Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“НАЦИОНАЛЬН ЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет ПИиКТ

ОТЧЁТ

По лабораторной работе №4

По предмету: Функциональная схемотехника

Вариант 2

Студент:

Андрейченко Леонид Вадимович

Группа P33301

Преподаватель:

Солонина Екатерина Александровна

Санкт-Петербург

2023

Цель работы

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

Описание задания

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

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

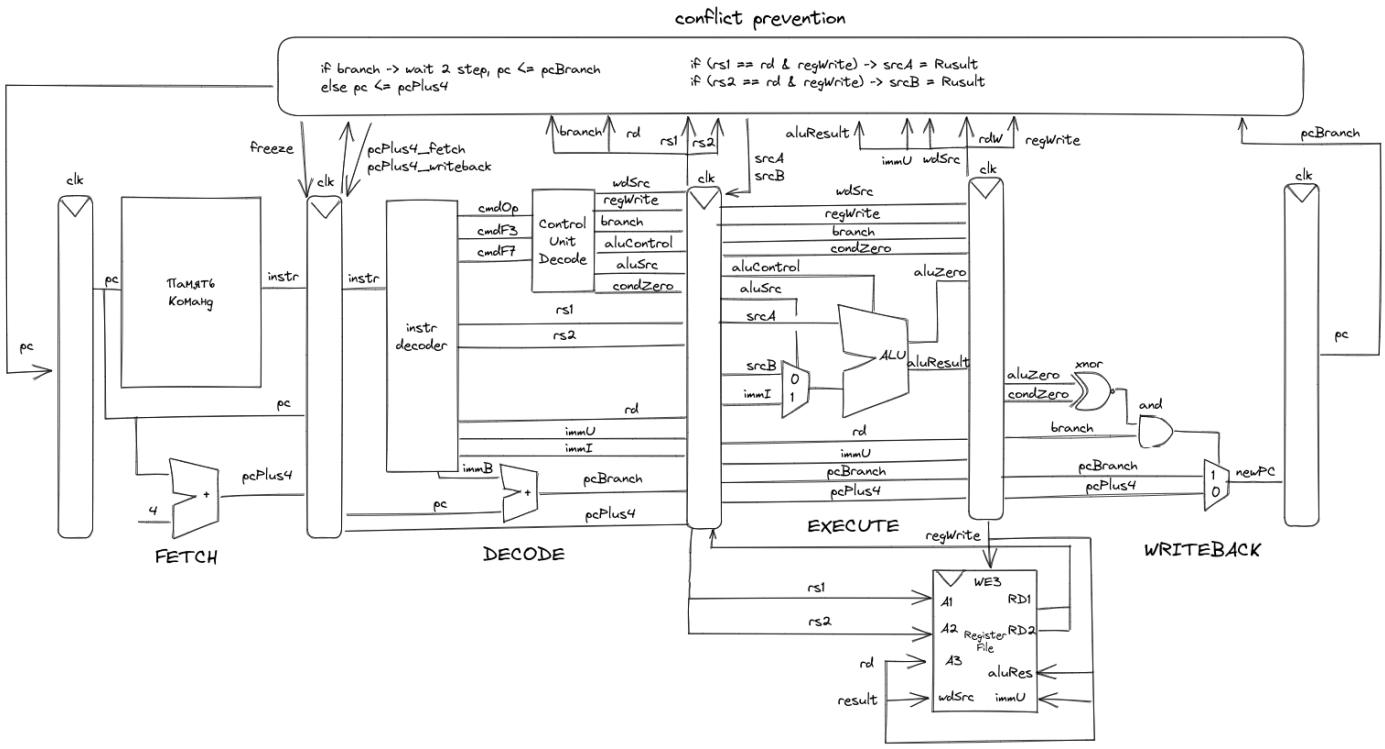
Вариант

2	BGE	Стандартная команда из набора RV32I

Микроархитектурная схема ядра

Изначальная однотактовая архитектура процессора была переделана под конвейерную.

