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»

ОТЧЕТ
по лабораторной работе № 2

Название Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина Архитектура элекронно-вычислительных машин

Студент: _____ Ву Хай Данг
подпись, дата Фамилия, И.О.

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

Москва — 2023 г.

Содержание

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.

1 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.

1.1 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.

1.2 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.

2 Индивидуальные варианты

Данная программа выполняет найдение максимальный элемент. Листинг 2.1.

Листинг 2.1 – Пример программы

```
1     .section .text
2     .globl _start;
3     len = 9 #Размер массива
4     enroll = 2 #Количество обрабатываемых элементов за одну итерацию
5     elem_sz = 4 #Размер одного элемента массива
6
7     _start:
8     la x1, _x
9     addi x20, x0, (len-1)/enroll
10    lw x31, 0(x1)
11    addi x1, x1, elem_sz*1
12    lp:
13    lw x2, 0(x1)
14    lw x3, 4(x1)
15    bltu x2, x31, lt1
16    add x31, x0, x2 #!
17    lt1:    bltu x3, x31, lt2
18    add x31, x0, x3
19    lt2:
20    add x1, x1, elem_sz*enroll
21    addi x20, x20, -1
22    bne x20, x0, lp
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
34    .4byte 0x9
```

Дизассемблерный код представлен на листинге 2.2.

Листинг 2.2 – Дизассемблированный код примера 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 2.3.

Листинг 2.3 – Псевдокод общей программы

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

3 Результаты исследования программы

3.1 Задание №2

На рисунке 3.1 снимок экране симуляции в среде Modelsim на стадии выборки и диспетчеризации команды с адресом 80000028 на 2-й итерации.

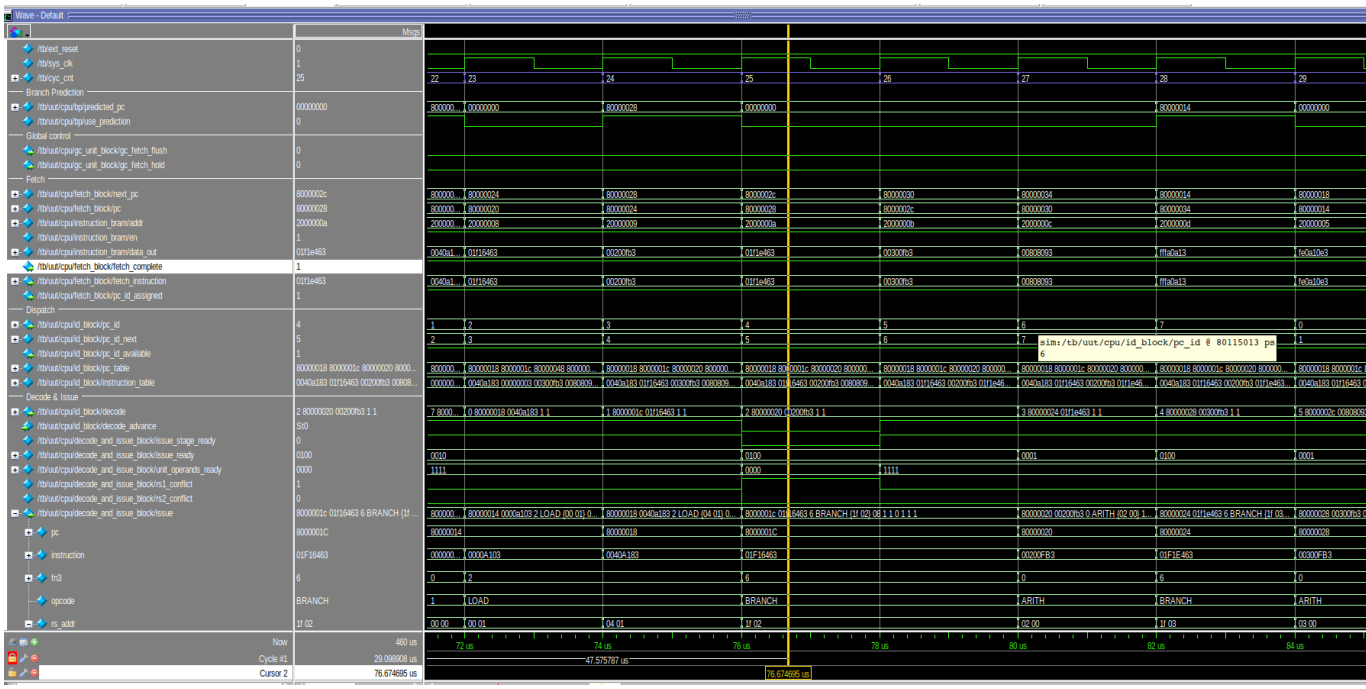

Рисунок 3.1 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 80000028 на второй итерации.

