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»

Лабораторная работа №2
по дисциплине «Архитектура ЭВМ»

Тема Изучение принципов работы микропроцессорного ядра RISC-V

Студент Тузов Даниил Александрович

Группа ИУ7-52Б

Преподаватель Калитвенец Максим, Попов А.Ю.

Москва, 2024 г.

1 Введение

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.

Для достижения поставленных целей в настоящей лабораторной работе используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

В ходе лабораторной работы используется средство моделирования MentorGrap Modelsim для моделирования работы исследуемого микропроцессора в процессе выполнения программы и наблюдения формы внутренних сигналов.

2 Задание 1

2.1 Исходный текст исследуемой программы

Листинг 1 – Исходный текст исследуемой программы

```
.section .text
        .globl _start;
        len = 9
        enroll = 2
        elem_sz = 4
_start:
        la x1, _x
        addi x20, x1, elem_sz*len
        lw x31, 0(x1)
        addi x1, x1, elem_sz*1
lp:
        lw x2, 0(x1)    #!
        lw x3, 4(x1)
        bltu x2, x31, lt1
        add x31, x0, x2
lt1:    bltu x3, x31, lt2
        add x31, x0, x3
```

```
lt2:
        add x1, x1, elem_sz*enroll
        bne x1, x20, lp
lp2: j lp2
        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x8
        .4byte 0x6
        .4byte 0x7
        .4byte 0x5
        .4byte 0x4
```

2.2 Дизассемблированный листинг

Листинг 2 – Дизассемблированный листинг программы

```
Disassembly of section .text:
80000000 <_start>:
80000000:  00000097            auipc  x1,0x0
80000004:  03808093            addi  x1,x1,56 # 80000038 <_x>
80000008:  02408a13            addi  x20,x1,36
8000000c:  0000af83            lw  x31,0(x1)
80000010:  00408093            addi  x1,x1,4
80000014 <lp>:
80000014:  0000a103            lw  x2,0(x1)
80000018:  0040a183            lw  x3,4(x1)
8000001c:  00808093            addi  x1,x1,8
80000020:  01f16463            bltu  x2,x31,80000028 <lt1>
80000024:  00200fb3            add  x31,x0,x2
80000028 <lt1>:
80000028:  01f1e463            bltu  x3,x31,80000030 <lt2>
8000002c:  00300fb3            add  x31,x0,x3
80000030 <lt2>:
80000030:  ff4092e3            bne  x1,x20,80000014 <lp>
80000034 <lp2>:
80000034:  0000006f            jal  x0,80000034 <lp2>
Disassembly of section .data:
80000038 <_x>:
80000038:  0001                  .insn  2, 0x0001
8000003a:  0000                  .insn  2, 0x
8000003c:  0002                  .insn  2, 0x0002
```

```
8000003e:  0000                    .insn  2, 0x
80000040:  00000003              lb  x0,0(x0) # 0 <enroll-0x2>
80000044:  0004                    .insn  2, 0x0004
80000046:  0000                    .insn  2, 0x
80000048:  0008                    .insn  2, 0x0008
8000004a:  0000                    .insn  2, 0x
8000004c:  0006                    .insn  2, 0x0006
8000004e:  0000                    .insn  2, 0x
80000050:  00000007              .insn  4, 0x0007
80000054:  0005                    .insn  2, 0x0005
80000056:  0000                    .insn  2, 0x
80000058:  0004                    .insn  2, 0x0004
```

