**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 по курсу "Архитектура ЭВМ"

**Тема** Изучение принципов работы микропроцессорного ядра RISC-V

**Студент** Лемешев А. П.

**Группа** ИУ7-52Б

**Преподаватель** Дубровин Е.Н.

Москва — 2022 г.

# Оглавление

**Введение** **2**

**1 Основные теоретические сведения** **3**

**2 Эксперименты** **4**

2.1 Задание 1 . . . . . 4

2.2 Задание 2 . . . . . 6

2.3 Задание 3 . . . . . 7

2.4 Задание 4 . . . . . 7

2.5 Задание 5 . . . . . 8

**Заключение** **12**

# Введение

Основной целью данной лабораторной 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

Изучение архитектуры суперскалярных конвейерных микропроцессоров используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

Термин RISC-V 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.

# 2 Эксперименты

## 2.1 Задание 1

Дизассемблировать программу по индивидуальному варианту.

```
        .section .text
        .globl _start;
        len = 8 #Размер массива
        enroll = 4 #Количество обрабатываемых элементов за одну итерацию
        elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x1, elem_sz*len #Адрес последнего элемента
        add x31, x0, x0
lp:
        lw x2, 0(x1)
        lw x3, 4(x1)
        add x31, x31, x2 #!
        add x31, x31, x3
        lw x4, 8(x1)
        lw x5, 12(x1)
        add x31, x31, x4
        add x31, x31, x5
        addi x1, x1, elem_sz*enroll
        bne x1, x20, lp
        addi x31, x31, 1
lp2: j lp2

        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x5
        .4byte 0x6
        .4byte 0x7
        .4byte 0x8
```

Рис. 2.1: Код неоптимизированной программы

Создается массив из 8 элементов. Он последовательно заполняется числами от 1 до 8. Потом все элементы суммируются и в регистр x31 записывается накопленное значение. Затем содержимое регистра x31 инкрементируется.

```
main.elf: file format elf32-littleriscv

SYMBOL TABLE:
80000000 l  d   .text   00000000    .text
80000040 l  d   .data   00000000    .data
00000000 l  df  *ABS*   00000000    main.o
00000008 l      *ABS*   00000000    len
00000004 l      *ABS*   00000000    enroll
00000004 l      *ABS*   00000000    elem_sz
80000040 l      .data   00000000    _x
80000010 l      .text   00000000    lp
8000003c l      .text   00000000    lp2
80000000 g      .text   00000000    _start
80000060 g      .data   00000000    _end
```

Рис. 2.2: Таблица символов

```
Disassembly of section .text:
80000000 <_start>:
80000000:   00000097    auipc x1,0x0
80000004:   04008093    addi x1,x1,64 # 80000040 <_x>
80000008:   02008a13    addi x20,x1,32
8000000c:   00000fb3    add x31,x0,x0
80000010 <lp>:
80000010:   0000a103    lw x2,0(x1)
80000014:   0040a183    lw x3,4(x1)
80000018:   002f8fb3    add x31,x31,x2
8000001c:   003f8fb3    add x31,x31,x3
80000020:   0080a203    lw x4,8(x1)
80000024:   00c0a283    lw x5,12(x1)
80000028:   004f8fb3    add x31,x31,x4
8000002c:   005f8fb3    add x31,x31,x5
80000030:   01008093    addi x1,x1,16
80000034:   fd409ee3    bne x1,x20,80000010 <lp>
80000038:   001f8f93    addi x31,x31,1
8000003c <lp2>:
8000003c:   0000006f    jal x0,8000003c <lp2>
```

Рис. 2.3: Дизассемблированная секция текста

```
39   Disassembly of section .data:
40   80000040 <_x>:
41   80000040:   0001        c.addi x0,0
42   80000042:   0000        unimp
43   80000044:   0002        0x2
44   80000046:   0000        unimp
45   80000048:   00000003    lb x0,0(x0) # 0 <elem_sz-0x4>
46   8000004c:   0004        c.addi4spn x9,x2,0
47   8000004e:   0000        unimp
48   80000050:   0005        c.addi x0,1
49   80000052:   0000        unimp
50   80000054:   0006        0x6
51   80000056:   0000        unimp
52   80000058:   00000007    0x7
53   8000005c:   0008        c.addi4spn x10,x2,0
```

Рис. 2.4: Дизассемблированная секция данных

## 2.2 Задание 2

Получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 80000000 на 2-ой итерации.

