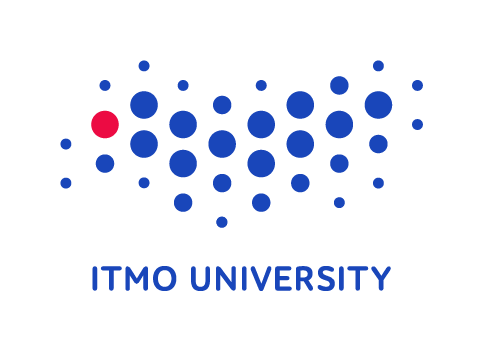
Санкт-Петербургский Национальный Исследовательский

Университет ИТМО

Факультет программной инженерии и компьютерной техники



Вариант № 5

Лабораторная работа № 4

«Расширение возможностей учебного процессорного ядра schoolRISCV»

по дисциплине

*«Функциональная схемотехника»*

Выполнил: Антонов Всеволод Владимирович

Студент группы: P33301

Преподаватель:

Салонина Екатерина Александровна

Санкт-Петербург, 2023 г.

Цель работы:

* Ознакомиться с архитектурой RISC-V;
* Получить базовое понимание работы микропроцессорных ядер;
* Получить навыки работы с системами «средней» сложности;

Задание:

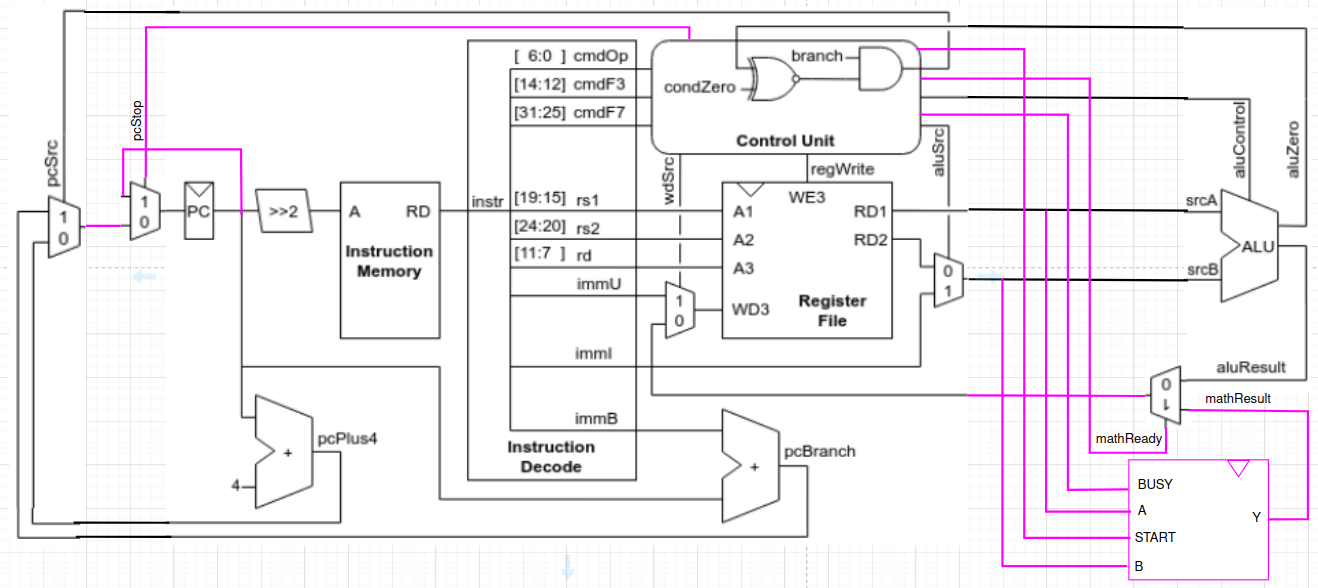
1. Расширить систему команд процессора двумя новыми командами в соответствии с вариантом;

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

| **Функция** | **Ограничения** |
| --- | --- |
|  | 1 сумматор и 2 умножителя |

| **Команда №2** | **Пояснения** |
| --- | --- |
| ANDI | Стандартная команда из набора RV32I. Реализовать в АЛУ. |

Микроархитектурная схема модифицированного процессорного ядра:

  
Описание изменений:

1. Добавлен дополнительный мультиплексор для предотвращения изменения регистра PC, формирующий зацикливание по активному сигналу pcStop.
2. Параллельно с АЛУ добавлен модуль для вычисления функции FUNC.
3. Добавлены сигналы mathBusy и mathReady, по которым определяется момент получения результата FUNC.
4. Добавлен дополнительный мультиплексор для получения результата работы FUNC или АЛУ (по сигналу mathReady, равному 1 и 0 соответственно).
5. SrcA и SrcB идут и в АЛУ, и в FUNC, но записываются во внутренние регистры последнего только при активном mathStart и по фронту тактового сигнала.