[Ссылка на github репозиторий проекта]



Данный процессор состоит из следующих модулей:

- Стадии конвейера
    - Fetch - на данной стадии происходит выборка команды из памяти. На вход подается адрес команды, которую необходимо выбрать, она выбирается и подается на выход из модуля. Также для выбора будущей команды передается как номер только что выбранной команды (для команд ветвления) так и номер следующей команды
    - Decode - главная задача данного этапа - сформировать управляющие сигналы и подготовить все данные для исполнения данной команды. В начале команда подается в блок декодирования, он формирует коды (cmdOp, cmdF3, cmdF7), по которым можно будет в дальнейшем сформировать управляющие сигналы, а так же константы и адреса операндов, которые потом могут использоваться конвейером. В конце сигналы cmdOp, cmdF3, cmdF7 подаются в блок формирования управляющих сигналов.
    - Execute - на данной стадии вычисляется результат как арифметических команд, так и команд ветвления. Данный результат на выходе, также формируется флаг Zero, который в дальнейшем будет использован в вычислении будущего PC (для команд ветвления).
    - Writeback - тут мы просто высчитываем следующий адрес команды, которая будет выполнена. Актуальна только для команд ветвления
- Память команд - данный модуль невероятно простой - на вход мы подаем адрес команды, которую хотим считать, на выходе получаем данную команду.
- Декодер инструкций - задача модуля разобрать команду согласно спецификации. Модуль формирует такие сигналы как
    - CmdOp, cmdF3, cmdF7 - благодаря данным полям мы можем однозначно идетифицировать команду.
    - Rs1, rs2 - адреса операндов, с которыми будет работать конвейер
    - Rd - адрес куда в будущем будет записан результат
    - ImmI ImmU ImmB - константы, которые формируются исходя их спецификации risc v
- Модуль управляющих сигналов - данный модуль принимает на вход сигналы, которые позволяют однозначно идентифицировать команду, и исходя из этих значений сформирует управляющие сигналы для данной команды.
- Арифметико-логическое устройство - данный блок принимает на вход два числа и сигнал операции АЛУ. После вычисления результата формируется сигнал Zero.
- Модуль предотвращения конфликтов данный модуль был разработан с целью предотвращения конфликтов конвейера. Он предотвращает как конфликты данных, так и конфликты управления.
- Регистровый файл - данная схема является комбинационной. Подавая адреса на входы A1 A2, мы получаем значения данных регистров на выходах RD1 RD2. При одновременной подаче на вход WE3 единицы, A3 адреса, wdSrc значения, данное значение записывается по адресу.