2.3 Псевдокод, поясняющий работу программы

Листинг 3 – Псевдокод

```
#define len 9
#define enroll 2
#define elem_sz 4
unsigned _x[] = { 1, 2, 3, 4, 8, 6, 7, 5, 4 };
void _start() {
    unsigned *x1 = _x; // la x1, _x
    unsigned *x20 = x1 + len; // addi x20, x1, elem_sz*len
    unsigned x31 = x1[0]; // lw x31, 0(x1)
    x1 += 1; // addi x1, x1, elem_sz*1
    do {
        // lp
        unsigned x2 = x1[0]; // lw x2, 0(x1)
        unsigned x3 = x1[1]; // lw x3, 4(x1)
        if (x2 < x31) { // bltu x2, x31, lt1
        else
            x31 = x2; // add x31, x0, x2
        // lt1
        if (x3 < x31) // bltu x3, x31, lt2
        else
            x31 = x3; // add x31, x0, x3
        // lt2
        x1 += enroll; // add x1, x1, elem_sz*enroll
    } while (x1 != x20); // bne x1, x20, lp
    for (;;); // lp2: j lp2
}
```

3 Задание 2

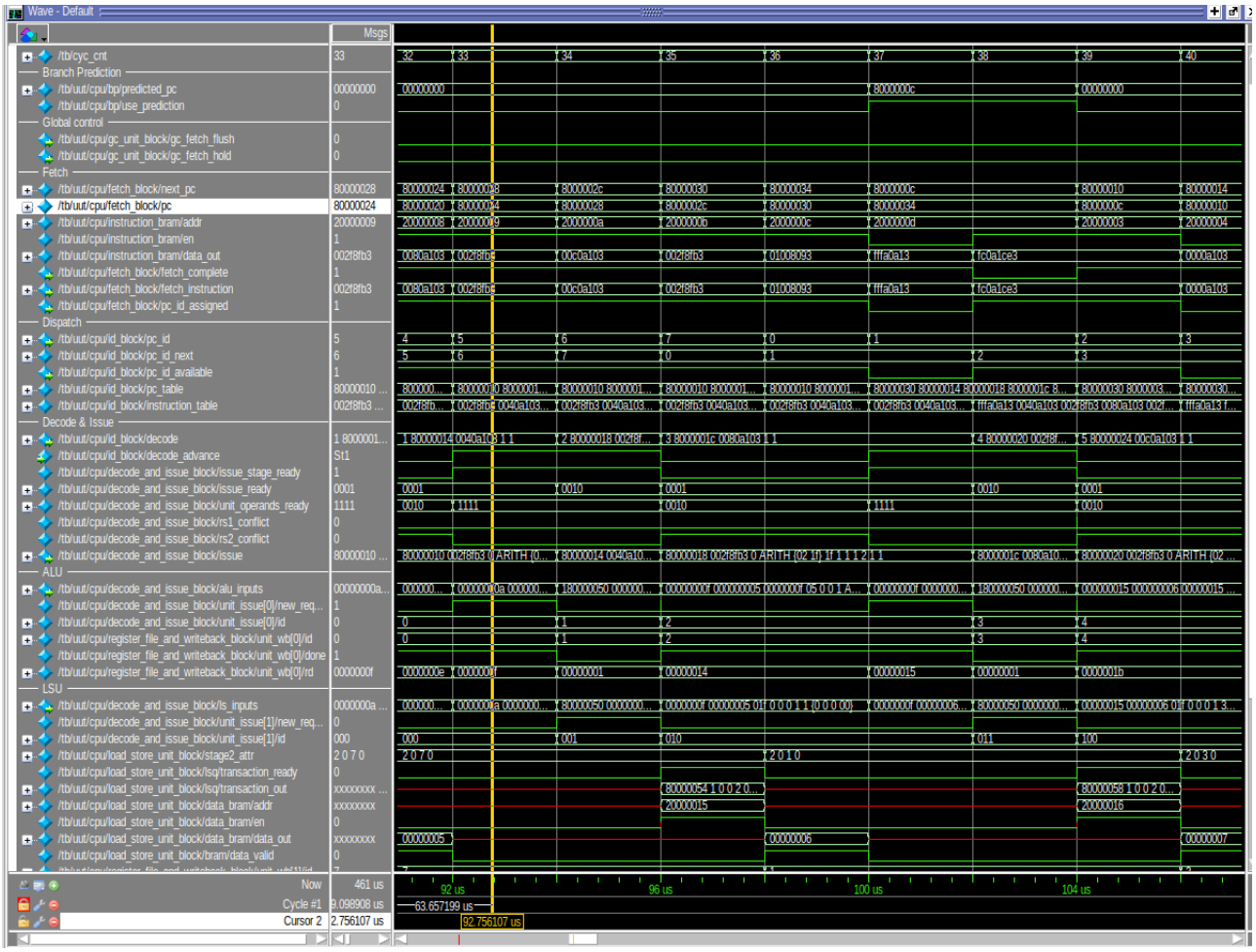

Рисунок 1 – Стадии выборки и диспетчеризации

4 Задание 3

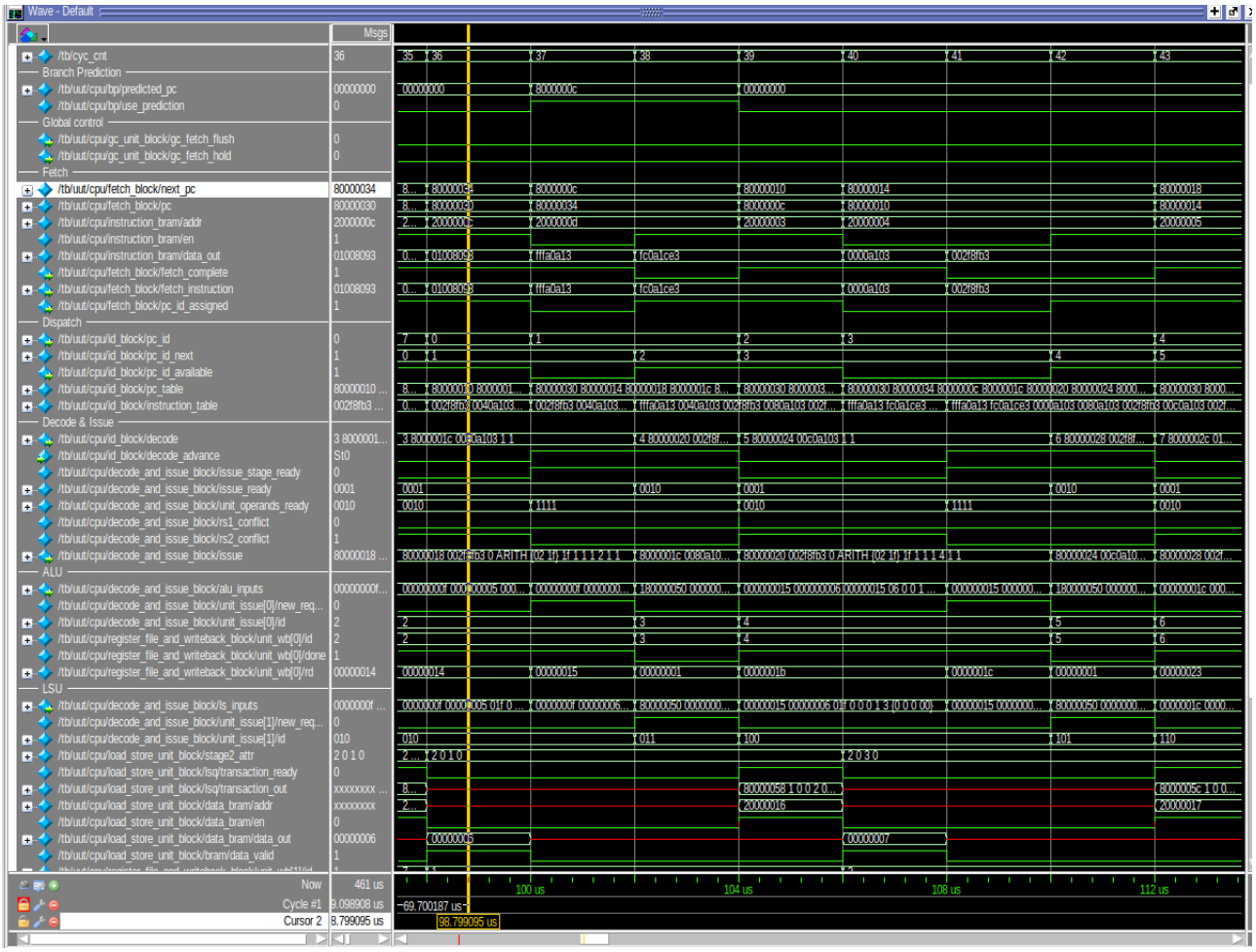

Рисунок 2 – Стадии декодирования и планирования

5 Задание 4

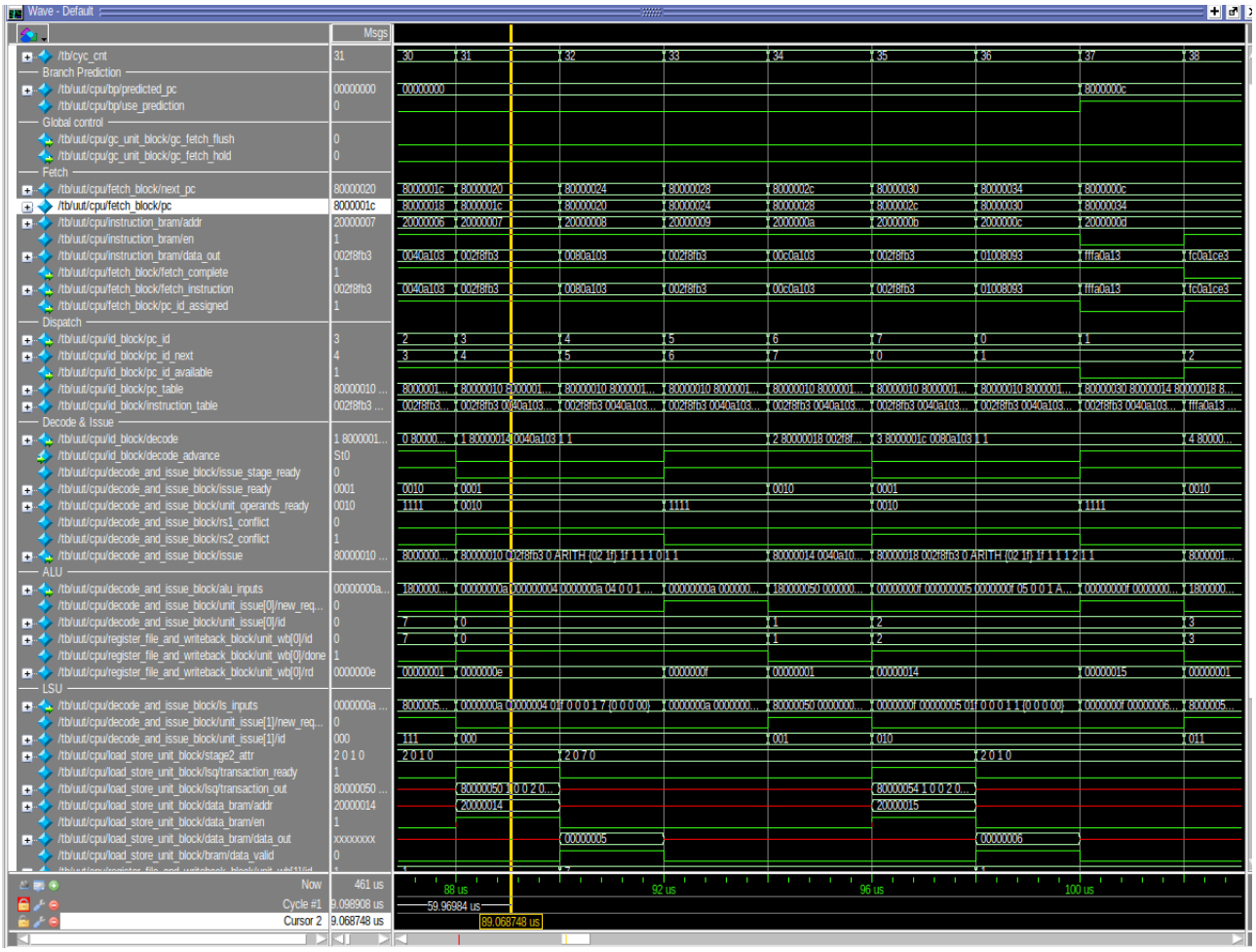

