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 «Программное обеспечение ЭВМ и информационные технологии»(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 «Программная инженерия»

О Т Ч Е Т
по лабораторной работе № 2

Название Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина Архитектура элекронно-вычислительных машин

Студент: _____ Козлова И. В.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Попов А. Ю.
подпись, дата Фамилия, И. О.

Москва — 2021 г.

Цель работы

Основной целью работы является ознакомление с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров. Дополнительной целью работы является знакомство с принципами проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС.

Основные теоретические сведения

RISC-V является открытым современным набором команд, который может использоваться для построения как микроконтроллеров, так и высокопроизводительных микропроцессоров. 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.

Модель памяти

Архитектура RV32I предполагает плоское линейное 32-х битное адресное пространство. Минимальной адресуемой единицей информации является 1 байт. Используется порядок байтов от младшего к старшему (Little Endian), то есть, младший байт 32-х битного слова находится по младшему адресу (по смещению 0). Отсутствует разделение на адресные пространства команд, данных и ввода-вывода. Распределение областей памяти между различными устройствами (ОЗУ, ПЗУ, устройства ввода-вывода) определяется реализацией.

Система команд

Большая часть команд RV32I является трехадресными, выполняющими операции над двумя заданными явно операндами, и сохраняющими результат в регистре. Операндами могут являться регистры или константы, явно заданные в коде команды. Операнды всех команд задаются явно.

Архитектура RV32I, как и большая часть RISC-архитектур, предполагает разделение команд на команды доступа к памяти (чтение данных из памяти в регистр или запись данных из регистра в память) и команды обработки данных в регистрах.

Общая программа для всех вариантов

Исследуемая программа

Код программы представлен в листинте 1

Листинг 1 – Код программы для всех вариантов

```
1     .section .text
2     .globl _start;
3     len = 8
4     enroll = 4
5     elem_sz = 4
6 _start:
7     addi x20, x0, len/enroll
8     la x1, _x
9 loop:
10    lw x2, 0(x1)
11    add x31, x31, x2
12    lw x2, 4(x1)
13    add x31, x31, x2
14    lw x2, 8(x1)
15    add x31, x31, x2
16    lw x2, 12(x1)
17    add x31, x31, x2
18    addi x1, x1, elem_sz*enroll
19    addi x20, x20, -1
20    bne x20, x0, loop
21    addi x31, x31, 1
22 forever: j forever
23
24    .section .data
25 _x: .4byte 0x1
26    .4byte 0x2
27    .4byte 0x3
28    .4byte 0x4
29    .4byte 0x5
30    .4byte 0x6
31    .4byte 0x7
32    .4byte 0x8
```

Дизассемблерный код представлен на листинге 2.

Листинг 2 – Дизассемблированный код общей программы

```
1  Disassembly of section .text:
2
3  80000000 <_start>:
4  80000000:       00200a13        addi    x20,x0,2
5  80000004:       00000097        auipc   x1,0x0
6  80000008:       03c08093        addi    x1,x1,60 # 80000040 <_x>
7
8  8000000c <lp>:
9  8000000c:       0000a103        lw      x2,0(x1)
10 80000010:       002f8fb3        add     x31,x31,x2
11 80000014:       0040a183        lw      x3,4(x1)
12 80000018:       003f8fb3        add     x31,x31,x3
13 8000001c:       0080a203        lw      x4,8(x1)
14 80000020:       00c0a283        lw      x5,12(x1)
15 80000024:       004f8fb3        add     x31,x31,x4
16 80000028:       005f8fb3        add     x31,x31,x5
17 8000002c:       01008093        addi    x1,x1,16
18 80000030:       fffa0a13        addi    x20,x20,-1
19 80000034:       fc0a1ce3        bne     x20,x0,8000000c <lp>
20 80000038:       001f8f93        addi    x31,x31,1
21
22 8000003c <lp2>:
23 8000003c:       0000006f        jal     x0,8000003c <lp2>
```

Можно сказать, что данная программа эквивалентна следующему псевдокоду на языке C, представленному на листинге 3.

Листинг 3 – Псевдокод общей программы

```
#define len 8
#define enroll 4
#define elem_sz 4
int _x[]={1,2,3,4,5,6,7,8};
void _start() {
    int x20 = len/enroll;
    int *x1 = _x;

    do {
        int x2 = x1[0];
        x31 += x2;
        x2 = x1[1];
        x31 += x2;
        x2 = x1[2];
        x31 += x2;
        x2 = x1[3];
        x31 += x2;
        x1 += enroll;
        x20--;
    } while(x20 != 0);
    x31++;
    while(1){}
}
```

Результаты исследования программы

Задания выполнялись по вараинту 9.