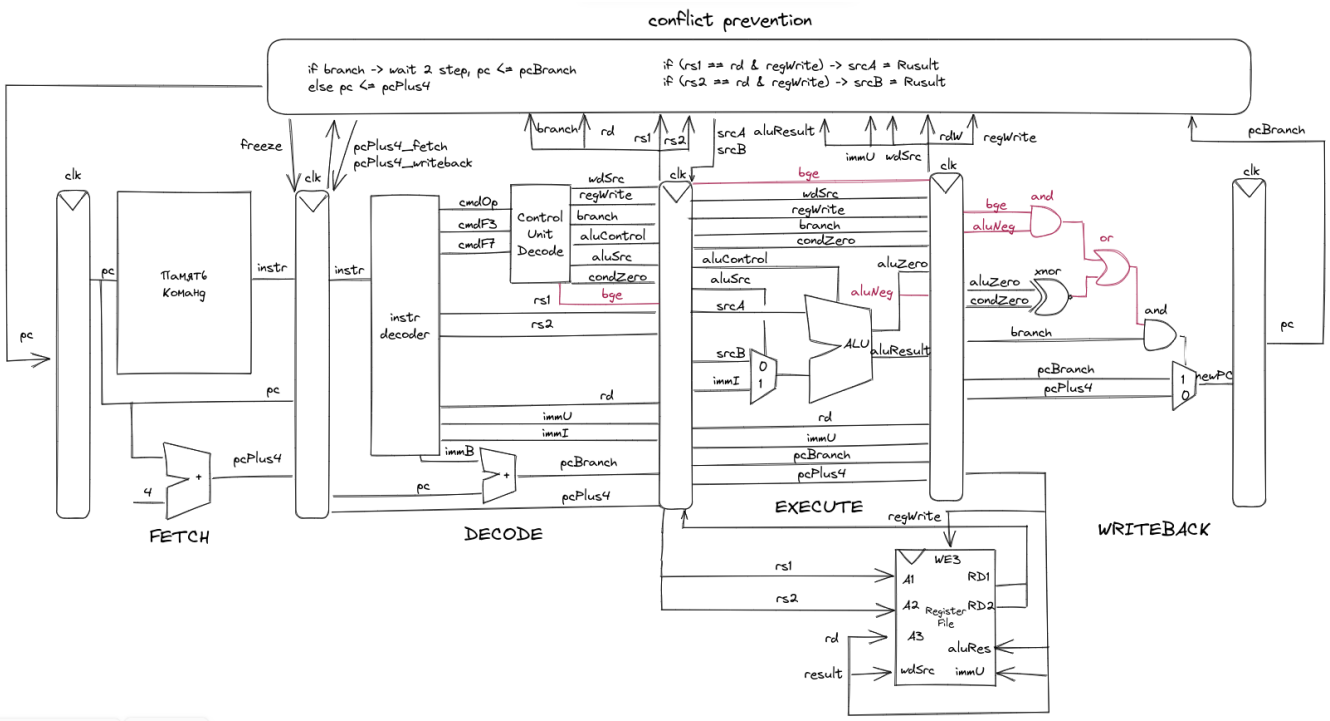
После того как процессор был переделан, необходимо было добавить первую команду - bge - выполните ответвление, если регистр rs1 больше или равен rs2, используя знаковое сравнение. Команда была взята из спецификации.

bge rs1,rs2,offset : if (x[rs1] >=s x[rs2]) -> pc += offset

Для этого микроархитектура процессора была дополнена. А им енно

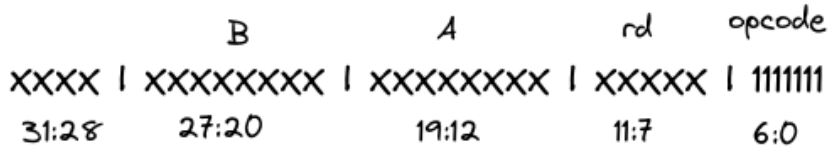
- Bge - управляющий сигнал, который будет использоваться при выборе будущего адреса команды
- AluNeg - сигнал, который говорит что результат alu отрицателный
- Добавлены схемы and or

Логика выполнения данной команды, следующая - мы формируем сигнал bge и aluNeg далее если установлены оба сигнала, то происходит переход, если нет, то будет просто выполнена следующая по очереди команда.