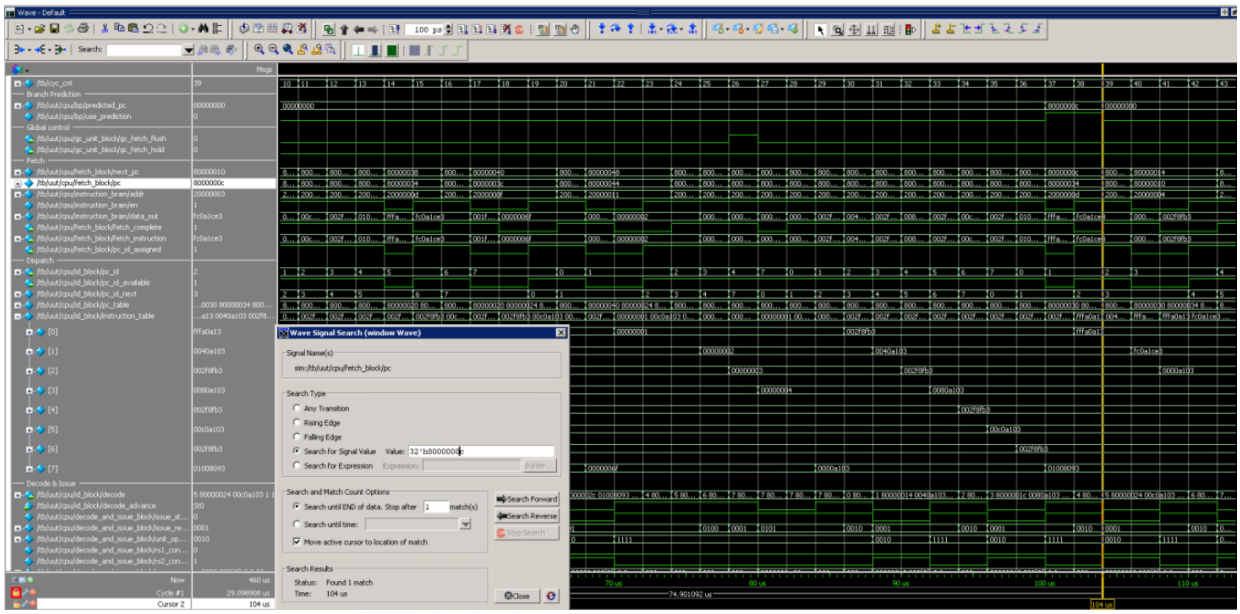

Рис. 2.5: Временная диаграмма выполнения стадий выборки и диспетчеризации команды 8000000c на 2-й итерации

## 2.3 Задание 3

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 80000018 на 2-ой итерации.