Задание №2

Получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 8000002c на первой итерации.

Результат представлен на рисунке 1

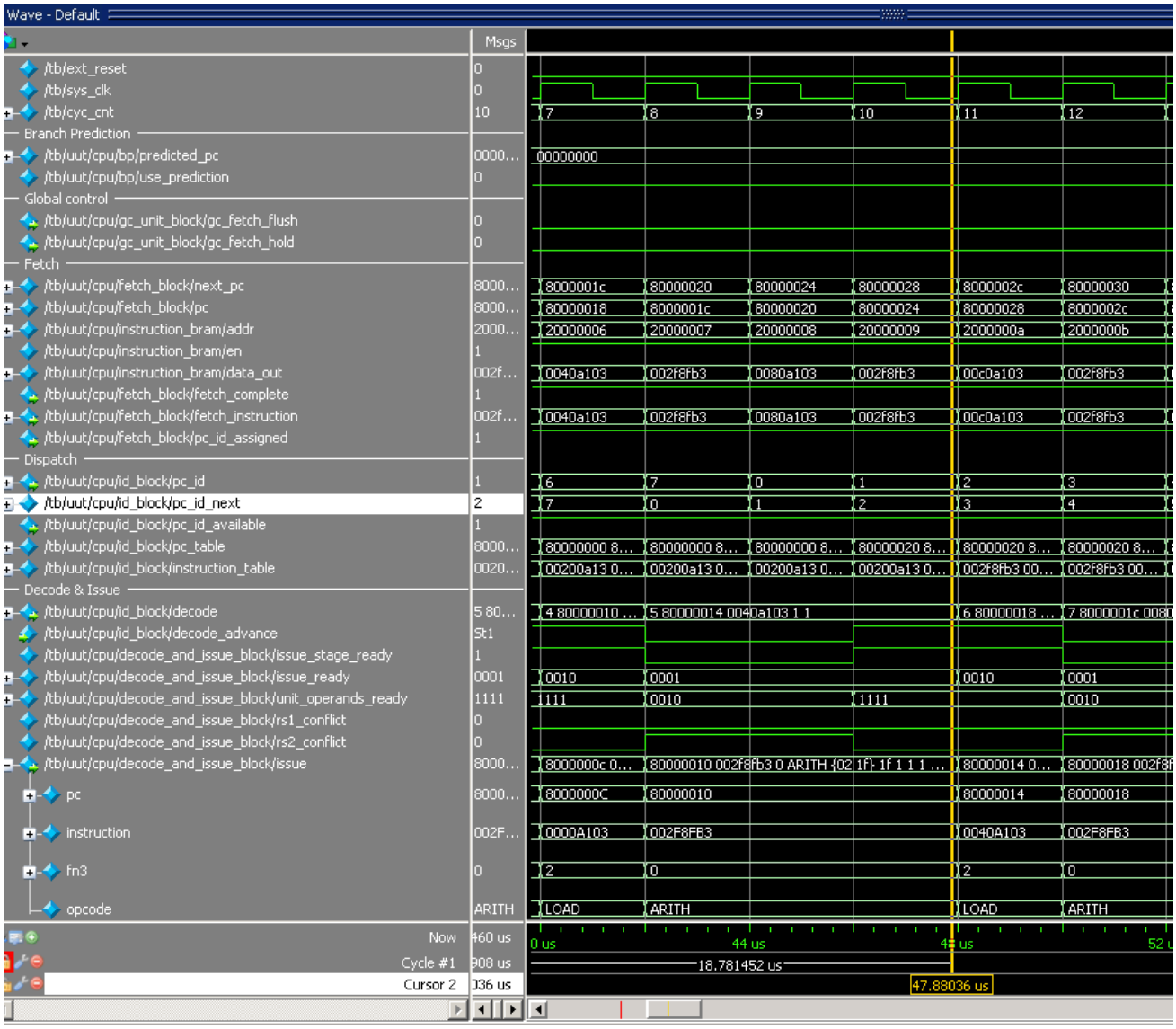

Рисунок 1 – Задание №2

Задание №3

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 8000000c на второй итерации.