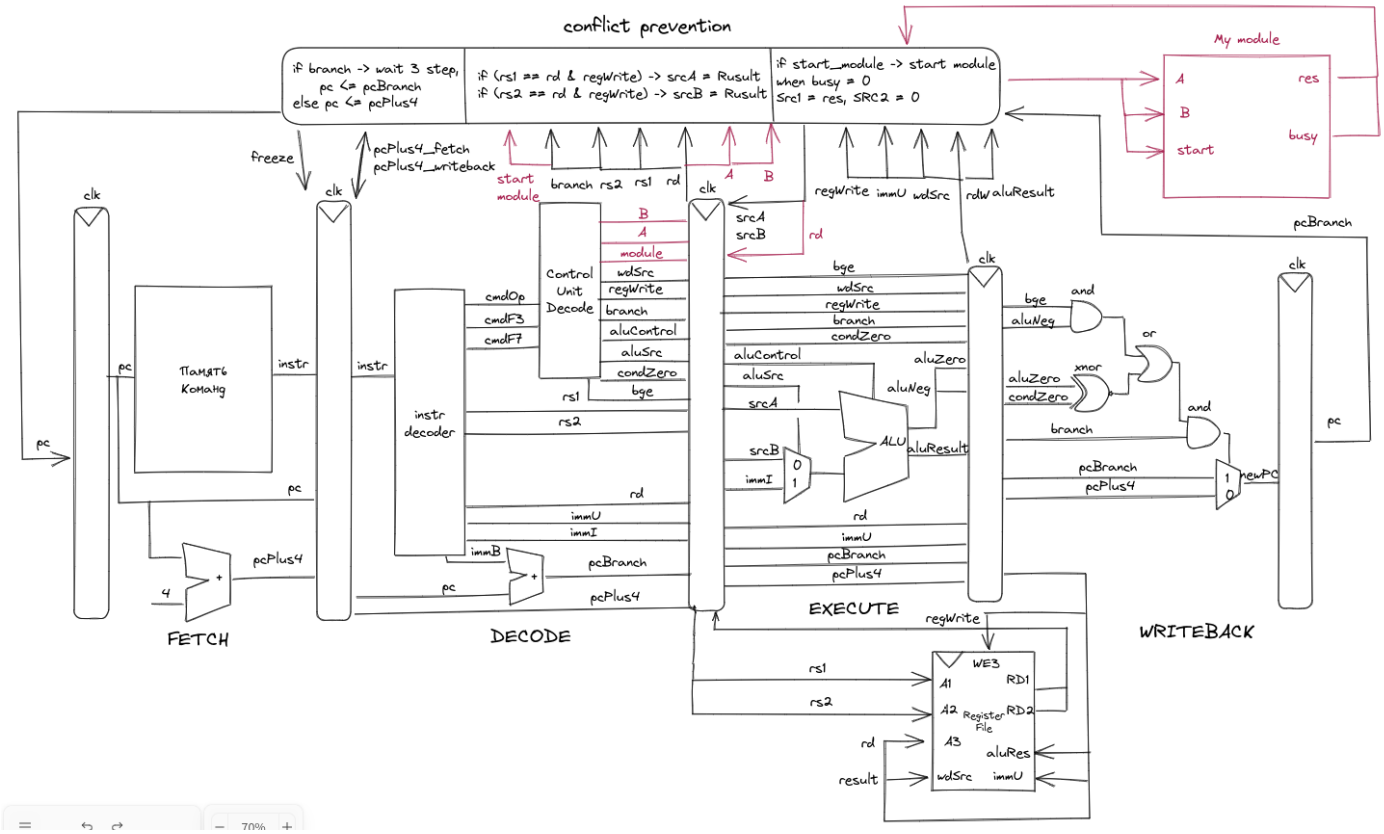
Для добавления команды, реализующей работу модуля, необходимо было сначала придумать ее двоичное представление в “машинном коде”.

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type



- В позициях 31-28 могут быть любые значения
- В позициях 27-20 записывается значение числа B, которое будет использовано в функции
- В позициях 19-12 записывается значение числа A, которое будет использовано в функции
- В позициях 11-7 записывается номер регистра, в который будет потом записан результат
- В позициях 6-0 классически располагается Opcode, с помощью которого будет опознана команда

Данная команда на этапе декодирования формирует сигнал начала работы модуля, и два входных числа A, B. Пока модуль вычисляет результат результат, конвейер в ожидании исполняет команды nop, как только результат готов он подается на этап execute. Далее логика работы ан алогична команде add.



Исполнение добавленных команд

Алгоритм работы команды bge

- На стадии fetch выбирается команда bge, она подается на выход вместе с адресом данной инструкции и адресом следующей.
- На стадии Decode формируются управляющие сигналы branch, bge, вычисляется константа, которая будет использована для относительной адресации
- Модуль разрешения конфликтов подаст сигнал freeze, который на время приостановит работу конвейера до окончания вычисления адреса следующей команды.
- На стадии execute высчитается результат сравнения двух регистров и сформируются флаги zero neg.
- На стадии writeback будет подсчитан адрес следующей команды, если результат работы alu отрицательный, то переход не произойдет, если нет то будет выбрана команда по следующему адресу
- Данный адрес передается в стадию fetch, выбирается новая команда, с конвейера снимается заморозка, и он продолжает свою работу