Рисунок 3 – Непосредственное выполнение комманды

6 Задание 5

6.1 Место, отмеченное решеткой

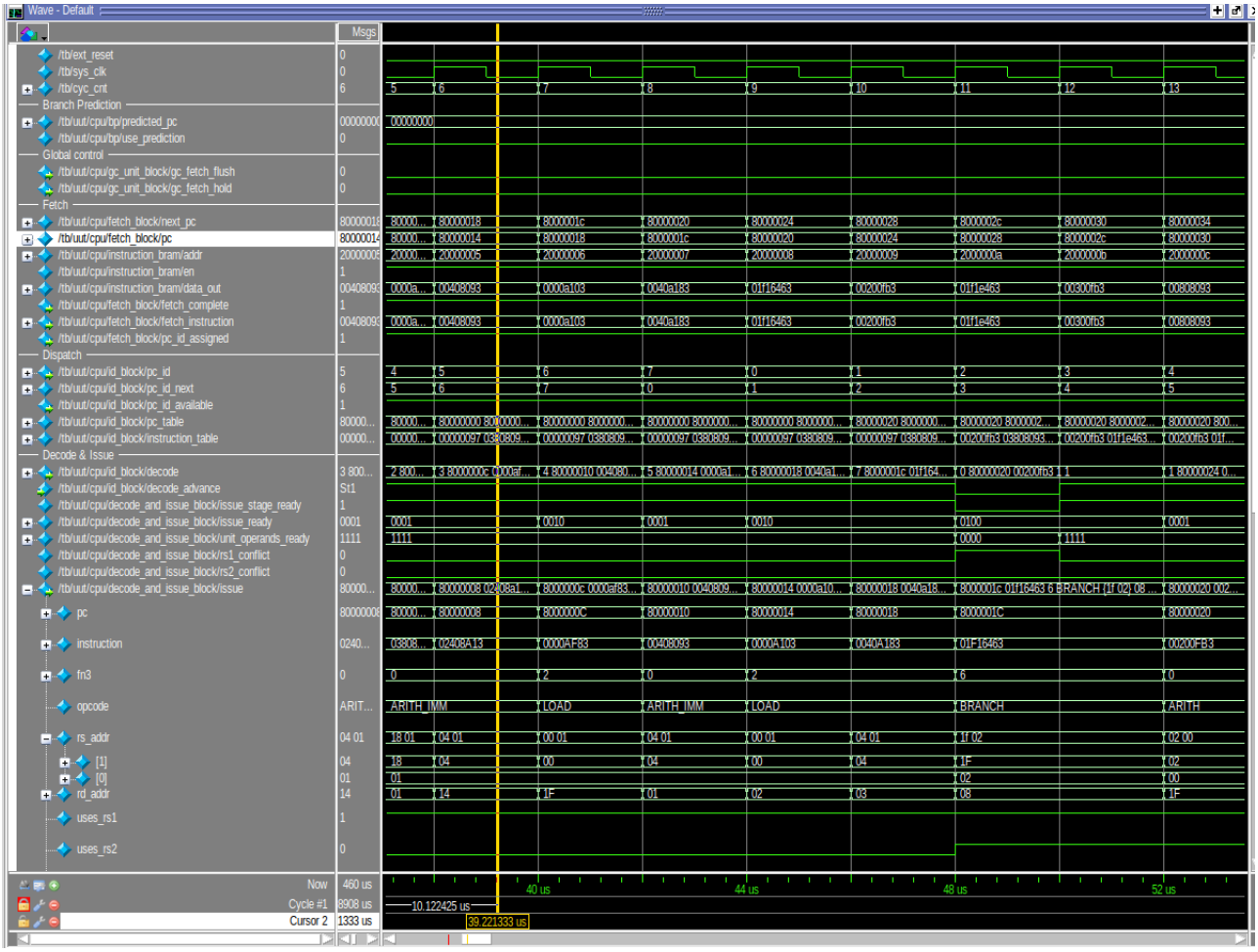

Рисунок 4 – Первая итерация команды, отмеченной #

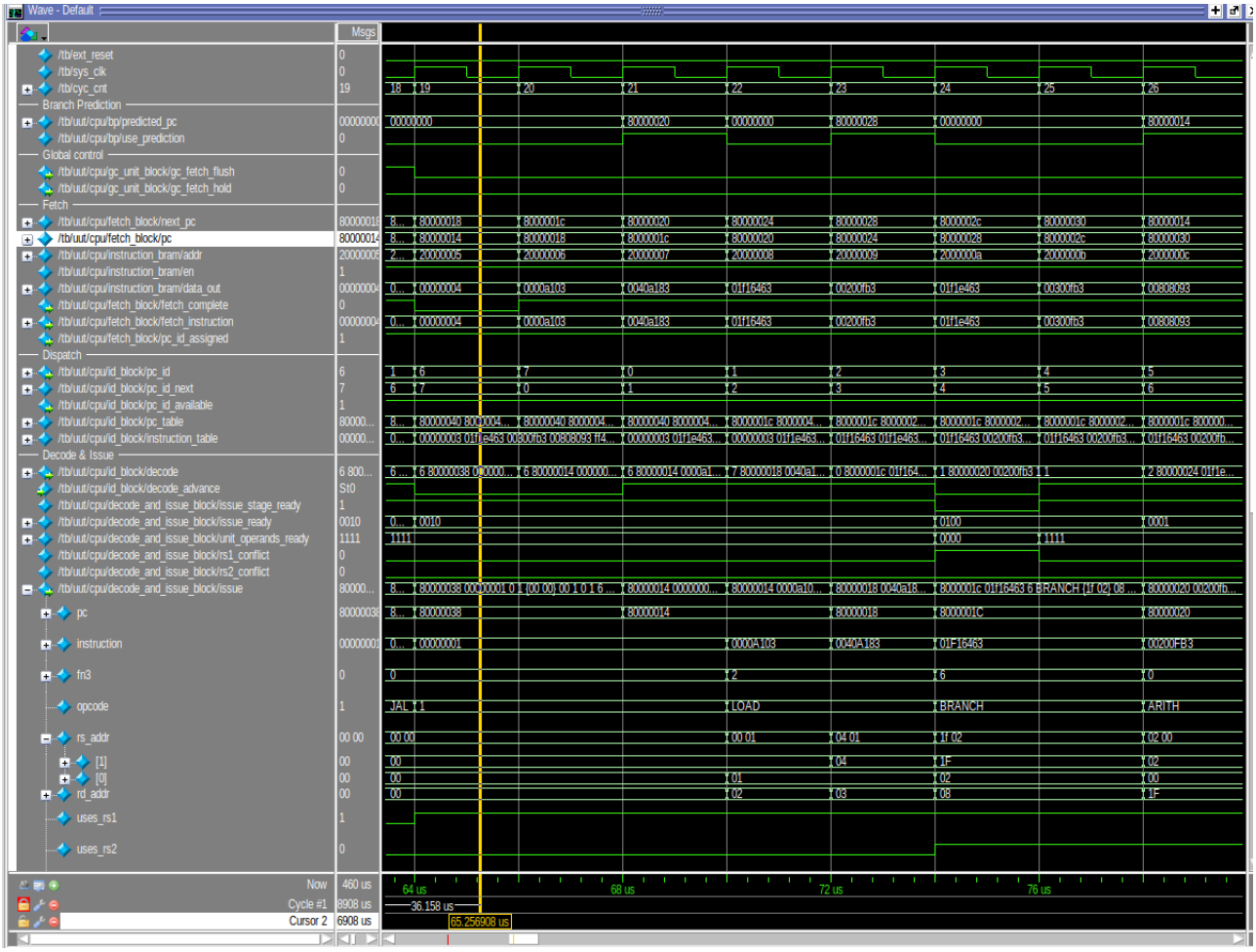

Рисунок 5 – Вторая итерация команды, отмеченной #