Результат представлен на рисунке 2

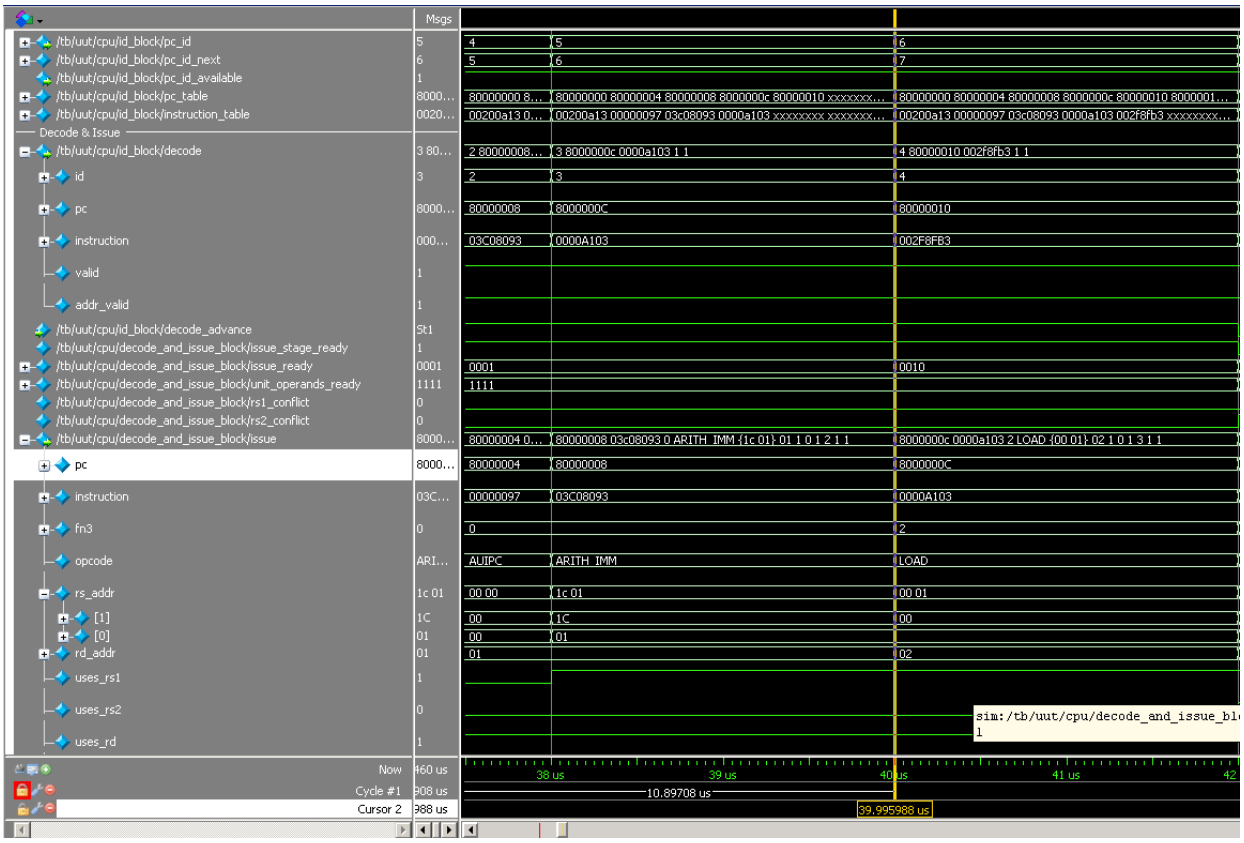

Рисунок 2 – Задание №3

Задание №4

Получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды с адресом 80000020 на первой итерации.

