Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление системного и прикладного программного обеспечения

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

курса «Функциональная схемотехника»

по теме: «Расширение возможностей учебного процессорного ядра schoolRISCV»

Вариант № 6

Выполнил студент:

Тюрин Иван Николаевич

группа: P33102

Преподаватель:

Кустарев П. В.,

Васильев С.Е.

Санкт-Петербург, 2024 г.

Содержание

Лабораторная работа № 4
Расширение возможностей учебного процессорного ядра schoolRISCV

1. Введение

2. Задание варианта № 6

В лабораторной работе вам предлагается разобраться во внутреннем устройстве простейшего процессорного ядра архитектуры RISC-V. Результатом изучения микроархитектуры процессорного ядра и системы команд RISC-V станут ваши функциональные и нефункциональные модификации ядра. Основное задание:

1. Модифицировать процессорное ядро, в соответствии с вашим вариантом;

2. Подготовить тестовое окружение системного уровня и убедиться в корректности вашей реализации путём запуска симуляционных тестов.

Варианты с арбитром должны реализовать однотактовую выборку двух инструкций и их арбитраж в остальную часть процессорного ядра. Вместо того, чтобы выбирать и исполнять одну инструкцию - процессорное ядро должно выбирать две инструкции одновременно и управлять их доступом к исполняемому ресурсу ядра.

3. Выполнение задания

В ходе выполнения задания потребовалось изучить устройство выданной модели процессора. С учетом условий задания возникло некоторое недопонимание. Пытаясь выполнить первый шаг я потерял много времени впустую, ведь там нужно сделать некоторую «симуляционную задержку», которую позже нужно убрать. Мне не удалось сделать задержку по какой-то причиние: либо я не понял, что нужно сделать, либо я не правильно воспользовался возможностями симулятора и неверно описал задержку при передаче сигналов, либо это невозможно сделать по причине устройства модели.

Почти сразу встает вопрос «чего мы пытаемся добиться?» (исходя из формулировки задания это не ясно). Кажется, что так можно было бы повысить параллелизм инструкций исполняющихся за такт и/или добиться какой-то спекулятивности, но возникают следующие проблемы.

- Сам процессор имеет конвейер из 2 стадий: IF (Instruction Fetch) и WB (Write Back), рис. 1.1 (синий пунктир). Причем, последняя в случаях ветвления отсутствует, рис. 1.2. Между IF и WB занята память команд, регистровый файл и АЛУ. Такое устройство не позволяет одновременно, без дублирования ресурсов (АЛУ, регистров), производить вычисления.

- Арбитр из предыдущей лабораторной работы чрезмерно простой: он лишь по порядку выводит данные со входов которые к нему пришли и не содержит логики распределения приоритетов. Поэтому он не может изменить динамически порядок инструкций.

