



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ «Информатика и системы управления»

КАФЕДРА _____ «Компьютерные системы и сети (ИУ6)»

НАПРАВЛЕНИЕ ПОДГОТОВКИ _____ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 по курсу «Архитектура ЭВМ»

«Изучение принципов работы микропроцессорного ядра RISC-V»

Студент: ИУ7-53Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ Е. Н. Дубровин
(подпись, дата) (И. О. Фамилия)

Москва, 2021

Содержание

Введение	3
1 Основные теоретические сведения	4
2 Задание 1	5
2.1 Результаты выполнения	5
2.2 Вывод	7
3 Задание 2	8
3.1 Результаты выполнения	8
3.2 Вывод	8
4 Задание 3	9
4.1 Результаты выполнения	9
4.2 Вывод	9
5 Задание 4	10
5.1 Результаты выполнения	10
5.2 Вывод	10
6 Задание 5	11
6.1 Проверка результата	11
6.2 Временные диаграммы сигналов стадий выполнения	11
6.3 Трасса выполнения программы	12
6.4 Оптимизация программы	13
6.5 Вывод	16
Заключение	18

Введение

Целью данной работы является ознакомление с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров, а также знакомство с принципами проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС.

1 Основные теоретические сведения

Для достижения поставленных целей в настоящей лабораторной работе используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V.

RISC-V является открытым современным набором команд, который может использоваться для построения как микроконтроллеров, так и высокопроизводительных микропроцессоров. В связи с такой широкой областью применения в систему команд введена вариативность. Таким образом, термин RISC-V фактически является названием для семейства различных систем команд, которые строятся вокруг базового набора команд, путем внесения в него различных расширений.

В данной работе исследуется набор команд RV32I, который включает в себя основные команды 32-битной целочисленной арифметики кроме умножения и деления. В рамках данного набора команд мы не будем рассматривать системные команды, связанные с таймерами, системными регистрами, управлением привилегиями, прерываниями и исключениями.

2 Задание 1

Дизассемблирование программы по индивидуальному варианту.

2.1 Результаты выполнения

На листинге 2.1 приведен текст программы по индивидуальному варианту.

Листинг 2.1 – Текст программы по индивидуальному варианту

```
1 # ВАРИАНТ 12
2     .section .text
3     .globl _start;
4     len = 8 #Размер массива
5     enroll = 4 #Количество обрабатываемых элементов за одну итерацию
6     elem_sz = 4 #Размер одного элемента массива
7
8 _start:
9     la x1, _x
10    addi x20, x1, elem_sz*len #Адрес последнего элемента
11 lp:
12    lw x2, 0(x1)
13    lw x3, 4(x1)
14    add x31, x31, x2 #!
15    add x31, x31, x3
16    lw x4, 8(x1)
17    lw x5, 12(x1)
18    add x31, x31, x4
19    add x31, x31, x5
20    addi x1, x1, elem_sz*enroll
21    bne x1, x20, lp
22    addi x31, x31, 1
23 lp2: j lp2
24
25     .section .data
26 _x:   .4byte 0x1
27       .4byte 0x2
28       .4byte 0x3
29       .4byte 0x4
30       .4byte 0x5
31       .4byte 0x6
32       .4byte 0x7
33       .4byte 0x8
```

На листиге 2.2 приведен псевдокод на языке C, соответствующий про-

грамме варианта.

Листинг 2.2 – Псевдокод на языке C программы варианта

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4
5 int _x[] = {1, 2, 3, 4, 5, 6, 7, 8};
6
7 void _start()
8 {
9     int *x1 = _x;
10    int *x20 = x1 + len;
11    int x31 = 0;
12
13    do
14    {
15        int x2 = x1[0];
16        int x3 = x1[1];
17        x31 += x2;
18        x31 += x3;
19        int x4 = x1[2];
20        int x5 = x1[3];
21        x31 += x4;
22        x31 += x5;
23        x1 += enroll;
24    } while(x1 != x20);
25
26    x31++;
27
28    while(1){}
29 }
```

На листиге 2.3 приведен дизассемблерный код.

Листинг 2.3 – Дизассемблерный листинг