Результат представлен на рисунке 3

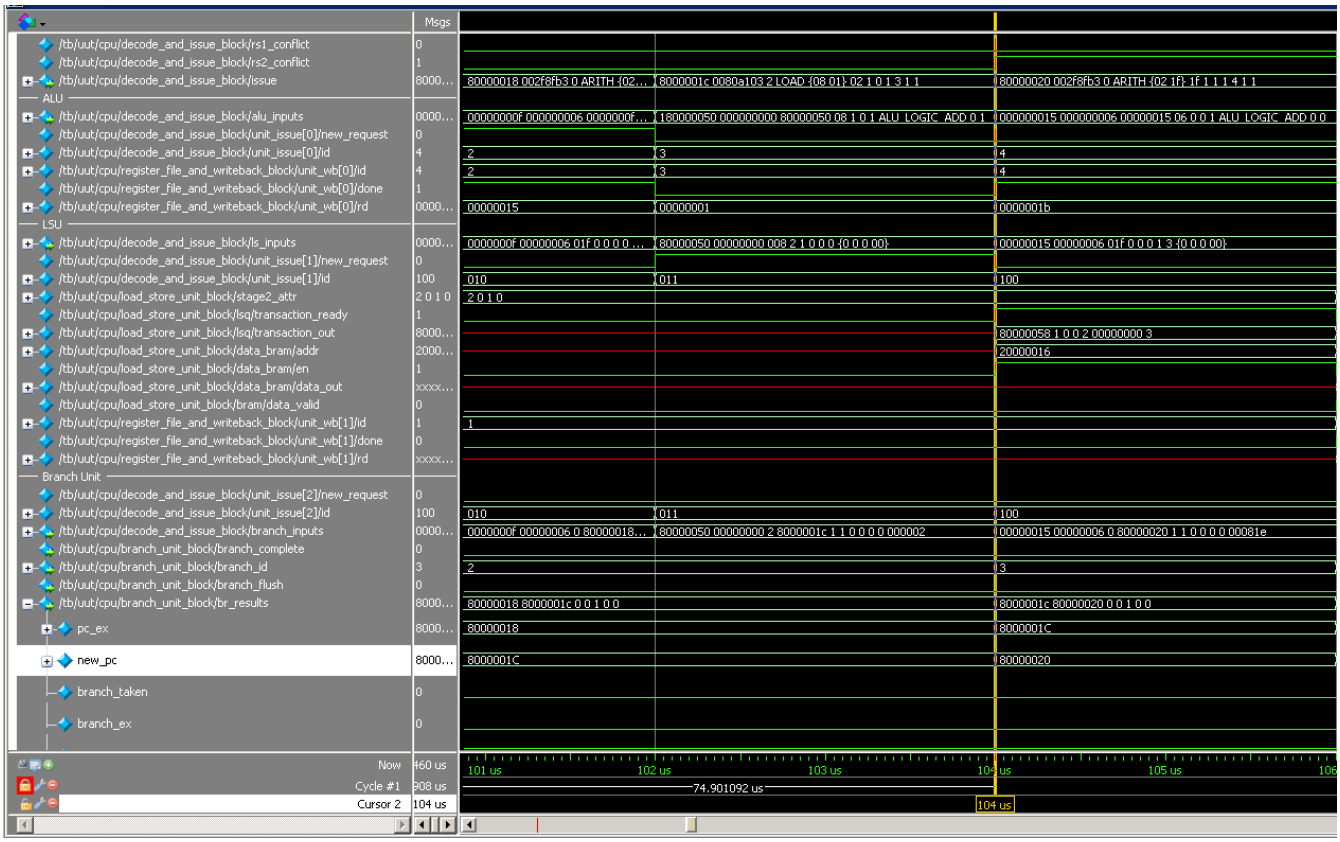

Рисунок 3 – Задание №4

Программа по варианту

Исследуемая программа

Код программы представлен в листинте 4

Листинг 4 – Код программы 9 варианта

```
1         .section .text
2 .globl _start;
3 len = 8
4 enroll = 4
5 elem_sz = 4
6
7 _start:
8     addi x20, x0, len/enroll
9     la x1, _x
10 lp:
11     lw x2, 0(x1)
12     add x31, x31, x2 #!
13     lw x3, 4(x1)
14     add x31, x31, x3
15     lw x4, 8(x1)
16     lw x5, 12(x1)
17     add x31, x31, x4
18     add x31, x31, x5
19     addi x1, x1, elem_sz*enroll
20     addi x20, x20, -1
21     bne x20, x0, 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
```

Дизассемблерный код представлен на листинге 5.

Листинг 5 – Дизассемблированный код 9 варианта

```
1  Disassembly of section .text:
2
3  80000000 <_start>:
4  80000000:       00200a13        addi    x20,x0,2
5  80000004:       00000097        auipc   x1,0x0
6  80000008:       03c08093        addi    x1,x1,60 # 80000040
       <_x>
7
8  8000000c <lp>:
9  8000000c:       0000a103        lw      x2,0(x1)
10 80000010:       0040a183        add     x31,x31,x2
11 80000014:       0080a203        lw      x4,8(x1)
12 80000018:       00c0a283        add     x31,x31,x3
13 8000001c:       002f8fb3        lw      x3,4(x1)
14 80000020:       003f8fb3        add     x31,x31,x4
15 80000024:       004f8fb3        lw      x5,12(x1)
16 80000028:       005f8fb3        add     x31,x31,x5
17 8000002c:       01008093        addi    x1,x1,16
18 80000030:       fffa0a13        addi    x20,x20,-1
19 80000034:       fc0a1ce3        bne     x20,x0,8000000c <lp>
20 80000038:       001f8f93        addi    x31,x31,1
21
22 8000003c <lp2>:
23 8000003c:       0000006f        jal     x0,8000003c <lp2>
```

Можно сказать, что данная программа эквивалентна следующему псевдокоду на языке C, представленному на листинге 6.