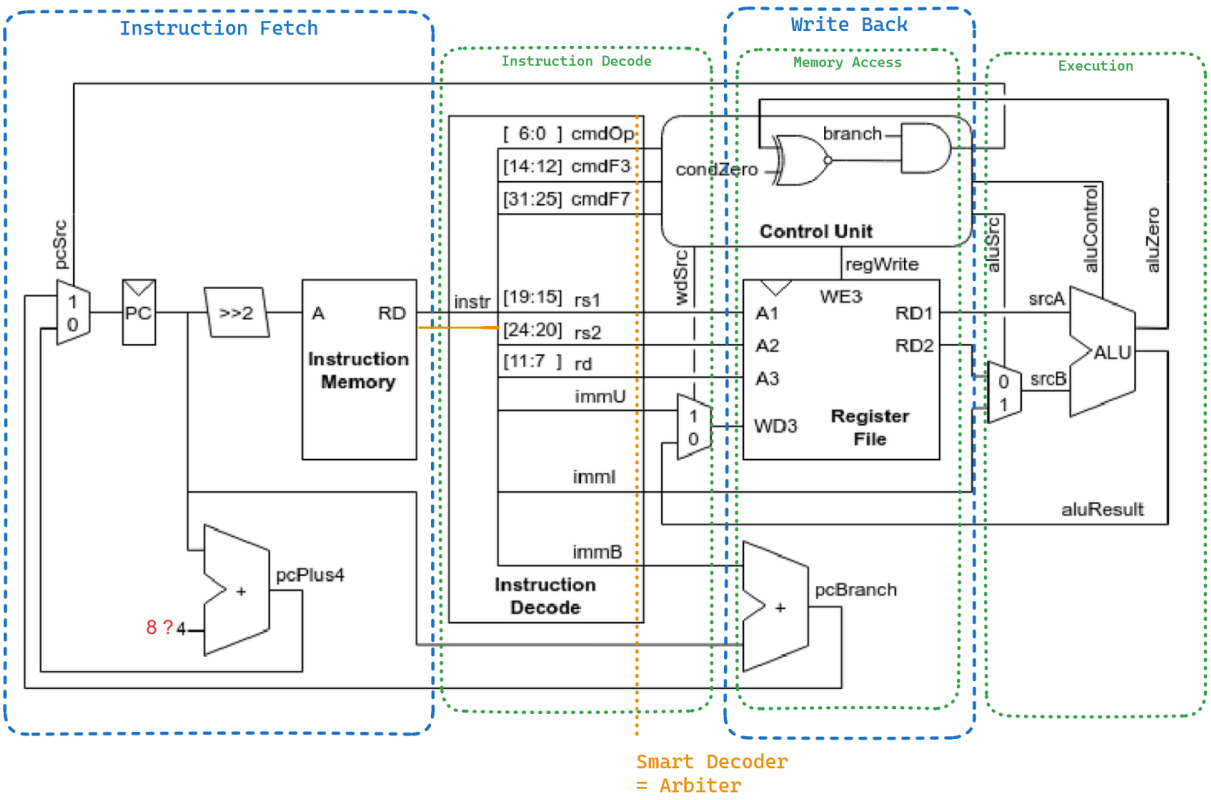

Рис. 1.1: Исследование модели ЦПУ schoorRISCV

```
ADD: { `RVF7_ADD,  `RVF3_ADD,  `RVOP_ADD  } : begin regWrite = 1; aluControl = `ALU_ADD;                  0  end
OR:  { `RVF7_OR,   `RVF3_OR,   `RVOP_OR   } : begin regWrite = 1; aluControl = `ALU_OR;                   0  end
SRL: { `RVF7_SRL,  `RVF3_SRL,  `RVOP_SRL  } : begin regWrite = 1; aluControl = `ALU_SRL;                  0  end
SLTU:{ `RVF7_SLTU, `RVF3_SLTU, `RVOP_SLTU } : begin regWrite = 1; aluControl = `ALU_SLTU;                 0  end
SUB: { `RVF7_SUB,  `RVF3_SUB,  `RVOP_SUB  } : begin regWrite = 1; aluControl = `ALU_SUB;                  0  end

ADDI:{ `RVF7_ANY,  `RVF3_ADDI, `RVOP_ADDI } : begin regWrite = 1; aluControl = `ALU_ADD;  aluSrc = 1;     0  end
LUI: { `RVF7_ANY,  `RVF3_ANY,  `RVOP_LUI  } : begin regWrite = 1;                                    wdSrc = 1;  end

BEQ: { `RVF7_ANY,  `RVF3_BEQ,  `RVOP_BEQ  } : begin              aluControl = `ALU_SUB;                    branch = 1; condZero = 1; end }branch
BNE: { `RVF7_ANY,  `RVF3_BNE,  `RVOP_BNE  } : begin              aluControl = `ALU_SUB;                    branch = 1;               end
```

Рис. 1.2: Исследование инструкций доступных в реализации schoolRISCV

Идея №1. Прочитав внимательно задание, откину затею сделать задержку внутри ядра, ведь ее нужно будет заменить модулем арбитра. Чтобы как-то удовлетворить заданию, можно сделать чтение 2 инструкций за раз, которые потом арбитр будет по порядку отправлять на исполнение. То есть, команды будут фетчиться раз в 2 такта, рис. 1.1 (оранжевый пунктир).

Идея №2. Тут же в голову приходит мысль, что не всегда можно считывать сразу 2 команды и последовательно их исполнять: например, branch (ветвление) может потребовать смены PC на значение отличное от 8 (шага в 2 инструкции). Поэтому нужно каким-то образом давать арбитру сигналы валидности команды для арбитража, что наталкивает на идею дублировать декодер для каждой считываемой инструкции, а затем в «conflict detector»'е выставлять сигналы валидности.