```
1 80000000 <_start>:
2 80000000:      00000097          auipc    x1,0x0
3 80000004:      03c08093          addi     x1,x1,60 # 8000003c <_x>
4 80000008:      02008a13          addi     x20,x1,32
5
6 8000000c <lp>:
7 8000000c:      0000a103          lw       x2,0(x1)
8 80000010:      0040a183          lw       x3,4(x1)
9 80000014:      002f8fb3          add      x31,x31,x2
10 80000018:      003f8fb3          add      x31,x31,x3
11 8000001c:      0080a203          lw       x4,8(x1)
12 80000020:      00c0a283          lw       x5,12(x1)
13 80000024:      004f8fb3          add      x31,x31,x4
14 80000028:      005f8fb3          add      x31,x31,x5
15 8000002c:      01008093          addi     x1,x1,16
16 80000030:      fd409ee3          bne      x1,x20,8000000c <lp>
17 80000034:      001f8f93          addi     x31,x31,1
18
19 80000038 <lp2>:
20 80000038:      0000006f          jal      x0,80000038 <lp2>
```

2.2 Вывод

Проанализировав исходный текст программы можно сделать вывод, что в регистре x31 в конце выполнения программы должна содержаться сумма элементов массива + 1 (37).

3 Задание 2

Получение снимка экрана, содержащего временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 8000000с на 2-ой итерации.

3.1 Результаты выполнения

На рисунке 3.1 представлены временные диаграммы выполнения стадий выборки и диспетчеризации необходимой команды.

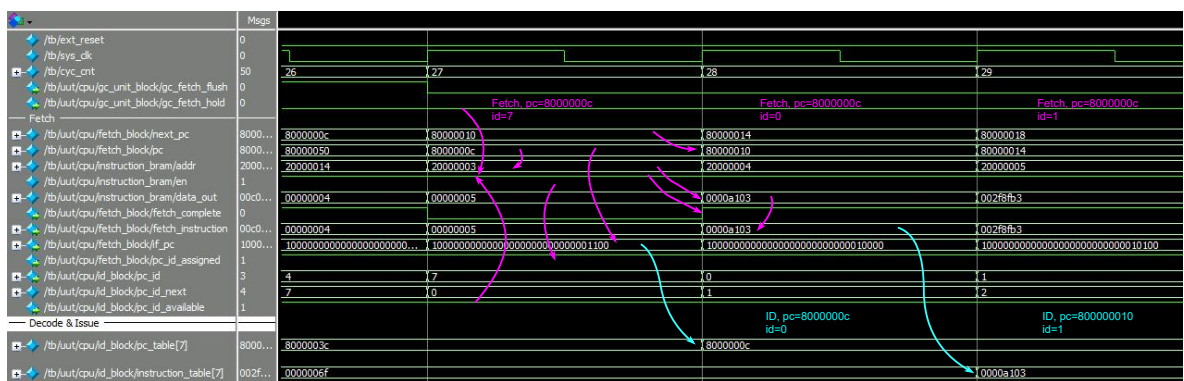


Рисунок 3.1 – Выборка и диспетчеризация команды с адресом 8000000с на второй итерации

3.2 Вывод

Выборка и диспетчеризация данной команды происходят в 27 и 28 тактах соответственно.

4 Задание 3