6.2 Трасса исходной программы

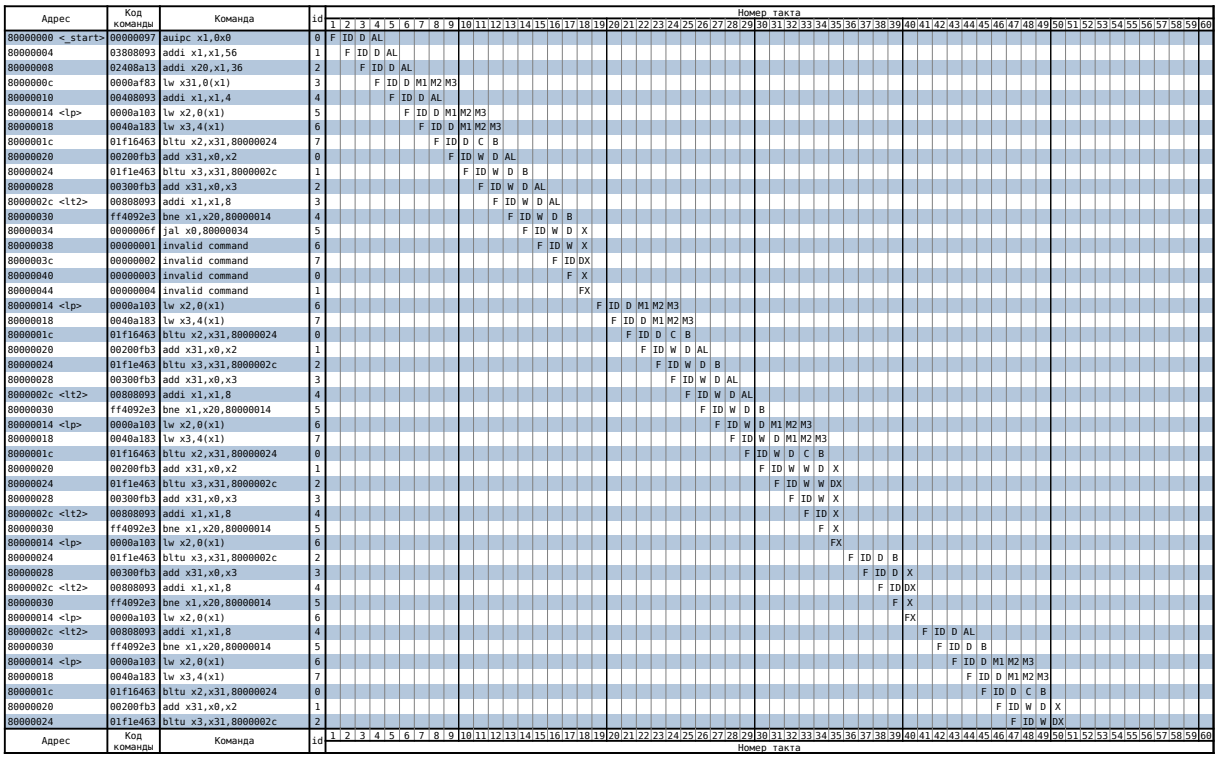

Рисунок 6 – Трасса выполнения неоптимизированной программы

Конфликты возникают из-за того, что происходит обращение к регистру, запись в который ещё не была завершена (регистр x2).