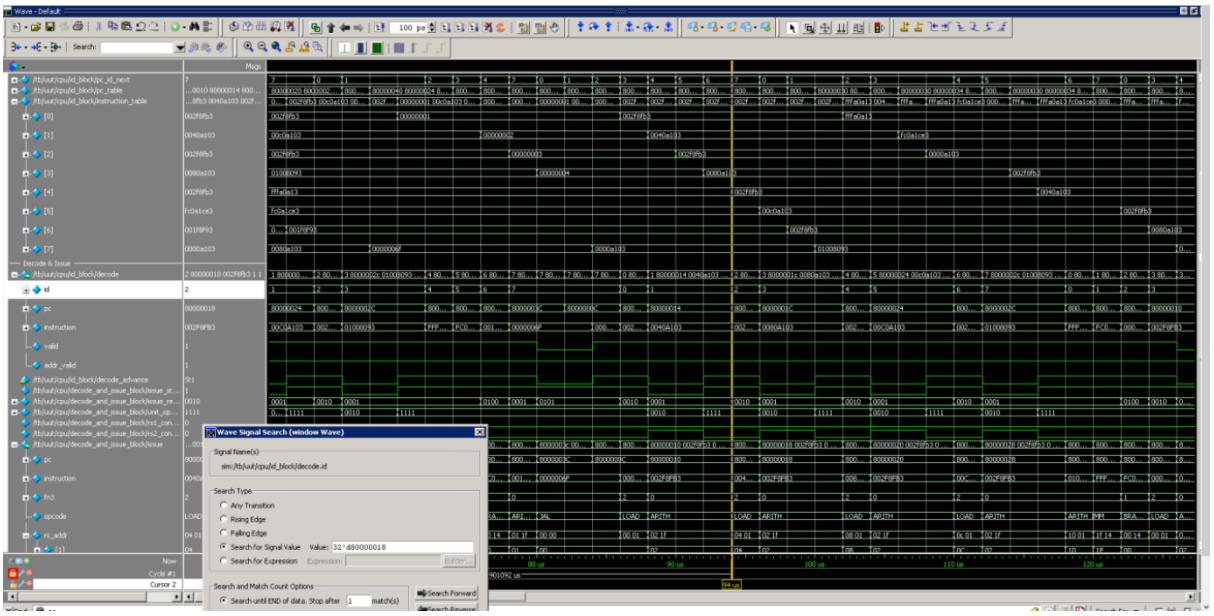

Рис. 2.6: Временная диаграмма выполнения стадий декодирования и планирования на выполнение команды 80000018 на 2-й итерации

## 2.4 Задание 4

Получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды с адресом 8000002c на 1-ой итерации.

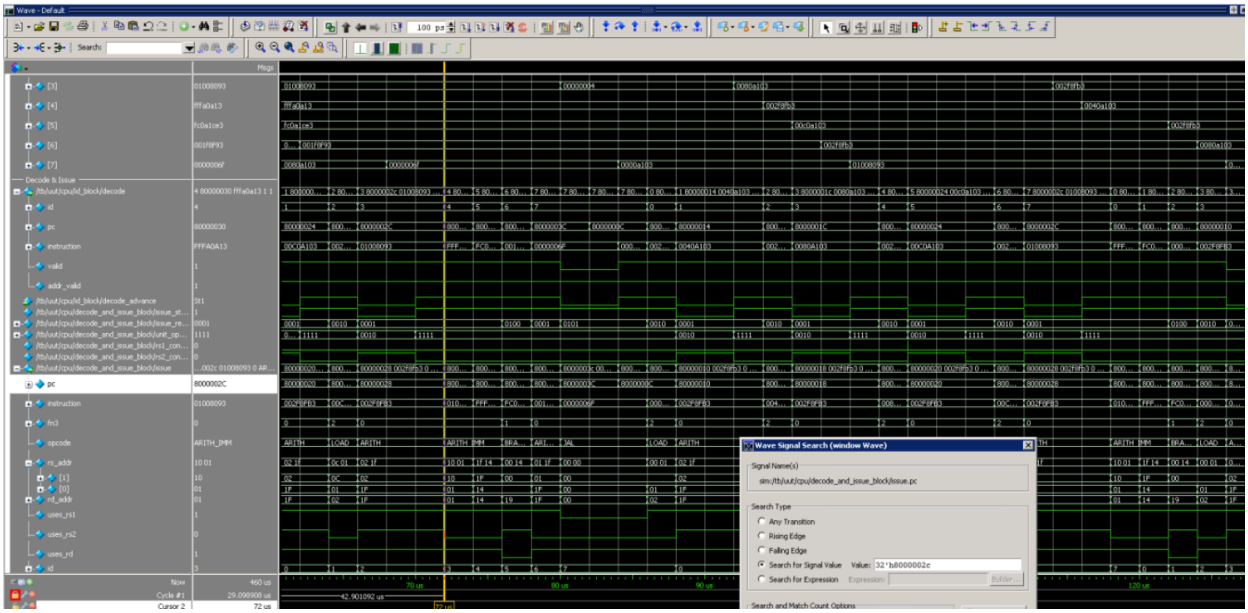

Рис. 2.7: Временная диаграмма выполнения стадии выполнения команды 8000002c на 1-й итерации

## 2.5 Задание 5

Оптимизировать программу.