Получение снимка экрана, содержащего временную диаграмму выполнения стадий декодирования и планирования на выполнение команды с адресом 80000018 на 2-ой итерации.

4.1 Результаты выполнения

На рисунке 4.1 представлены временные диаграммы выполнения стадий декодирования и планирования на выполнение необходимой команды.

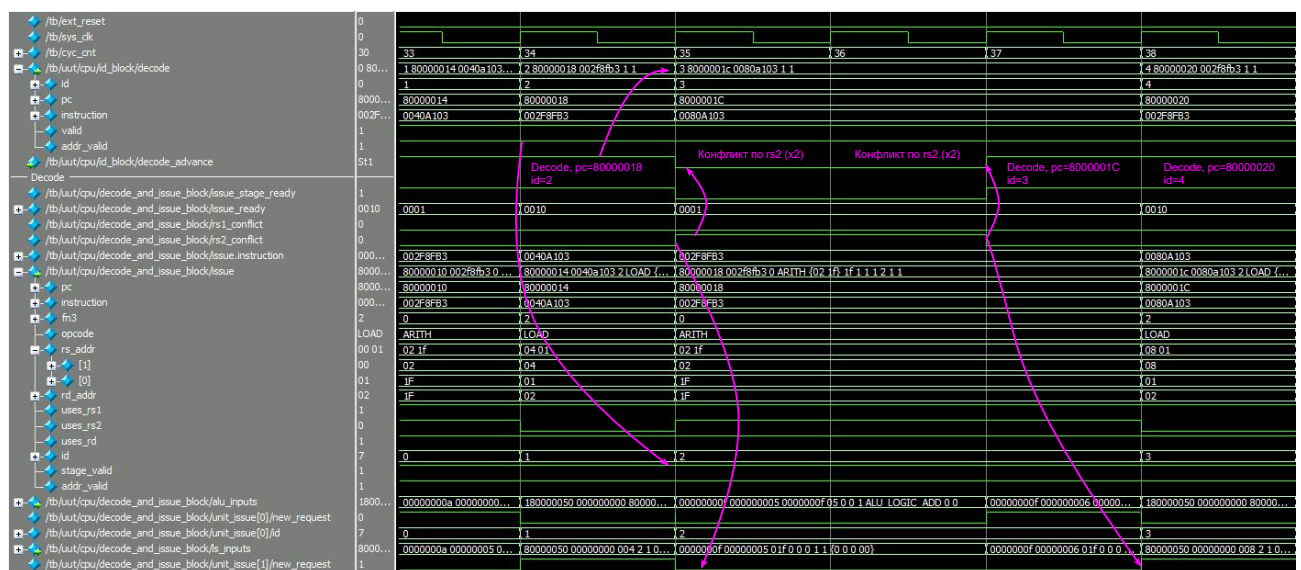


Рисунок 4.1 – Декодирование и планирование на выполнение команды с адресом 80000018 на второй итерации

4.2 Вывод

Декодирование данной команды происходит на 34 такте, однако так как представленная команда является командой загрузки из памяти, а следующая команда использует загруженное значение, возникает конфликт по rs2 (x2) и декодирование следующей команды происходит только на 37 такте.

5 Задание 4

Получение снимка экрана, содержащего временную диаграмму выполнения стадии выполнения команды с адресом 8000002с на 1-ой итерации.

5.1 Результаты выполнения

На рисунке 5.1 представлены временные диаграммы выполнения стадии выполнения необходимой команды.

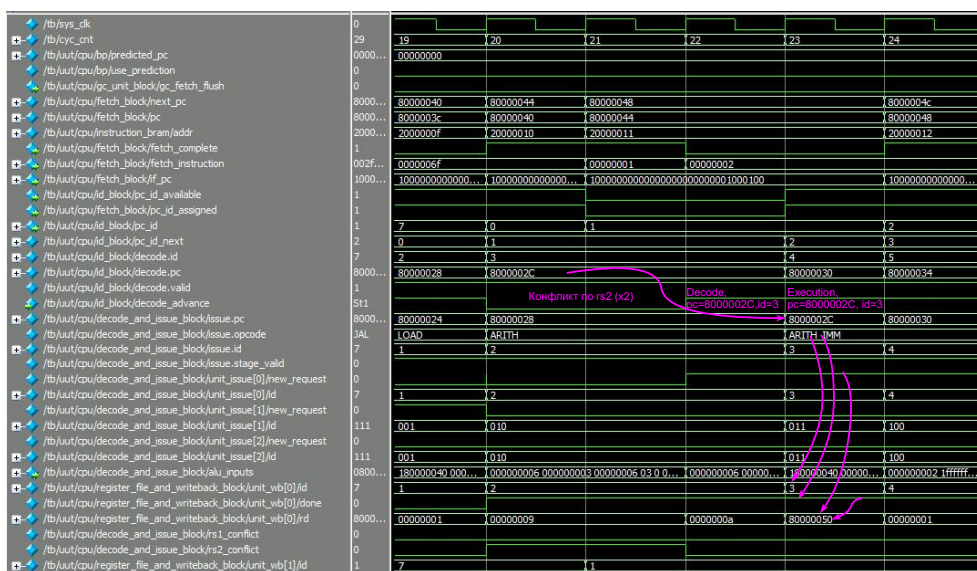


Рисунок 5.1 – Выполнение команды с адресом 8000002с на первой итерации

5.2 Вывод

Представленная команда является командой обработки данных и выполняется на АЛУ за 1 такт, а именно на 23-ем такте, при этом, так как предыдущая команда является командой загрузки из памяти, а результат загрузки используется в данной команде, 2 такта происходит ожидание завершения выполнения команды загрузки блоком обращения к памяти.

6 Задание 5

6.1 Проверка резултата

Значение регистра x31 представлено на рисунке 6.1. Значение равно 25 в шестнадцатиричной системе, что соответствует значению 37 в десятиричной системе счисления, приведенном в задании 1.