Алгоритм работы команды func

- На стадии fetch выбирается команда func, она подается на выход вместе с адресом данной инструкции и адресом следующей.
- На стадии Decode формируются управляющие сигналы start_module, A, B
- Модуль разрешения конфликтов подаст сигнал freeze, который на время приостановит работу конвейера до окончания вычисления результата модуля. Значение rd сохраняется до получения результата. Модуль начинает вычисления. После получения результата он передается на стадию execute в качестве аргумента srcA, в srcB передается 0. На вход rd передается сохраненное значение. Конвейер снимает заморозку, и следующая команда поступает на стадию decode.
- На стадии execute srcA и srcB суммируются. В конце полученный результат будет записан по нужному адресу.

Результаты тестирования

Пример теста команды bge

Данная программа сначала инициализирует регистры a0, a1 нулями, после этого загружает в регистр a1 значение 2. Далее прибавляет к регистру a0 единицу и проверяет, если регистр a1 строго больше, чем a0, то продолжаем наращивать значение в a0. Как только значения в регистрах сравняется - программа закончит свое выполнение.

```
1           .text
2
3   start:  lui a0, , 0x0          # t0 = 0
4           lui a1, , 0x0
5           addi a1, a1, 2
6   counter: addi a0, a0, 1         # t0 = t0 + 1
7           bge a1, a0, counter  # if t0 == t1 then counter
```

Потактовый консольный вывод

```
 0  pc = xxxxxxxx instr = 00000013 a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   new/unknown
 1  pc = 00 instr = 00000537 a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   new/unknown
 2  pc = 04 instr = 000005b7 a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   lui   $10, 0x00000000
 3  pc = 08 instr = 00258593 a0 = 0x00000000 a1 = 0xxxxxxxxx   :   lui   $11, 0x00000000
 4  pc = 0c instr = 00150513 a0 = 0x00000000 a1 = 0x00000000   :   addi  $11, $11, 0x00000002
 5  pc = 10 instr = fea5dee3 a0 = 0x00000000 a1 = 0x00000002   :   addi  $10, $10, 0x00000001
 6  pc = 14 instr = 00000013 a0 = 0x00000001 a1 = 0x00000002   :   bge   $11, $10, 0xfffffffc (-4)
 7  pc = 14 instr = 00000013 a0 = 0x00000001 a1 = 0x00000002   :   nop
 8  pc = 14 instr = 00000013 a0 = 0x00000001 a1 = 0x00000002   :   nop
 9  pc = 0c instr = 00150513 a0 = 0x00000001 a1 = 0x00000002   :   nop
10  pc = 10 instr = fea5dee3 a0 = 0x00000001 a1 = 0x00000002   :   addi  $10, $10, 0x00000001
11  pc = 14 instr = 00000013 a0 = 0x00000002 a1 = 0x00000002   :   bge   $11, $10, 0xfffffffc (-4)
12  pc = 14 instr = 00000013 a0 = 0x00000002 a1 = 0x00000002   :   nop
13  pc = 14 instr = 00000013 a0 = 0x00000002 a1 = 0x00000002   :   nop
14  pc = 14 instr = 00000013 a0 = 0x00000002 a1 = 0x00000002   :   nop
15  pc = 18 instr = 00000013 a0 = 0x00000002 a1 = 0x00000002   :   new/unknown
 The program has finished execution!
```

Как можно видеть, как только значение регистра a0 стало таким же, как и a1, то программа завершилась.

Пример теста команды func

Для тестирования данной функции были выбраны крайние точки, использованные во второй лабораторной работе, а именно. Числа: соответствующая команда в двоичном : в шестнадцатеричном : результат