Листинг 6 – Псевдокод программы 9 варианта

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

Трасса работы программы

Трасса работы представлена на рисунке 4.

Рисунок 4 – Трасса выполнения программы

Временные диаграммы

Временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом #! (add x31, x31, x2) представлены на рисунке 5.

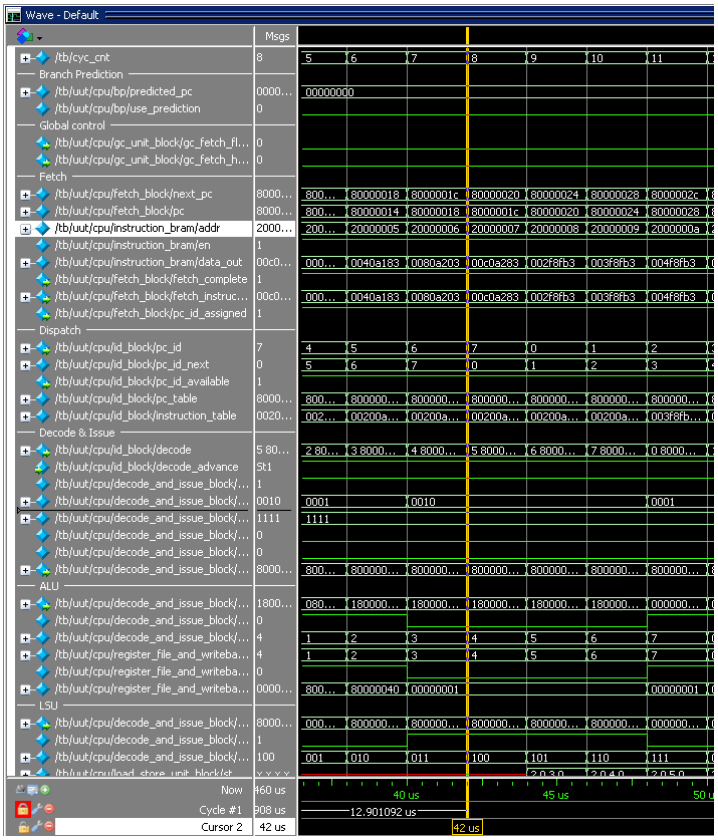

Рисунок 5 – Временные диаграммы сигналов

Вывод и предложение по оптимизации

Как видно на трассе работы программы, представленой на рисунке 4, конфликты возникают из-за того, что данные загружаются в память тогда, когда уже готова выполниться операция сложения тех данных, которые загружаются. Из-за этого и возникают конфликты, так как нечего складывать, так как в памяти пока нет ничего.

Можно заметить, что трижды возникает ситуация ошибочной выборки, которая негативно сказывается на производительности, так как приводит к очистки конвейера.

Оптимизировать программы можно тем, что сначала загрузить все данные в память, а потом их складывать. Тем самым у нас не будет конфликтов, не будет ожидания конца загрузки данных в память.

В итоге, можно будет уменьшить программу на 2 такта 4 раза в программе, на 1 такт 3 раза в программе, то есть на $11/53 = 20\%$ программа будет работать быстрее.

Оптимизированная программа

Код программы представлен в листинте 7

Листинг 7 – Код программы 9 варианта(оптимизированный)

```
1           .section .text
2 .globl _start;
3 len = 8
4 enroll = 4
5 elem_sz = 4
6
7 _start:
8     addi x20, x0, len/enroll
9     la x1, _x
10 lp:
11     lw x2, 0(x1)
12     lw x3, 4(x1)
13     lw x4, 8(x1)
14     lw x5, 12(x1)
15     add x31, x31, x2 #!
16     add x31, x31, x3
17     add x31, x31, x4
18     add x31, x31, x5
19     addi x1, x1, elem_sz*enroll
20     addi x20, x20, -1
21     bne x20, x0, 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
```

Дизассемблерный код представлен на листинге 8.