+  /tb/cyc_cnt	43	40
+  /tb/register_file[31]	00000025	00000025
+  /tb/uut/cpu/decode_and_issue_block/issue.pc	80000040	80000038

Рисунок 6.1 – Проверка значения регистра x31

6.2 Временные диаграммы сигналов стадий выполнения

В данном подразделе представлены временные диаграммы сигналов стадий выполнения сигналов команды `add x31, x31, x2`, находящейся по адресу `0x80000014` (на первой итерации).

На рисунке 6.2 представлены временные диаграммы сигналов стадий выборки и диспетчеризации. На рисунке 6.3 представлены временные диаграммы сигналов стадий декодирования и планирования на выполнение. На рисунке 6.4 представлены временные диаграммы сигналов стадии выполнения.

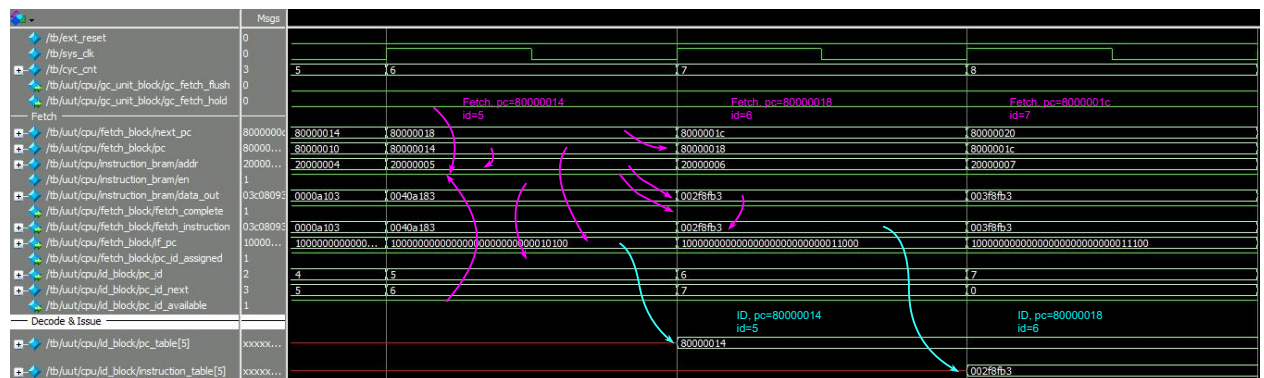
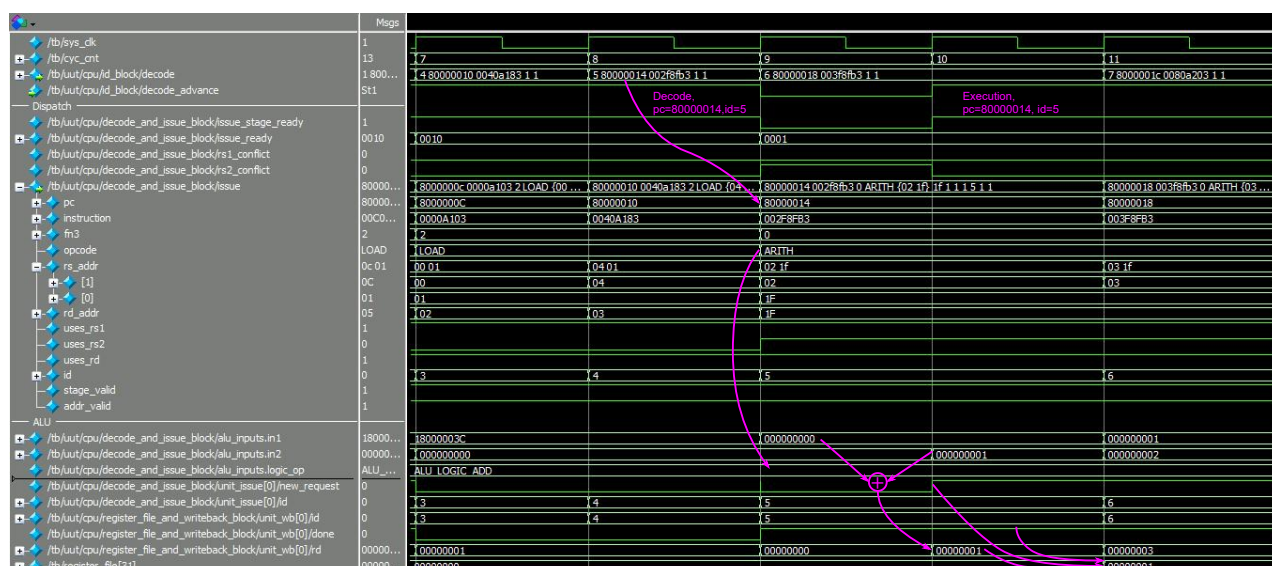
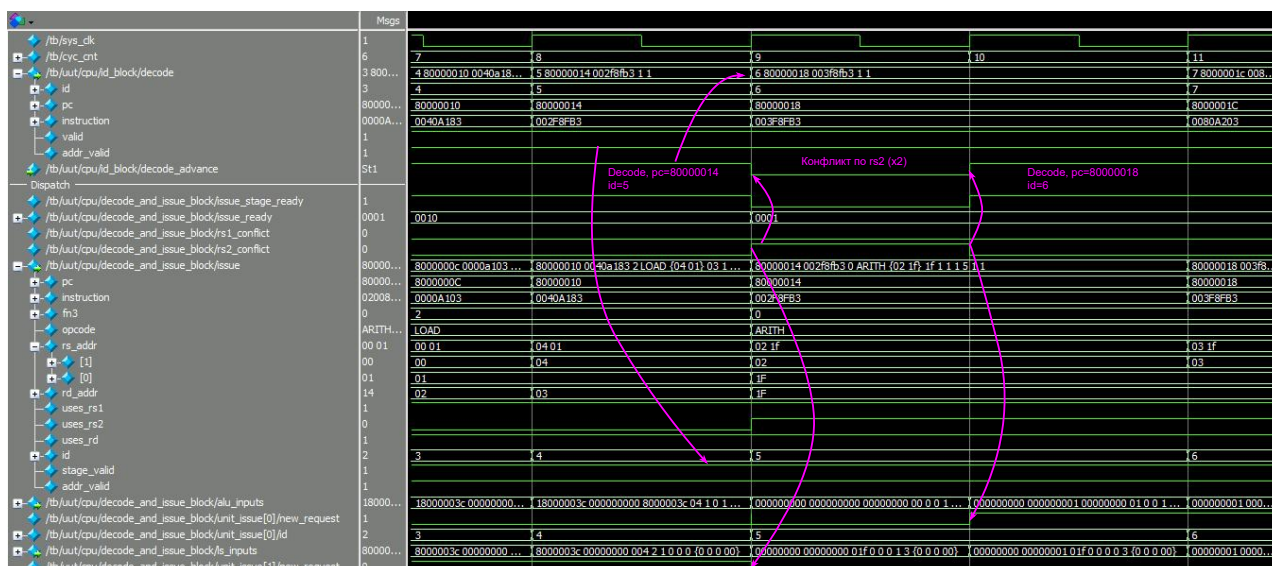


Рисунок 6.2 – Выборка и диспетчеризация команды #!



6.3 Трасса выполнения программы

На рисунке 6.5 представлена трасса выполнения программы.

Проанализировав трассу выполнения программы, можно сделать вывод о том, что конфликты возникают тогда, когда блок обращения к памяти не успевает загрузить необходимое для выполнения арифметической операции значение. Это происходит потому, что блок обращения к памяти выполняет загрузку за три такта, а арифметическая команда следует за загрузкой нужной переменной через одну команду, таким образом блок обращения к памяти завершает только 2 такта обработки.

[illegible]

6.4 Оптимизация программы

На листинге 6.1 приведен текст оптимизированной программы по индивидуальному варианту.

Листинг 6.1 – Текст оптимизированной программы по индивидуальному варианту