А еще в арбитре нужно убрать стадию ожидания входных данных (как было в предыдущей ЛР), которая занимает целый такт. Поэтому стоит «кешировать» по 2 инструкции на каждом цикле арбитража.

Мы не можем пропускать инструкции, поэтому мы можем либо исполнить 2 по порядку, либо только первую.

Идея №3. Наверное нам не нужно даже хитро управлять PC, потому что арбитр «кеширует» инструкции, которые к нему пришли, и, когда он исчерпал возможность к исполнению (только первую удалось исполнить), он просто закэширует 2 актуальные инструкции, на которые указывает PC.

При этом у арбитра подразумевается сигнал `valid_o=1`, а номер исполняемой инструкции `t_number_o` нас интересует лишь для отладки.

Идея №4. Другая мысль была растянуть конвейер на большее количество стадий, но тогда нужно сохранять все управляющие сигналы и адреса записи в память. Что значительно усложняет устройство процессора. Поэтому было принято решение временно отказаться от такой идеи.

3.1. Итоговая микроархитектура ЦПУ

В результате применения описанных идей, была разработана микроархитектура измененного ЦПУ, рис. 1.3. В ней между декодером инструкций и памятью инструкций добавляется модуль модифицированного арбитра и некоторый предсказатель конфликтов.

Арбитр модифицирован таким образом, чтобы он считывал за раз 2 инструкции и сохранял их во внутренней памяти, но исполнял их в соответствии с сигналами валидности, которые он получает от предсказателя конфликтов. Арбитр так же лишился стадии ожидания ввода, которая была в начальной реализации. Это требуется для того, чтобы не терять такты исполнения процессора. Временную диаграмму его сигналов можно видеть на рис. 1.4.

Предсказатель конфликтов подразумевается как комбинаторный блок, который по двум входным инструкциям определяет, можно ли исполнить обе считанные инструкции или можно выполнить только одну.