28

3.2 Задание №3

Условие задания

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 80000034 на второй итерации.

Результаты выполнения

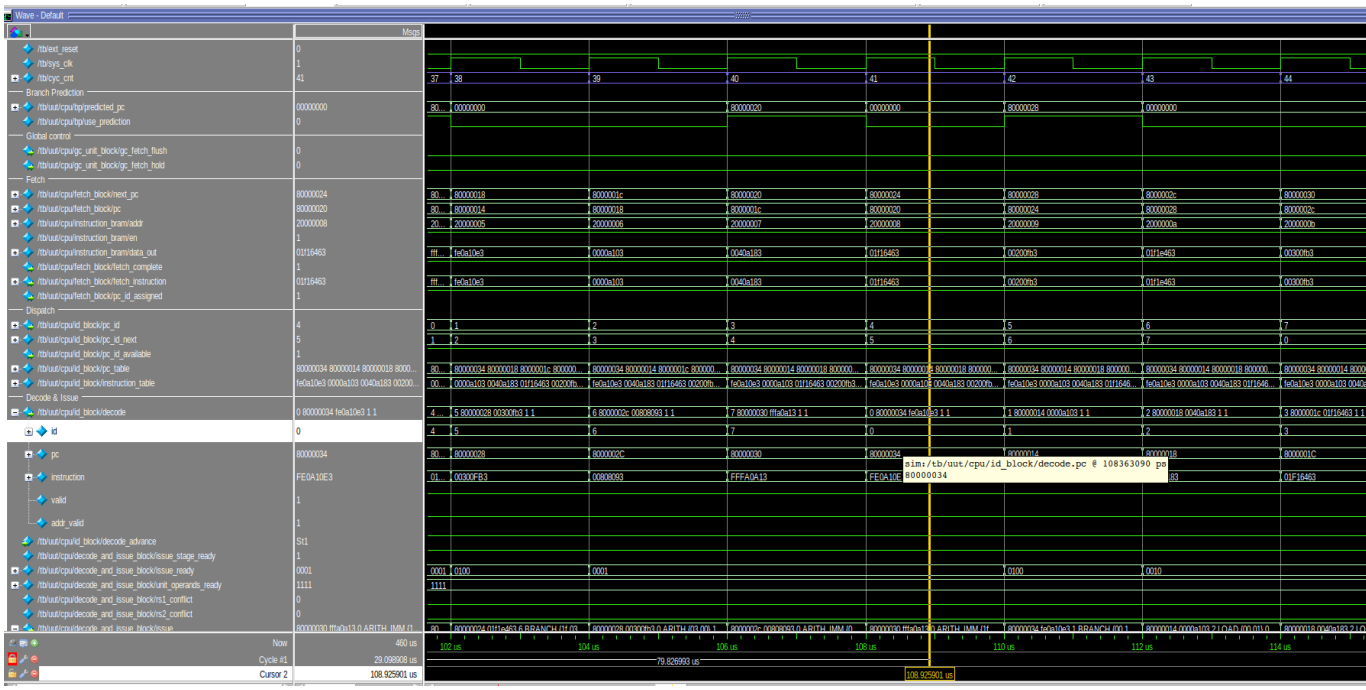

Рисунок 3.2 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 80000034 на второй итерации.