```
1 # ВАРИАНТ 12 (оптимизация)
2     .section .text
3     .globl _start;
4     len = 8 #Размер массива
5     enroll = 4 #Количество обрабатываемых элементов за одну итерацию
6     elem_sz = 4 #Размер одного элемента массива
7
8 _start:
9     la x1, _x
10    addi x20, x1, elem_sz*len #Адрес последнего элемента
11 lp:
12    lw x2, 0(x1)
13    lw x3, 4(x1)
14    lw x4, 8(x1)
15    lw x5, 12(x1)
16    add x31, x31, x2 #!
17    add x31, x31, x3
18    add x31, x31, x4
19    add x31, x31, x5
20    addi x1, x1, elem_sz*enroll
21    bne x1, x20, lp
22    addi x31, x31, 1
23 lp2: j lp2
24
25     .section .data
26 _x:   .4byte 0x1
27       .4byte 0x2
28       .4byte 0x3
29       .4byte 0x4
30       .4byte 0x5
31       .4byte 0x6
32       .4byte 0x7
33       .4byte 0x8
```

На листиге 6.2 приведен псевдокод на языке C, соответствующий оптимизированной программе варианта.

Листинг 6.2 – Псевдокод на языке C оптимизированной программы варианта

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4
5 int _x[] = {1, 2, 3, 4, 5, 6, 7, 8};
6
7 void _start()
8 {
9     int *x1 = _x;
10    int *x20 = x1 + len;
11    int x31 = 0;
12
13    do
14    {
15        int x2 = x1[0];
16        int x3 = x1[1];
17        int x4 = x1[2];
18        int x5 = x1[3];
19        x31 += x2;
20        x31 += x3;
21        x31 += x4;
22        x31 += x5;
23        x1 += enroll;
24    } while(x1 != x20);
25
26    x31++;
27
28    while(1){}
29 }
```

