, ведь представление даже самых простых операций не является очевидным.

Хотелось бы иметь архитектуру, число команд в которой будет небольшим, но при этом достаточным для того чтобы в ней было удобно писать программы. Вопрос создания такой архитектуры раскрывается в данной работе.

1. Постановка задачи

Целью работы является создание симулятора процессора. Для её выполнения были поставлены следующие задачи:

1. Спроектировать архитектуру набора команд(раздел 3.1).
2. Реализовать симулятор для этой архитектуры(раздел 3.2).
3. Протестировать симулятор на простых примерах (раздел 3.3).

2. Обзор

2.1. Существующие подходы к проектированию процессоров

- CISC

  Архитектура процессора, в которой длина инструкции может быть разной. Для более распространённых команд используются короткие, простые кода, а более специализированных — длинные, сложные. Одна инструкция является комплексной, может выполняться несколько циклов и совершать различные типы операций(арифметическая + обращение к памяти).

- RISC

  В этом подходе используются простые инструкции фиксированной длины, за счёт этого их декодирование достаточно простое. Каждая инструкция выполняется за один цикл. Для работы с памятью используются только специальные инструкции(load/store), остальные операции работают только с регистрами.

В RISC необходим меньший объём памяти для конкретной инструкции, но при этом общий размер кода больше. Декодирование инструкций в CISC сложнее и медленнее, на одну инструкцию приходится больше работы. [1]

2.2. Машинный язык в RISC архитектурах

Рассмотрим пример реализации RISC подхода в конкретных архитектурах: RISC-V [5, 6] и MIPS [2].

Форматы инструкций

Инструкции кодируются 32 битами.

В MIPS существует 3 формата инструкций(R, I, J). Каждая инструкция содержит код, который занимает 6 бит. При этом R инструкция со-

держит адреса трёх регистров, I — двух, а J ни одного, там находится адрес перехода.

В RISC-V 4 формата(R, I, U/J, S/B). Каждая инструкция кодируется 7 битами. При этом R инструкция содержит адреса трёх регистров, I и S/B — двух, а U/J – одного.

Регистры

Для хранения данных в рассматриваемых архитектурах выделяется 32 регистра. Существуют соглашение о том, в каких регистрах сохраняются определённые данные. Выделяют регистры для хранения 0, адреса возврата, указателей: глобального, на стек, на фрейм; регистры, значения в которых не будут изменены, при вызове подпрограммы(RISC-V — 11, MIPS — 8); регистры, в которые записываются аргументы и возвращаемые значения при вызове(RISC-V — 8, MIPS — 6).

Инструкции

- арифметические операции;
- логические операции;
- операции с константой;
- действия с памятью;
- условный переход;

  В MIPS отдельно реализована операция $<$, в условной конструкции можно проверить только равенство или не равенство значений. Другие переходы реализованы в виде псевдоинструкций.

- безусловный переход.

2.3. Симуляторы

Симулировать работу процессора можно на разных уровнях:

1. Уровень программной логики и оборудования. Это симуляторы, основанные на программируемых пользователем вентильных мат-

рицах(FPGA), которые создаются с помощью языков описания аппаратуры.

2. Уровень микропроцессора.

   Симуляция многоядерных систем с помощью ПЛИС слишком сложная, затратная и долгая, поэтому в таких случаях используется микроархитектурная симуляция, в том числе в симуляторе тысячеядерной системы ZSim [4]

3. Уровень инструкций.

   Для их создания используются языки программирования высокого уровня, не воспроизводит мелкие детали программного обеспечения, только поддерживает переменные описывающие состояние процесса.

3. Описание решения

3.1. Проектирование набора команд процессора

В первую очередь определена длина машинного слова процессора — 16 бит. Из них четыре бита выделим под кодирование операции, таким образом можно закодировать 16 инструкций. Такое же количество памяти выделим и для кодирования номера регистра.

В таких условиях в одной инструкции можно сохранить три регистра без дополнительных полей(Рис. 1, формат 1). Команда такого формата выбраны для представления арифметических и логических операций: сложение, вычитание, умножение, деление, и, или, исключающее или, побитовые сдвиги. В первый регистр записывается результат действий между двумя последующими.

формат	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
1	код операции	регистр	регистр	регистр
2	код операции	регистр	число	
3	код операции	регистр	регистр	число

Рис. 1: Форматы инструкций

В процессоре необходимо предусмотреть команду с помощью которого данные попадут в регистры. В процессорах с RISC архитектурой для этого часто используются арифметические операции с константой. В данном процессоре из-за ограничения на количество команд не рационально добавлять большое количество инструкций с константой, которые можно заменить комбинацией стандартных арифметических операций с единственной командой — добавление данных в регистр. Также в случае с операцией с константой инструкция должна содержать два регистра, следовательно для числа останется четыре бита. Для операции добавления в регистр требуется один регистр, а значит под числовые данные остаётся восемь бит(Рис. 1, формат 2). Для добавления больших данных можно использовать инструкцию побитового сдвига.

Кроме регистров процессор содержит память, а значит необходимы

операции записывающие данные из регистра в память и извлекающие из неё. Для данных операций требуется два регистра: первый хранит данные, второй адрес в памяти. Остаются свободными четыре бита, их выделим для сдвига адреса в памяти(Рис. 1, формат 3).