3.3 Задание №4

Условие задания

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

Результаты выполнения

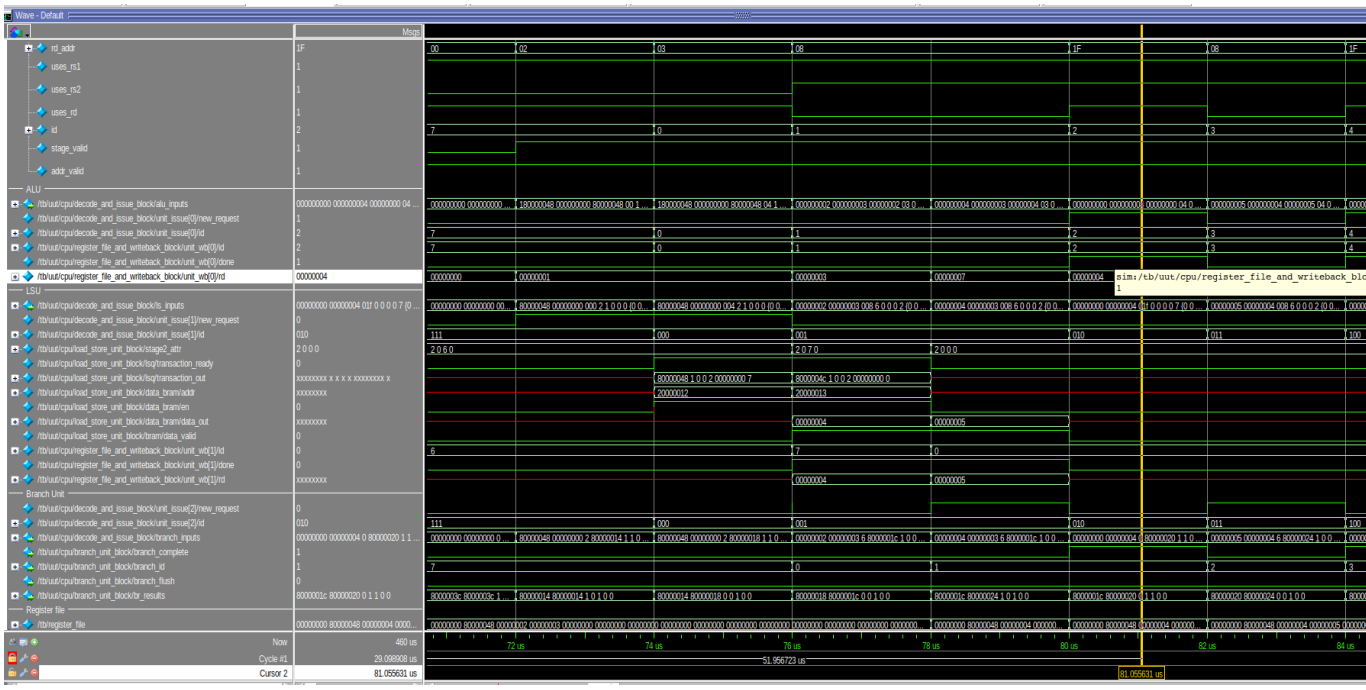

Рисунок 3.3 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 80000020 на первой итерации.

3.4 Задание №5

Трасса работы программы

Трасса работы представлена на рисунке 3.4.