Листинг 8 – Дизассемблированный код 9 варинта (оптимизированный)

```
Disassembly of section .text:

80000000 <_start>:
80000000:       00200a13        addi    x20,x0,2
80000004:       00000097        auipc   x1,0x0
80000008:       03c08093        addi    x1,x1,60 # 80000040 <_x>

8000000c <lp>:
8000000c:       0000a103        lw      x2,0(x1)
80000010:       0040a183        lw      x3,4(x1)
80000014:       0080a203        lw      x4,8(x1)
80000018:       00c0a283        lw      x5,12(x1)
8000001c:       002f8fb3        add     x31,x31,x2
80000020:       003f8fb3        add     x31,x31,x3
80000024:       004f8fb3        add     x31,x31,x4
80000028:       005f8fb3        add     x31,x31,x5
8000002c:       01008093        addi    x1,x1,16
80000030:       fffa0a13        addi    x20,x20,-1
80000034:       fc0a1ce3        bne     x20,x0,8000000c <lp>
80000038:       001f8f93        addi    x31,x31,1

8000003c <lp2>:
8000003c:       0000006f        jal     x0,8000003c <lp2>
```

Можно сказать, что данная программа эквивалентна следующему псевдокоду на языке C, представленному на листинге 9.

Листинг 9 – Псевдокод программы 9 варинта (оптимизированный)

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

Трасса работы программы

Трасса работы представлена на рисунке 6.

Адрес	Код команды	Команда	id
80000000<_start>	00200a13	addi x20,x0,2	0
80000004	00000097	auipc x1,0x0	1
80000008	03c08093	addi x1,x1,60#80000040<x>	2
8000000c<loop>	0000a103	lw x2,0(x1)	3
80000010	0040a183	lw x3,4(x1)	4
80000014	0080a203	lw x4,8(x1)	5
80000018	00c0a283	lw x5,12(x1)	6
8000001c	002f8fb3	add x31,x31,x2	7
80000020	003f8fb3	add x31,x31,x3	0
80000024	004f8fb3	add x31,x31,x4	1
80000028	005f8fb3	add x31,x31,x5	2
8000002c	01008093	addi x1,x1,16	3
80000030	fff20a13	addi x20,x20,-1	4
80000034	fc0a1ce3	bne x20,x0,8000000c <lp>	5
8000000c<loop>	0000a103	lw x2,0(x1)	6
80000010	0040a183	lw x3,4(x1)	7
80000014	0080a203	lw x4,8(x1)	0
80000018	00c0a283	lw x5,12(x1)	1
8000001c	002f8fb3	add x31,x31,x2	2
80000020	003f8fb3	add x31,x31,x3	3
80000024	004f8fb3	add x31,x31,x4	4
80000028	005f8fb3	add x31,x31,x5	5
8000002c	01008093	addi x1,x1,16	6
80000030	fff20a13	addi x20,x20,-1	7
80000034	fc0a1ce3	bne x20,x0,8000000c <lp>	0
80000038	001f8f93	addi x31,x31,1	1
8000003c	fc0a1ce3	bne x20,x0,80000000c <lp>	2
80000040	00000001	<invalid operation>	3
80000044	00000002	<invalid operation>	4
8000000c<loop>	0000a103	lw x2,0(x1)	5
80000010	0040a183	lw x3,4(x1)	6
80000014	0080a203	lw x4,8(x1)	7
80000018	00c0a283	lw x5,12(x1)	0
8000001c	002f8fb3	add x31,x31,x2	1
80000020	003f8fb3	add x31,x31,x3	2
80000024	004f8fb3	add x31,x31,x4	3
80000028	005f8fb3	add x31,x31,x5	4
8000002c	01008093	addi x1,x1,16	5
80000030	fff20a13	addi x20,x20,-1	6
80000034	fc0a1ce3	bne x20,x0,8000000c <lp>	7
80000038	001f8f93	addi x31,x31,1	0
8000003c	fc0a1ce3	bne x20,x0,80000000c <lp>	1
80000040	00000001	<invalid operation>	2
80000044	00000002	<invalid operation>	3
80000048	00000003	<invalid operation>	4
8000004c	00000004	<invalid operation>	5
80000050	0000006f	jal x0,8000003c <lp2>	6

Номер такта: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

Рисунок 6 – Трасса выполнения программы

Вывод

В результате выполнения лабораторной работы были изучены принципы функционирования, построения и особенности архитектуры суперскалярных конвейерных микропроцессоров.

Также были рассмотрены принципы проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС.

На основе изученных материалов был найден способ оптимизации программы.

Поставленная цель достигнута.