Следующим необходимым типом команд являются переходы: безусловный и условный. Для условного перехода ключевым является подбор условия при котором будет происходить переход. Важно чтобы все стандартные случаи(равно, не равно, строгое и не строгое сравнение) можно было представить небольшим количеством инструкций. Используем для перехода следующее условие: регистр не отрицателен. Кроме регистра в инструкции укажем на какое количество команд происходит переход при выполнении условия(Рис. 1, формат 2). Для безусловного перехода тоже необходимо количество команд, кроме этого понадобится регистр, в который будет записан адрес возврата(формат 2). Реализуем дополнительно инструкцию безусловного перехода относительно регистра. В ней сдвиг будет определяться значением сохранённым в регистре в дополнении к числу(формат 3).

Таким образом определены пятнадцать команд и остаётся одна свободная, которая выделена для дополнительных операций.(Таблица 1)

3.2. Реализация симулятора

Для спроектированной архитектуры на практике реализован симулятор процессора на уровне инструкций. Для этого использован язык программирования Python.

Процессор представлен счётчиком команд, регистрами и памятью.

Входными данными симулятора является файл с программой, написанной с использованием предложенной системы команд. Он транслируется в двоичный код, который потом исполняется процессором.

Для всех типов инструкций реализованы методы: трансляция в двоичный код и исполнение двоичного кода.

В качестве дополнительных операций реализованы системные вызовы: ввод, вывод информации и выход из программы.

команда	**действие**	**формат**
изп	выгрузка из памяти	3
вп	загрузка в память	3
сл	сложить	1
выч	вычесть	1
умн	умножить	1
дел	разделить	1
свп	сдвиг вправо	1
свл	сдвиг влево	1
вр	загрузка в регистр	2
и	and	1
или	or	1
иили	xor	1
усл	условный переход	2
пер	переход	2
опер	относительный переход	3
доп	дополнительные команды	4

Таблица 1: Набор команд процессора

3.3. Программы для процессора

Работоспособность реализованного симулятора проверена с помощью реализации некоторых программ.

Листинг 1: Вычисление чисел Фибоначчи.

```
    вр p1 100//main
    вр p8 5
    вр p9 1
    вр p10 1
    вр p15 3
    выч p8 p8 p15
    усл p8 2
    опер p0 p1 0//end main
    сл p14 p9 p0
    сл p9 p10 p0
    сл p10 p14 p10
    вр p15 1
    выч p8 p8 p15
    пер p0 -7
```

Листинг 2: Вычисление факториала.

```
    вр р4 2//main
    сис
    сл р8 р5 р0
    сл р4 р8 р0
    пер р1 6
    сл р5 р4 р0
    вр р4 1
    сис
    вр р4 0
    сис//end main
    вр р12 1//func factorial result
    вр р13 1//i
    вр р15 1
    выч р14 р4 р13//start for
    усл р14 3
    сл р4 р12 р0
    опер р0 р1 0//return
    умн р12 р12 р13
    сл р13 р13 р15
    пер р0 -6//end factorial
```

Заключение

1. Спроектирована архитектура набора команд для процессора.

   Проанализированы возможные варианты и выбраны 16 команд(арифметические и логические операции, операции с памятью и переходы), которые стали языком для написания программ, исполняемых процессором.

2. Реализован симулятор для этого процессора.

   Создана программа, которая преобразует инструкции на спроектированном языке в двоичный код и исполняет его, тем самым симулируя работу процессора.

   https://github.com/katvus/my_processor

3. Симулятор протестирован на простых примерах.

   Написаны программы, с помощью которых можно вычислить факториал и числа Фибоначчи.

В дальнейшем можно рассмотреть работу процессора на уровне логических схем. Это поможет лучше понять преимущества и недостатки выбранной архитектуры набора команд.

Список литературы

[1] ISA wars: Understanding the relevance of ISA being RISC or CISC to performance, power, and energy on modern architectures / Emily Blem, Jaikrishnan Menon, Thiruvengadam Vijayaraghavan, Karthikeyan Sankaralingam // ACM Transactions on Computer Systems (TOCS). — 2015. — Vol. 33, no. 1. — P. 1–34.

[2] MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set, Revision 3.00. — 2010. — URL: https://pages.hmc.edu/harris/class/e155/11/MIPS32InstructionSet.pdf (дата обращения: 21 октября 2024 г.).

[3] Mavaddat Farhad, Parhami Behrooz. URISC: the ultimate reduced instruction set computer // International Journal of Electrical Engineering Education. — 1988. — Vol. 25, no. 4. — P. 327–334.

[4] Sanchez Daniel, Kozyrakis Christos. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems // ACM SIGARCH Computer architecture news. — 2013. — Vol. 41, no. 3. — P. 475–486.

[5] Waterman Andrew, Asanovi´c Krste. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213. — 2019. — URL: https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf (дата обращения: 21 октября 2024 г.).

[6] Сара Л. Харрис Дэвид Харрис. Архитектура // Цифровая схемотехника и архитектура компьютера: RISC-V / Ed. by А. Ю. Романов. — ДМК Пресс, 2021. — P. 400 – 411.