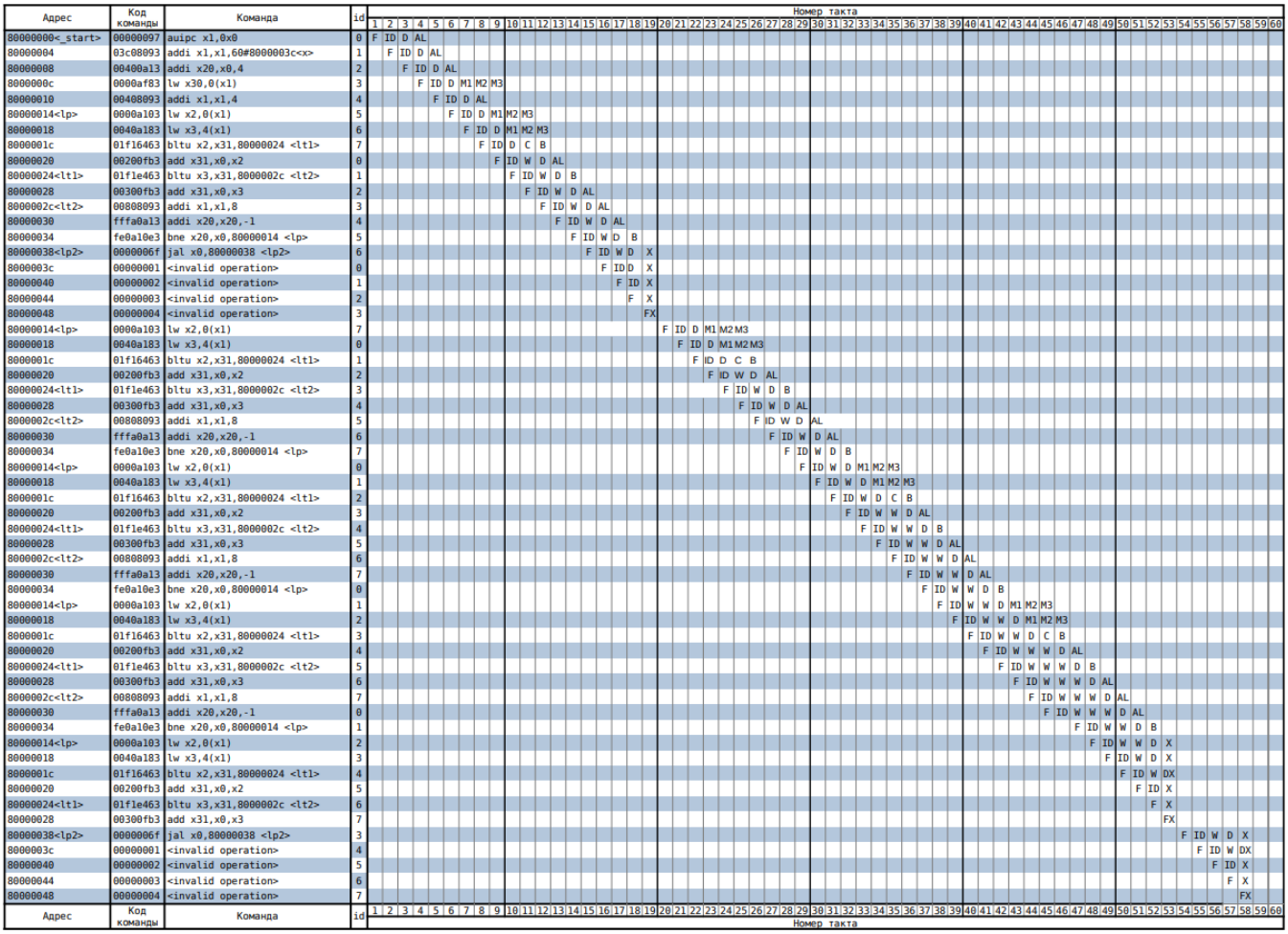

Рисунок 3.4 – Трасса работы программы.

Временные диаграммы

Временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом #! (add x31, x0, x2) с адресом *80000020* представлены на рисунке 3.5 – 3.8.

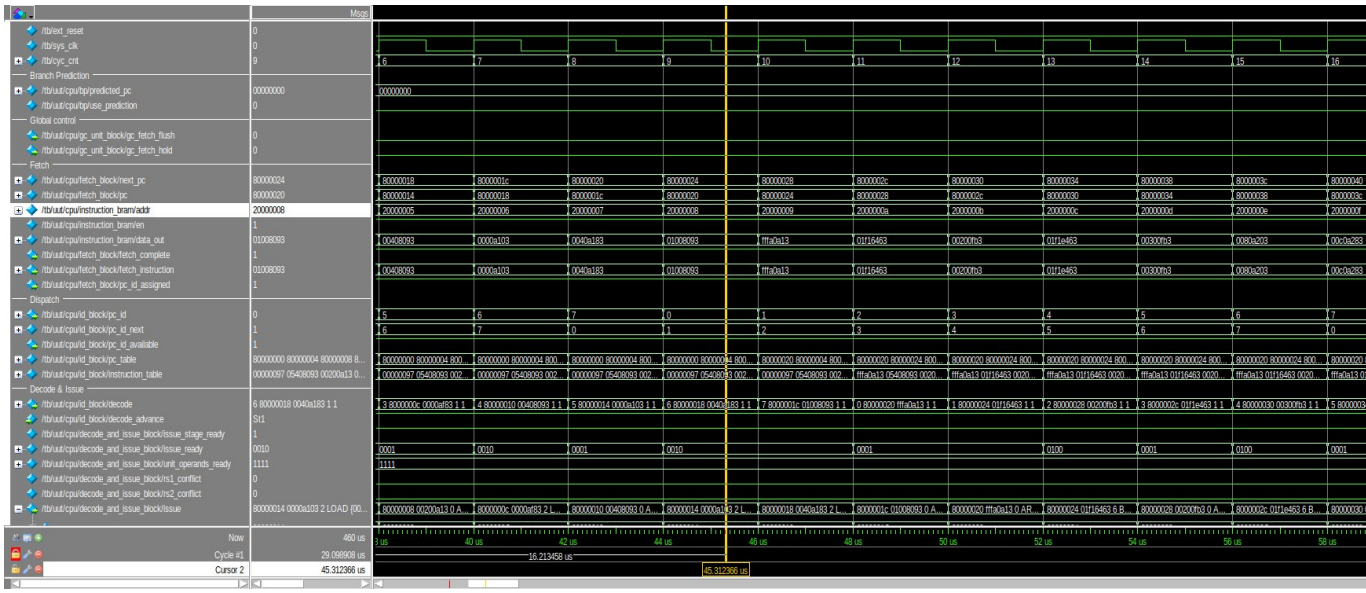

Рисунок 3.5 – Временные диаграммы сигналов выборки (Fetch) с адресом 80000020.