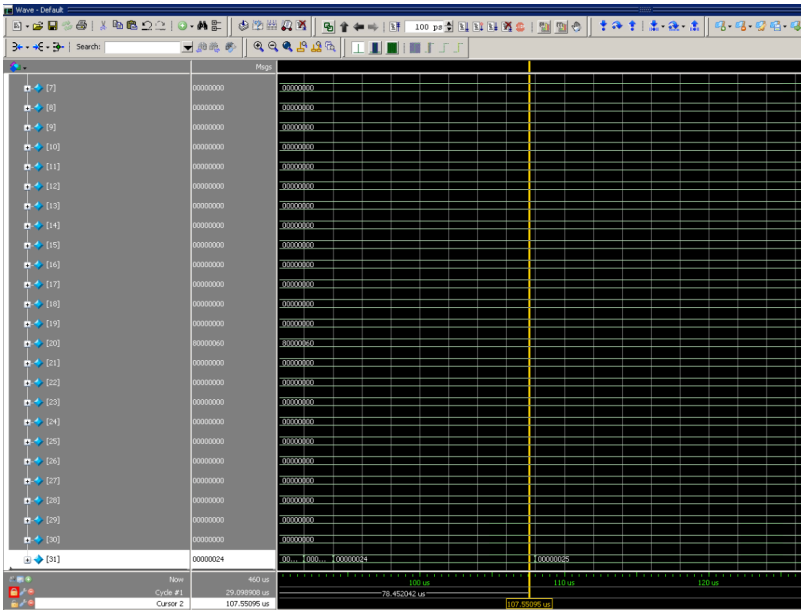

Рис. 2.8: Результат выполнения программы

Значение регистра x31 в конце выполнения программы равно 25h = 37, как и предполагалось ранее.