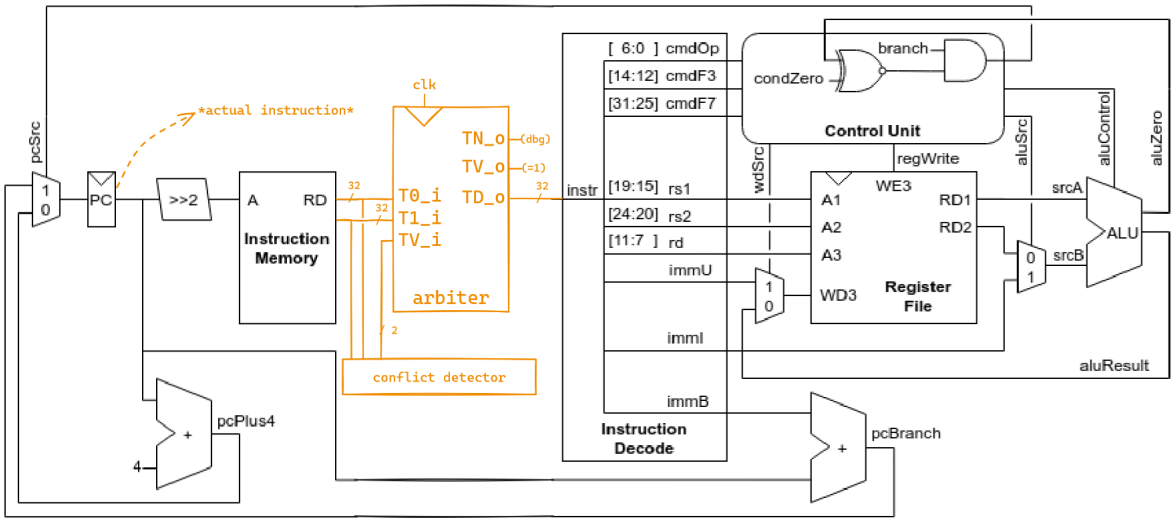

Рис. 1.3: Модифицированная микроархитектура процессора

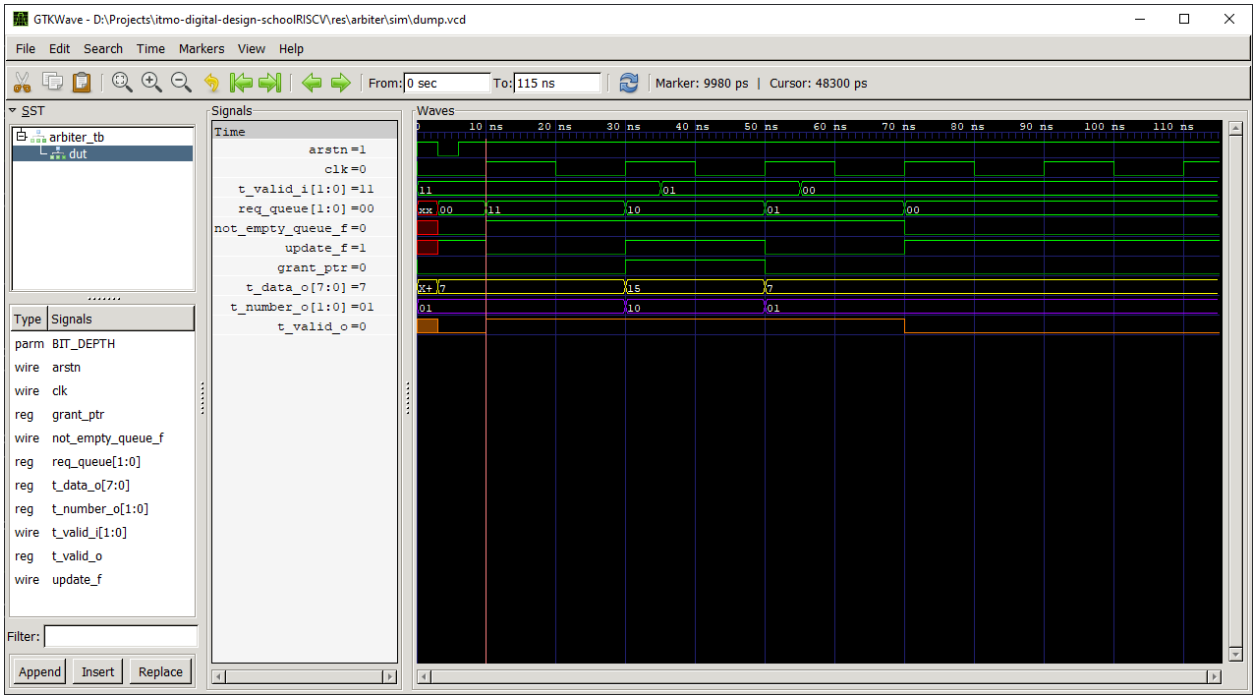

Рис. 1.4: Временная диаграмма состояний измененной версии арбитра

4. Вывод

К сожалению, после попытки внедрить новый арбитр в имеющуюся модель, произошли проблемы с языковыми возможностями симулятора, из-за чего пришлось вернуть систему в начальное состояние, но какие-то изменения остались и модель перестала работать корректно. Времени доделать модель до полноценной работы не хватило.

В качестве вывода можно указать, что моя реализация выглядит наиболее требовательной к переработке устройства процессора, но при этом не

приносит никаких преимуществ в вычислениях, а лишь усложняет устройство ядра.

Как уже отмечалось, решение возникших проблем лежит в значительном изменении архитектуры ядра, добавления дополнительных стадий конвейера с сохранением управляющих сигналов, добавление дополнительных вычислительных модулей, усложнением модуля памяти, чтобы можно одновременно писать в разные регистры, добавлением «теневых» (или временных) регистров, значение из которых бы потом специальным образом, при достижении некоторого условия переносились в основные регистры.

В рамках работы с schoolRISCV стало видно, насколько тяжело в сфере цифровой электроники что-то разрабатывать без проприетарного ПО: симулятор IcarusVerilog не поддерживает все возможности современных стандартов SystemVerilog, а все открытые приложения для просмотра VCD-файлов очень неудобные.

Другие комментарии по разработке schoolRISCV (на Windows) можно найти в моем репозитории: github.com/e1turin/itmo-digital-design (lab-4/schoolRISCV).