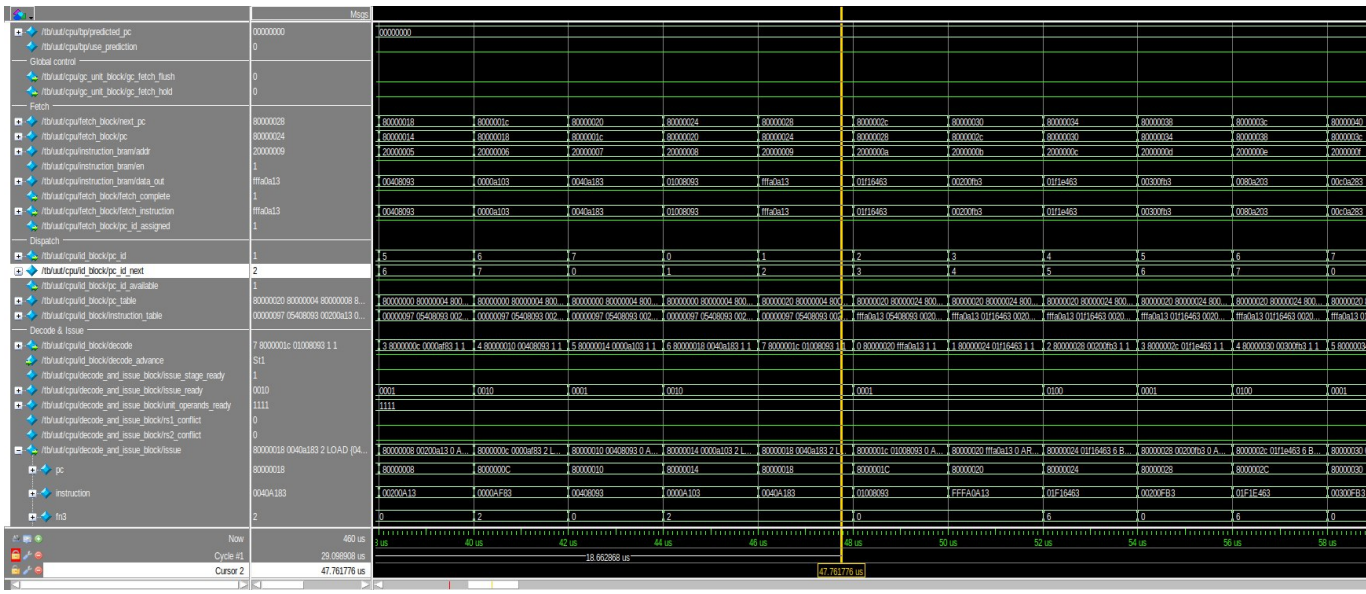

Рисунок 3.6 – Временные диаграммы сигналов диспетчеризации (Dispatch) с адресом 80000020.

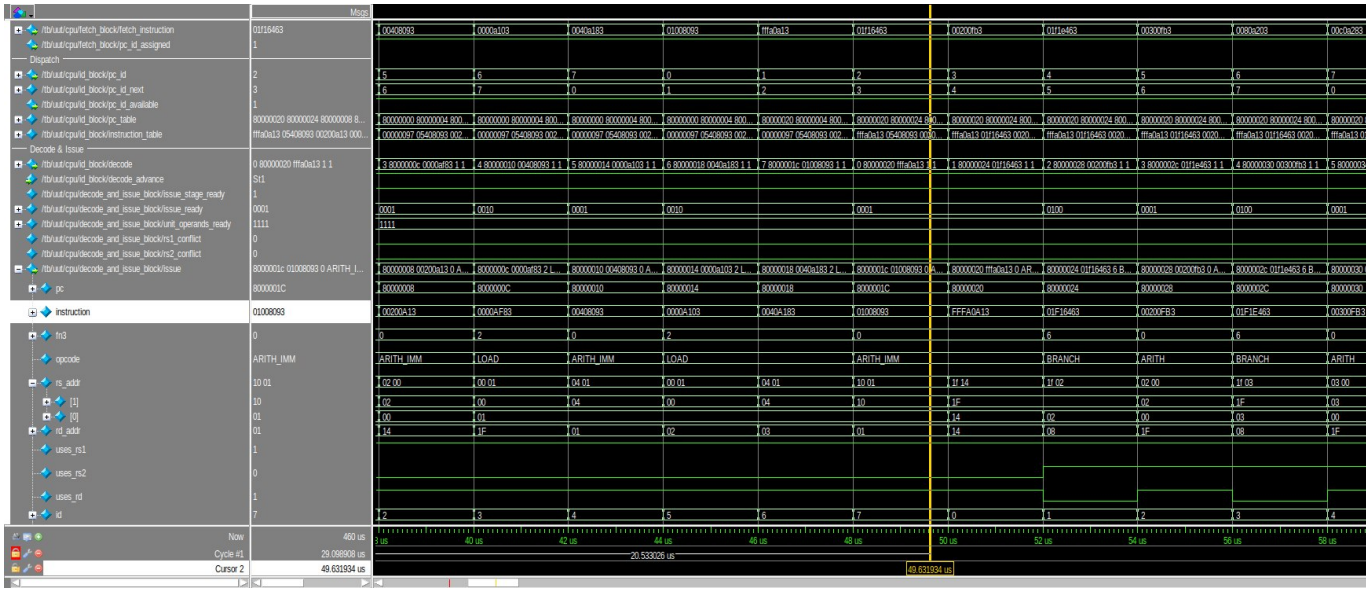

Рисунок 3.7 – Временные диаграммы сигналов декодирование и планирование (Decode & Issue) с адресом 80000020.