Описание алгоритмов:

1. **ANDI** - однотактовая инструкция, выдающая результат логического И двух аргументов (непосредственного операнда и содержимого регистра). Для этой команды потребовалось добавить её декодирование (funct3 и opcode) и операцию AND в ALU.

Таблица истинности логического И:

| A | B | R |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Количество тактов: 1.

Для тестирования команды была разработана программа, из нескольких тестов, в каждом из которых происходит загрузка регистрового аргумента и верного результата, выполнение ANDI первого и непосредственного операнда и сравнение результата с последним. Если они равны, то происходит переход к следующему тесту, и при верном выполнении всех тестов в x10 записывается 0, иначе следующие (если есть) тесты не выполняются, а в x10 записывается 1.

start: li a2, 9 ## a = 1001

li a1, 8 ## y = 1000

andi a0, a2, 12 ## b = 1100

bne a0, a1, fail ## 9 & 12 = 8

li a2, 254 ## a = 11111110

li a1, 250 ## y = 11111010

andi a0, a2, 251 ## b = 11111011

bne a0, a1, fail ## 254 & 251 = 250

li a2, 0 ## a = 0

li a1, 0 ## y = 0

andi a0, a2, 0 ## b = 0

bne a0, a1, fail ## 0 & 0 = 0

li a2, 42 ## a = 0101010

li a1, 0 ## y = 0

andi a0, a2, 69 ## b = 1000101

bne a0, a1, fail ## 42 & 69 = 0

li a2, 14 ## a = 0001110

li a1, 8 ## y = 0001000

andi a0, a2, 88 ## b = 1011000

bne a0, a1, fail ## 14 & 88 = 8

success: li a0, 0 ## all test passed

beqz zero, end

fail: li a0, 1 ## at least one test failed

beqz zero, end

end: beqz zero, end ## while(1);

1. **FUNC** - многотактовая инструкция, выдающая результат вычисления математической функции двух аргументов (содержимого регистров). Если до этого функция не запускалась, то PC перекрывается и выставляется mathStart, который снимается при mathBusy (с сохранением перекрытого PC). По завершении выполнения функции (mathBusy на предыдущем такте был активен, а на текущем нет) PC открывается и выставляется разрешение записи в регистровый файл.

Количество тактов: 81.

Для тестирования команды была разработана программа, из нескольких тестов, в каждом из которых происходит загрузка регистровых аргументов и верного результата, вычисление функции от первых и сравнение результата с последним. Если они равны, то происходит переход к следующему тесту, и при верном выполнении всех тестов в x10 записывается 0, иначе следующие (если есть) тесты не выполняются, а в x10 записывается 1.