На листиге 6.3 приведен дизассемблерный код оптимизированной программы.

Листинг 6.3 – Дизассемблерный листинг

```
1 80000000 <_start>:
2 80000000:      00000097          auipc    x1,0x0
3 80000004:      03c08093          addi     x1,x1,60 # 8000003c <_x>
4 80000008:      02008a13          addi     x20,x1,32
5
6 8000000c <lp>:
7 8000000c:      0000a103          lw       x2,0(x1)
8 80000010:      0040a183          lw       x3,4(x1)
9 80000014:      0080a203          lw       x4,8(x1)
10 80000018:      00c0a283          lw       x5,12(x1)
11 8000001c:      002f8fb3          add      x31,x31,x2
12 80000020:      003f8fb3          add      x31,x31,x3
13 80000024:      004f8fb3          add      x31,x31,x4
14 80000028:      005f8fb3          add      x31,x31,x5
15 8000002c:      01008093          addi     x1,x1,16
16 80000030:      fd409ee3          bne      x1,x20,8000000c <lp>
17 80000034:      001f8f93          addi     x31,x31,1
18
19 80000038 <lp2>:
20 80000038:      0000006f          jal      x0,80000038 <lp2>
```

На рисунке 6.6 представлена трасса выполнения оптимизированной программы.

6.5 Вывод

При выполнении неоптимизированной программы вычисления завершаются за 41 такт, при выполнении оптимизированной – за 37 (без учета бесконечного цикла). Таким образом, в результате оптимизации удалось сократить выполнение на 4 такта, то есть получилось ускорить программу на $\frac{4}{41} \approx 0.1(10\%)$

Заключение

В ходе лабораторной работы были изучены принципы функционирования и построения, а также особенности архитектуры суперскалярных конвейерных микропроцессоров на примере микропроцессорного ядра Taiga, реализующего систему команд семейства RISC-V. Таким образом, цель данной работы была достигнута.