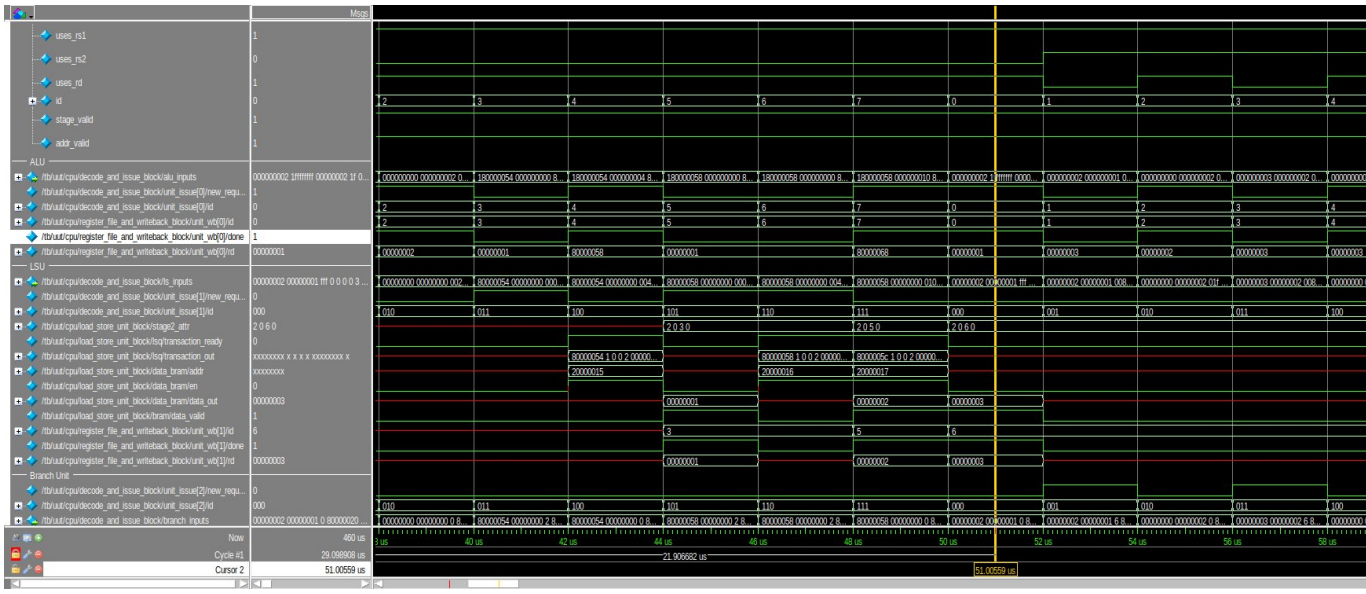

Рисунок 3.8 – Временные диаграммы сигналов ALU с адресом 80000020.

Вывод и предложение по оптимизации

Как видно на трассе работы программы, представленной на рисунке, 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.

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 3 такта в оптимизированной программе.

Оптимизированная программа

Код программы представлен в листинте 3.1

Листинг 3.1 – Код программы (оптимизированный)

```
1     .section .text
2     .globl _start;
3     len = 9 #Размер массива
4     enroll = 2 #Количество обрабатываемых элементов за одну итерацию
5     elem_sz = 4 #Размер одного элемента массива
6
7     _start:
8     la x1, _x
9     addi x20, x0, (len-1)/enroll
10    lw x31, 0(x1)
11    addi x1, x1, elem_sz*1
12    lp:
13    lw x2, 0(x1)
14    lw x3, 4(x1)
15    addi x20, x20, -1
16    bltu x2, x31, lt1
17    add x31, x0, x2 #!
18    lt1:    bltu x3, x31, lt2
19    add x31, x0, x3
20    lt2:
21    add x1, x1, elem_sz*enroll
22    bne x20, x0, lp
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
34 .4byte 0x9
```

Дизассемблерный код представлен на листинге 3.2.