- 0, 0 : 00000000000000000000010101111111 : 0000057F : 0x00000000
- 1, 1 : 00000000000100000001010101111111 : 0010157F : 0x00000002
- 2, 2 : 00000000001000000010010101111111 : 0020257F : 0x0000000c
- 15, 240 : 00000000111111110000010101111111 : 00FF057F : 0x00d2fe10
- 240, 15 : 00001111000000001111010101111111 : 0F00F57F : 0x00001b3f
- 85, 170 : 00000101010110101010010101111111 : 055AA57F : 0x004b2fda
- 170, 85 : 00001010101001010101010101111111 : 0AA5557F: 0x0009975f
- 255,255 :00001111111111111111110101111111 : 0FFFF57F : 0x00fe0100

```
  0  pc = xxxxxxxx instr = 00000013 a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   new/unknown
  1  pc = 00 instr = 0000057f a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   new/unknown
  2  pc = 04 instr = 0000157f a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   func   $10, $00000000, $00000000
  3  pc = 08 instr = 0010157f a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   nop
 .....................................................................
 38  pc = 08 instr = 0010157f a0 = 0xxxxxxxxx a1 = 0xxxxxxxxx   :   nop
 39  pc = 08 instr = 0010157f a0 = 0x00000000 a1 = 0xxxxxxxxx   :   func   $10, $00000001, $00000001
 40  pc = 0c instr = 0020257f a0 = 0x00000000 a1 = 0xxxxxxxxx   :   nop
 .....................................................................
 75  pc = 0c instr = 0020257f a0 = 0x00000000 a1 = 0xxxxxxxxx   :   nop
 76  pc = 0c instr = 0020257f a0 = 0x00000002 a1 = 0xxxxxxxxx   :   func   $10, $00000010, $00000010
 77  pc = 10 instr = 00ff057f a0 = 0x00000002 a1 = 0xxxxxxxxx   :   nop
 .....................................................................
112  pc = 10 instr = 00ff057f a0 = 0x00000002 a1 = 0xxxxxxxxx   :   nop
113  pc = 10 instr = 00ff057f a0 = 0x0000000c a1 = 0xxxxxxxxx   :   func   $10, $11110000, $00001111
114  pc = 14 instr = 0f00f57f a0 = 0x0000000c a1 = 0xxxxxxxxx   :   nop
 .....................................................................
149  pc = 14 instr = 0f00f57f a0 = 0x0000000c a1 = 0xxxxxxxxx   :   nop
150  pc = 14 instr = 0f00f57f a0 = 0x00d2fe10 a1 = 0xxxxxxxxx   :   func   $10, $00001111, $11110000
151  pc = 18 instr = 055aa57f a0 = 0x00d2fe10 a1 = 0xxxxxxxxx   :   nop
 .....................................................................
186  pc = 18 instr = 055aa57f a0 = 0x00d2fe10 a1 = 0xxxxxxxxx   :   nop
187  pc = 18 instr = 055aa57f a0 = 0x00001b3f a1 = 0xxxxxxxxx   :   func   $10, $10101010, $01010101
188  pc = 1c instr = 0aa5557f a0 = 0x00001b3f a1 = 0xxxxxxxxx   :   nop
 .....................................................................
223  pc = 1c instr = 0aa5557f a0 = 0x00001b3f a1 = 0xxxxxxxxx   :   nop
224  pc = 1c instr = 0aa5557f a0 = 0x004b2fda a1 = 0xxxxxxxxx   :   func   $10, $01010101, $10101010
225  pc = 20 instr = 0ffff57f a0 = 0x004b2fda a1 = 0xxxxxxxxx   :   nop
 .....................................................................
260  pc = 20 instr = 0ffff57f a0 = 0x004b2fda a1 = 0xxxxxxxxx   :   nop
261  pc = 20 instr = 0ffff57f a0 = 0x0009975f a1 = 0xxxxxxxxx   :   func   $10, $11111111, $11111111
262  pc = 24 instr = 00000013 a0 = 0x0009975f a1 = 0xxxxxxxxx   :   nop
 .....................................................................
297  pc = 24 instr = 00000013 a0 = 0x0009975f a1 = 0xxxxxxxxx   :   nop
298  pc = 24 instr = 00000013 a0 = 0x00fe0100 a1 = 0xxxxxxxxx   :   new/unknown
  The program has finished execution!
```

Все значение корректны!