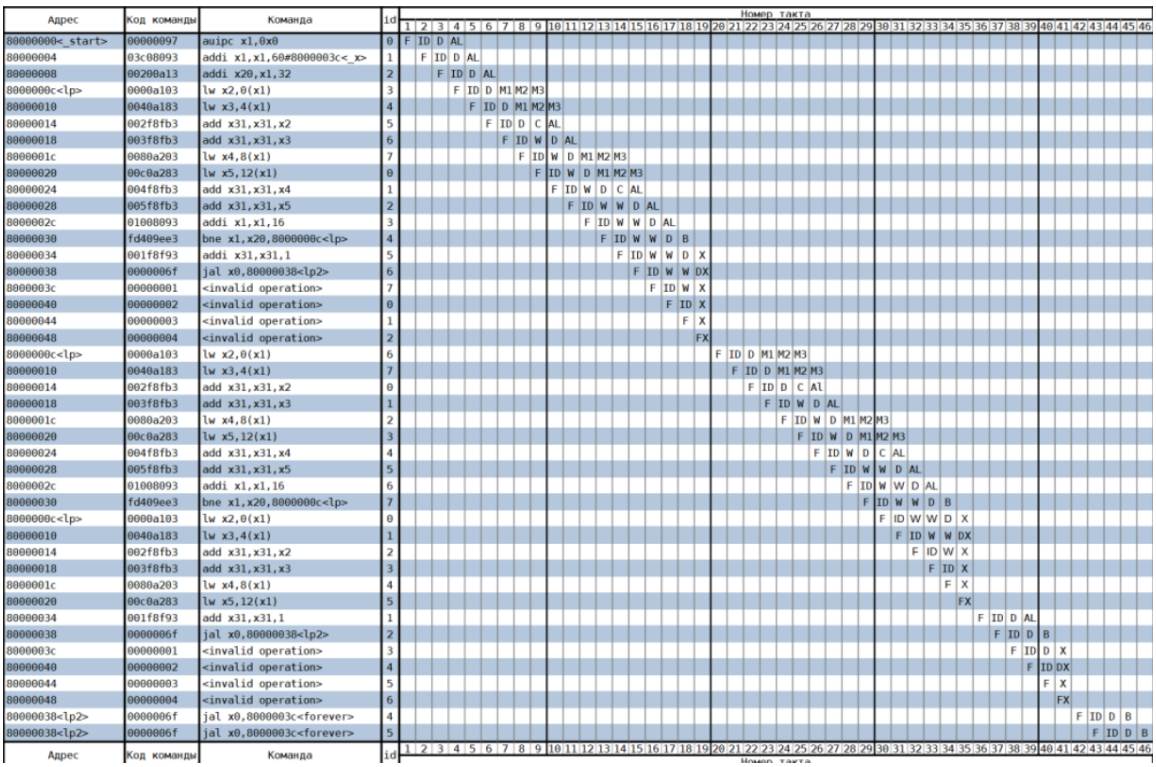


Рис. 2.9: Трасса работы неоптимизированной программы

Конфликты возникают, когда после выполнения инструкции *lw* выполняется инструкция *add*. Оптимизируем программу, перенеся все команды *lw* выше инструкций *add*.

```
        .section .text
        .globl _start;
        len = 8 # Размер массива
        enroll = 4 # Количество обрабатываемых элементов за одну итерацию
        elem_sz = 4 # Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x1, elem_sz*len #Адрес последнего элемента
        add x31, x0, x0
lp:
        lw x2, 0(x1)
        lw x3, 4(x1)
          lw x4, 8(x1)
        lw x5, 12(x1)
        add x31, x31, x2 #!
        add x31, x31, x3
        add x31, x31, x4
        add x31, x31, x5
        addi x1, x1, elem_sz*enroll
        bne x1, x20, lp
        addi x31, x31, 1
lp2: j lp2

        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x5
        .4byte 0x6
        .4byte 0x7
        .4byte 0x8
```

Рис. 2.10: Код оптимизированной программы

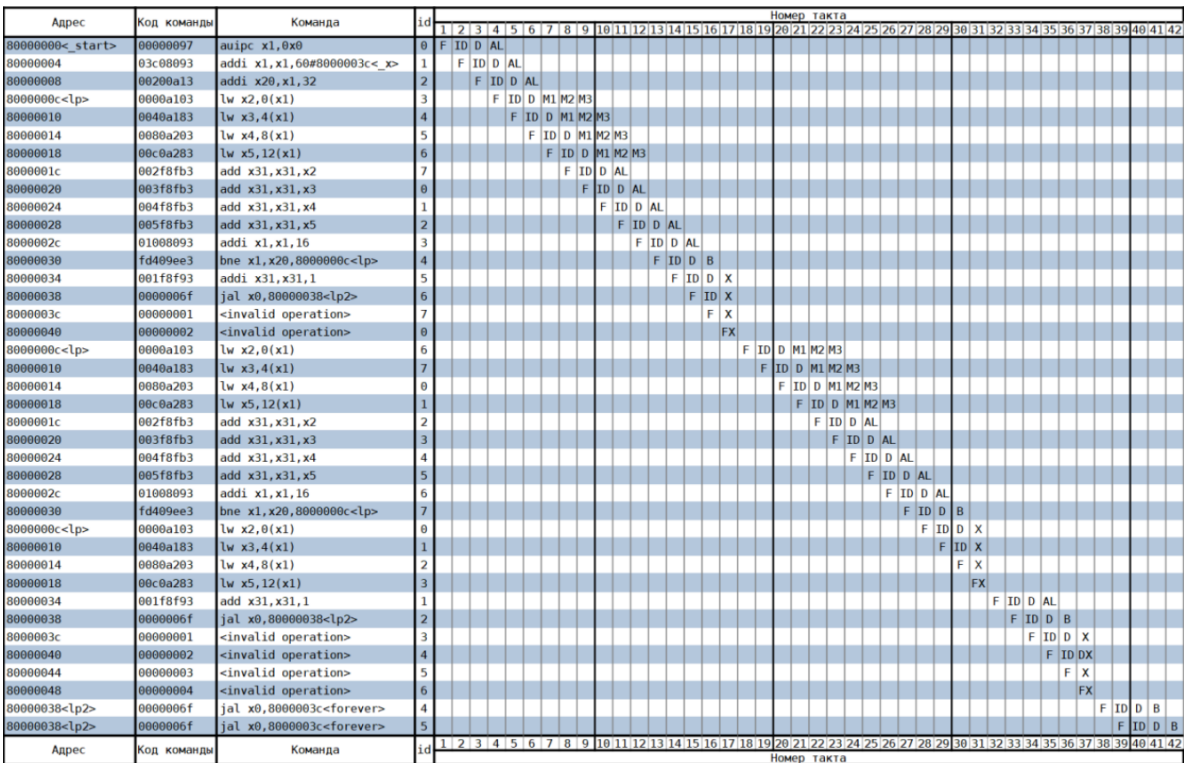


Рис. 2.11: Трасса работы оптимизированной программы

Благодаря оптимизации программы получилось избавиться от конфликтов.

# 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.