Листинг 3.2 – Дизассемблированный код (оптимизированный)

```
1     Disassembly of section .text:
2
3     80000000 <_start>:
4     80000000:   00000097            auipc   x1,0x0
5     80000004:   03c08093            addi    x1,x1,60 # 8000003c <_x>
6     80000008:   00400a13            addi    x20,x0,4
7     8000000c:   0000af83            lw  x31,0(x1)
8     80000010:   00408093            addi    x1,x1,4
9
10    80000014 <lp>:
11    80000014:   0000a103            lw  x2,0(x1)
12    80000018:   0040a183            lw  x3,4(x1)
13    8000001c:   fffa0a13            addi    x20,x20,-1
14    80000020:   01f16463            bltu    x2,x31,80000028 <lt1>
15    80000024:   00200fb3            add x31,x0,x2
16
17    80000028 <lt1>:
18    80000028:   01f1e463            bltu    x3,x31,80000030 <lt2>
19    8000002c:   00300fb3            add x31,x0,x3
20
21    80000030 <lt2>:
22    80000030:   00808093            addi    x1,x1,8
23    80000034:   fe0a10e3            bne x20,x0,80000014 <lp>
24
25    80000038 <lp2>:
26    80000038:   0000006f            jal x0,80000038 <lp2>
```

Можно сказать, что данная программа эквивалентна следующему псевдокоду на языке C, представленному на листинге 3.3.

Листинг 3.3 – Псевдокод программы (оптимизированный)

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

Трасса работы оптимизированной программы

Трасса работы представлена на рисунке 3.9.

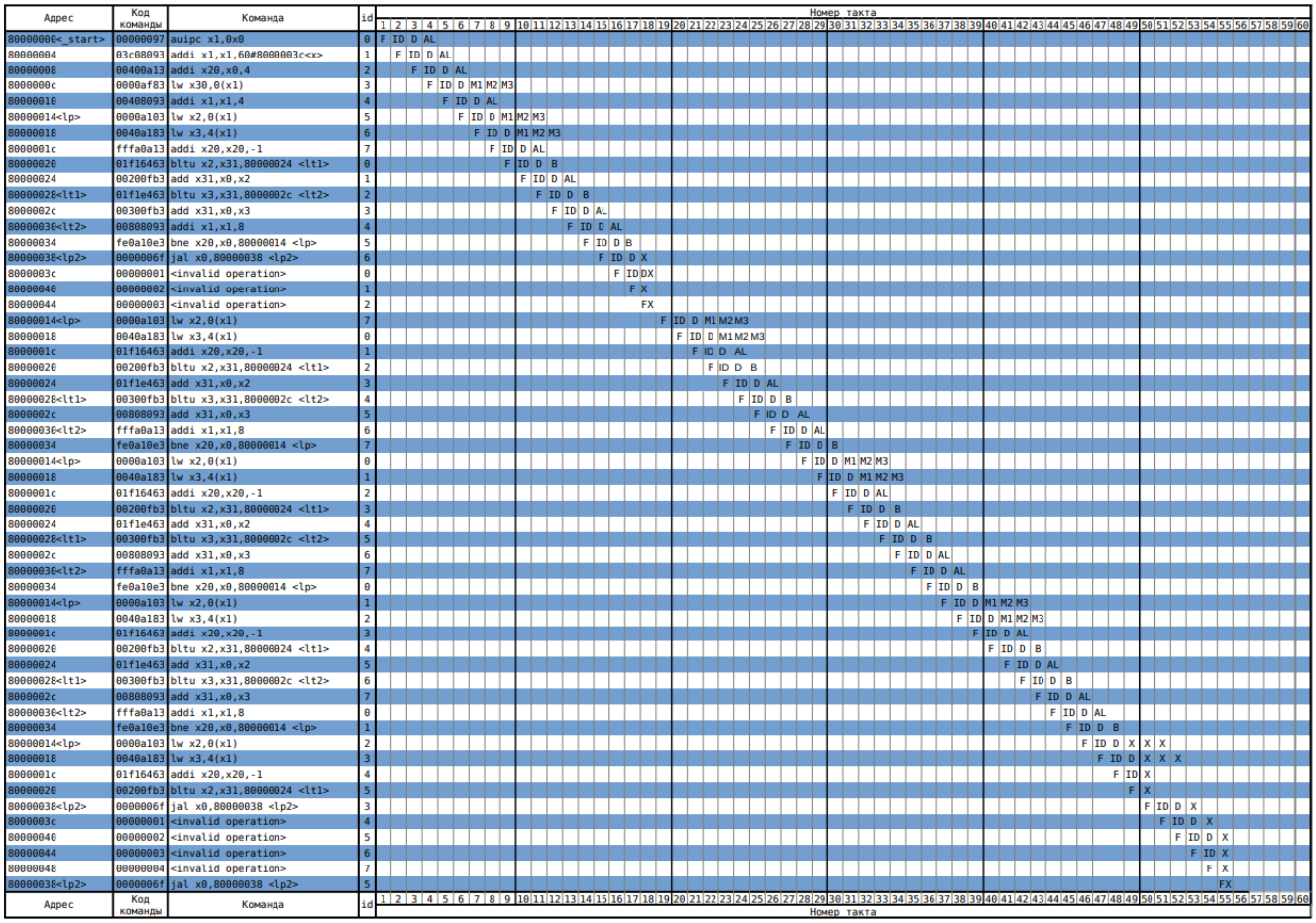

Рисунок 3.9 – Трасса работы оптимизированной программы.