Временные диаграммы

Setup

Worst Negative Slack (WNS):	inf
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	107

Hold

Worst Hold Slack (WHS):	inf
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	107

Pulse Width

Worst Pulse Width Slack (WPWS):	NA
Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

There are no user specified timing constraints.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	5	5	2	sm_clk_divider...r/q_reg[29]/C	clk	1000000000.000	1000000000.000	0.948	∞	
Path 2	∞	2	3	32	rst_n	sm_clk_divide.../q_reg[0]/CLR	1.264	0.082	1.181	∞	input port clock
Path 3	∞	2	3	32	rst_n	sm_clk_divide..._reg[10]/CLR	1.264	0.082	1.181	∞	input port clock
Path 4	∞	2	3	32	rst_n	sm_clk_divide..._reg[11]/CLR	1.264	0.082	1.181	∞	input port clock
Path 5	∞	2	3	32	rst_n	sm_clk_divide..._reg[12]/CLR	1.264	0.082	1.181	∞	input port clock
Path 6	∞	2	3	32	rst_n	sm_clk_divide..._reg[13]/CLR	1.264	0.082	1.181	∞	input port clock
Path 7	∞	2	3	32	rst_n	sm_clk_divide..._reg[14]/CLR	1.264	0.082	1.181	∞	input port clock
Path 8	∞	2	3	32	rst_n	sm_clk_divide..._reg[15]/CLR	1.264	0.082	1.181	∞	input port clock
Path 9	∞	2	3	32	rst_n	sm_clk_divide..._reg[16]/CLR	1.264	0.082	1.181	∞	input port clock
Path 10	∞	2	3	32	rst_n	sm_clk_divide..._reg[17]/CLR	1.264	0.082	1.181	∞	input port clock

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 11	∞	2	2	1	sm_clk_divide...tr/q_reg[8]/C	sm_clk_divider...r/q_reg[10]/D	0.058	0.000	0.058	-∞
Path 12	∞	2	2	1	sm_clk_divider...r/q_reg[11]/C	sm_clk_divider...r/q_reg[11]/D	0.058	0.000	0.058	-∞
Path 13	∞	3	3	1	sm_clk_divider...r/q_reg[11]/C	sm_clk_divider...r/q_reg[12]/D	0.058	0.000	0.058	-∞
Path 14	∞	2	2	1	sm_clk_divider...r/q_reg[12]/C	sm_clk_divider...r/q_reg[13]/D	0.058	0.000	0.058	-∞
Path 15	∞	2	2	1	sm_clk_divider...r/q_reg[12]/C	sm_clk_divider...r/q_reg[14]/D	0.058	0.000	0.058	-∞
Path 16	∞	2	2	1	sm_clk_divider...r/q_reg[15]/C	sm_clk_divider...r/q_reg[15]/D	0.058	0.000	0.058	-∞
Path 17	∞	3	3	1	sm_clk_divider...r/q_reg[15]/C	sm_clk_divider...r/q_reg[16]/D	0.058	0.000	0.058	-∞
Path 18	∞	3	3	1	sm_clk_divider...r/q_reg[15]/C	sm_clk_divider...r/q_reg[17]/D	0.058	0.000	0.058	-∞
Path 19	∞	3	3	1	sm_clk_divider...r/q_reg[15]/C	sm_clk_divider...r/q_reg[18]/D	0.058	0.000	0.058	-∞
Path 20	∞	3	3	1	sm_clk_divider...r/q_reg[15]/C	sm_clk_divider...r/q_reg[19]/D	0.058	0.000	0.058	-∞

Выводы по работе

В ходе выполнения данной лабораторной работы мной были сделаны следующие замечания (выводы):

- Реализованная мной архитектура (конвейер) позволила увеличить пропускную способность (количество команд в секунду) процессора, также пять команд теперь выполняются параллельно, из-за чего тактовая частота процессора может быть увеличена (в идеальном случае в пять раз).
- В ходе реализации конвейера я в качестве примера взял вариант из книги Харрис Харрис, однако его пришлось переделать
- После реализации я выделил ряд улучшений, которые можно сделать в ядре
    - Убрать последнюю стадию, и сделать конвейер трехстадийным
    - Разделить блок предотвращения конфликтов на несколько подблоков