start: li a0, 251 ## a = 251

li a1, 254 ## b = 254

li a2, 63007 ## y = 63007

func rx10, rx11, rx10 ## 251\*251 + cbrt(254) = 63007

bne a0, a2, fail

li a0, 0 ## a = 0

li a1, 0 ## b = 0

li a2, 0 ## y = 0

func rx10, rx11, rx10 ## 0\*0 + cbrt(0) = 0

bne a0, a2, fail

li a0, 0 ## a = 0

li a1, 42 ## b = 42

li a2, 3 ## y = 3

func rx10, rx11, rx10 ## 0\*0 + cbrt(42) = 3

bne a0, a2, fail

li a0, 12 ## a = 12

li a1, 9 ## b = 9

li a2, 146 ## y = 146

func rx10, rx11, rx10 ## 12\*12 + cbrt(9) = 146

bne a0, a2, fail

li a0, 14 ## a = 14

li a1, 88 ## b = 88

li a2, 200 ## y = 200

func rx10, rx11, rx10 ## 14\*14 + cbrt(88) = 200

bne a0, a2, fail

success: li a0, 0 ## all test passed

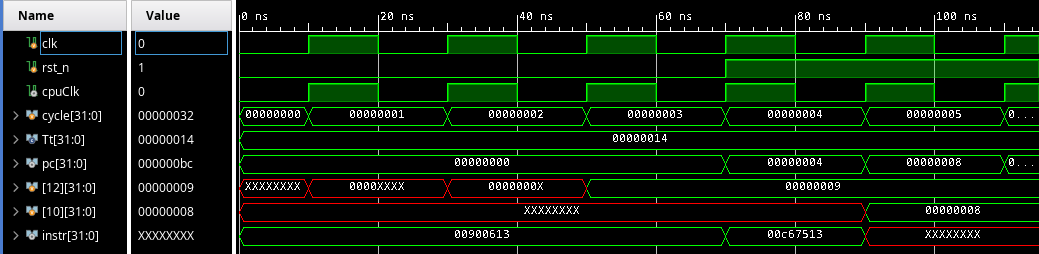
beqz zero, end

fail: li a0, 1 ## at least one test failed

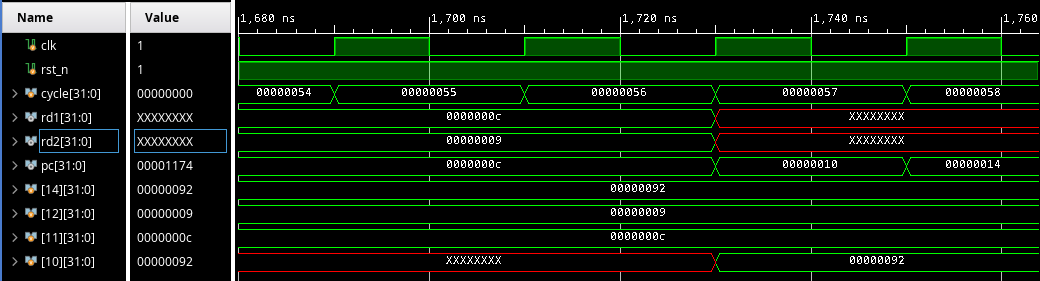
beqz zero, end

end: beqz zero, end ## while(1);

Временные диаграммы:

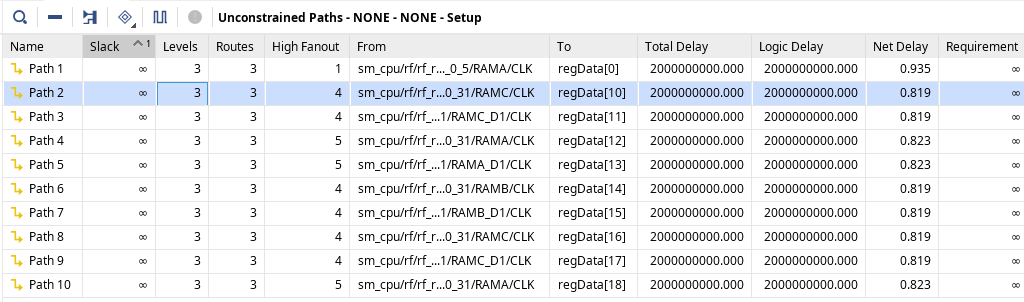


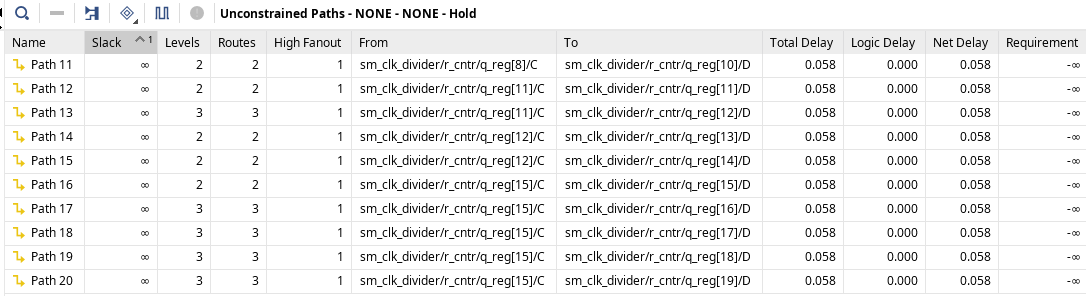
В регистре x12 лежит значение 9. После выполнения соответствующей команды (00c67513 = ANDI x10, x12, 12) в регистре x10 оказывается значение 12 & 9 = & = = 8.



В этом тесте вычисляется значение функции . Можно заметить, что rd1 == x11 и rd2 == x12, так как это соответствующие аргументы. В x14 хранится истинный результат. После такта PC открывается, а rd1 и rd2 становятся неопределёнными, так как программа закончилась. В x10 же записывается результат вычисления выражения, который оказывается верным.

Временные параметры проекта:





Выводы:

Я изучил основы функционирования процессора с архитектурой RISC-V и реализовал в нём две команды, для одной из которых мне пришлось делать его многотактным. Я обратил внимание, что для реализации многотактности существует несколько вариантов - можно добавить к регистру PC сигнал Enable или дополнительный мультиплексор с зацикливанием на себя (я выбрал последний вариант). Кроме того, из-за проблем с использованием виртуальной машины, потребляющей почти всю RAM компьютера до такой степени, что он полностью зависал, я решил упростить себе жизнь, сначала установив требуемый софт в Docker-контейнере, а потом отказавшись и от этой идеи и установив RISC-V-компилятор на хостовую ОС Manjaro, при том, что Java и Vivado уже были установлены, и импортировав проект в Vivado. Это позволило мне сэкономить RAM, время и нервы.