Заключение

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.

Приложение

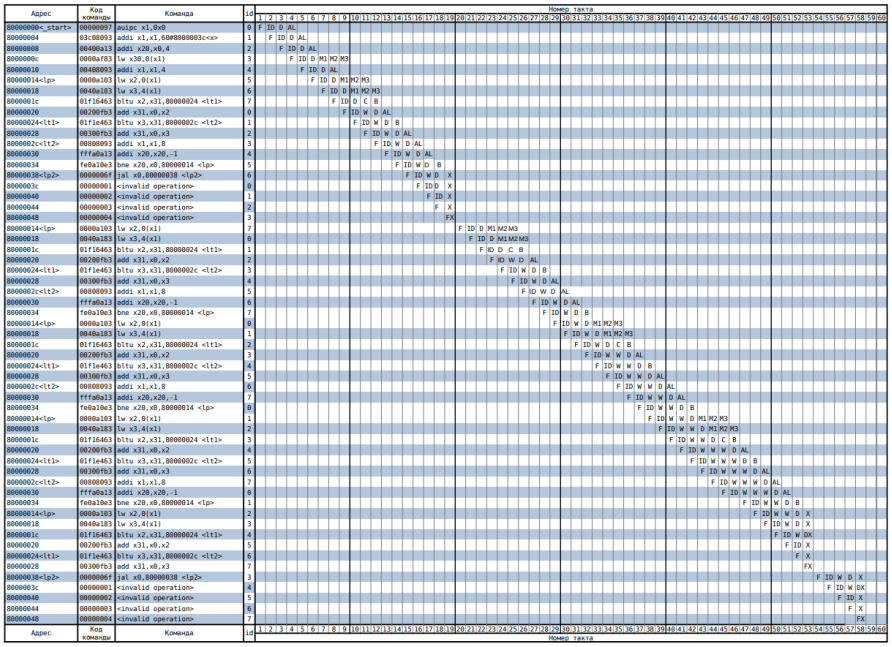

Рисунок 3.10 – Трасса работы программы.

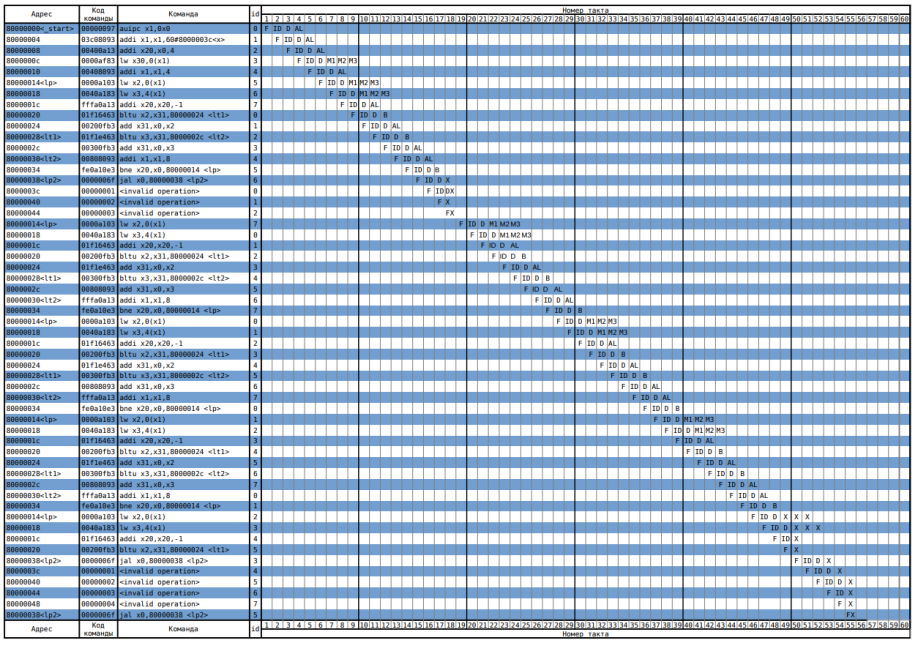

Рисунок 3.11 – Трасса работы оптимизированной программы.