Строка x1 += enroll (add x1, x1, elem_sz*enroll) будет выполнена вне зависимости от предшествующих условных переходов, а также значение регистра x1 не используется после строки lw x3, 4(x1) в пределах выполнения тела цикла.

Следовательно, если поместить строку add x1, x1, slem_sz*enroll сразу после lw x3, 4(x1), это не повлияет на результат выполнения программы, зато даст дополнительную задержку в один такт между загрузкой значения в x2 и чтением из него, что позволит устранить конфликты.

6.3 Исходной текст оптимизированной программы

Листинг 4 – Исходный текст оптимизированной программы

```
.section .text
        .globl _start;
        len = 9
        enroll = 2
        elem_sz = 4
_start:
        la x1, _x
        addi x20, x1, elem_sz*len
        lw x31, 0(x1)
        addi x1, x1, elem_sz*1
lp:
        lw x2, 0(x1) #!
        lw x3, 4(x1)
        add x1, x1, elem_sz*enroll
        bltu x2, x31, lt1
        add x31, x0, x2
lt1:    bltu x3, x31, lt2
        add x31, x0, x3
lt2:
        bne x1, x20, lp
lp2: j lp2
        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x8
        .4byte 0x6
        .4byte 0x7
        .4byte 0x5
        .4byte 0x4
```

6.4 Дизассемблированный листинг оптимизированной программы

Листинг 5 – Дизассемблированный листинг оптимизированной программы

```
Disassembly of section .text:
80000000 <_start>:
80000000:  00000097            auipc  x1,0x0
80000004:  03808093            addi  x1,x1,56 # 80000038 <_x>
80000008:  02408a13            addi  x20,x1,36
8000000c:  0000af83            lw  x31,0(x1)
80000010:  00408093            addi  x1,x1,4
80000014 <lp>:
80000014:  0000a103            lw  x2,0(x1)
80000018:  0040a183            lw  x3,4(x1)
8000001c:  01f16463            bltu  x2,x31,80000024 <lt1>
80000020:  00200fb3            add  x31,x0,x2
80000024 <lt1>:
80000024:  01f1e463            bltu  x3,x31,8000002c <lt2>
80000028:  00300fb3            add  x31,x0,x3
8000002c <lt2>:
8000002c:  00808093            addi  x1,x1,8
80000030:  ff4092e3            bne  x1,x20,80000014 <lp>
80000034 <lp2>:
80000034:  0000006f            jal  x0,80000034 <lp2>
Disassembly of section .data:
80000038 <_x>:
80000038:  0001                  .insn  2, 0x0001
8000003a:  0000                  .insn  2, 0x
8000003c:  0002                  .insn  2, 0x0002
8000003e:  0000                  .insn  2, 0x
80000040:  00000003            lb  x0,0(x0) # 0 <enroll-0x2>
80000044:  0004                  .insn  2, 0x0004
80000046:  0000                  .insn  2, 0x
80000048:  0008                  .insn  2, 0x0008
8000004a:  0000                  .insn  2, 0x
8000004c:  0006                  .insn  2, 0x0006
8000004e:  0000                  .insn  2, 0x
80000050:  00000007            .insn  4, 0x0007
80000054:  0005                  .insn  2, 0x0005
80000056:  0000                  .insn  2, 0x
80000058:  0004                  .insn  2, 0x0004
```

6.5 Трасса оптимизированной программы

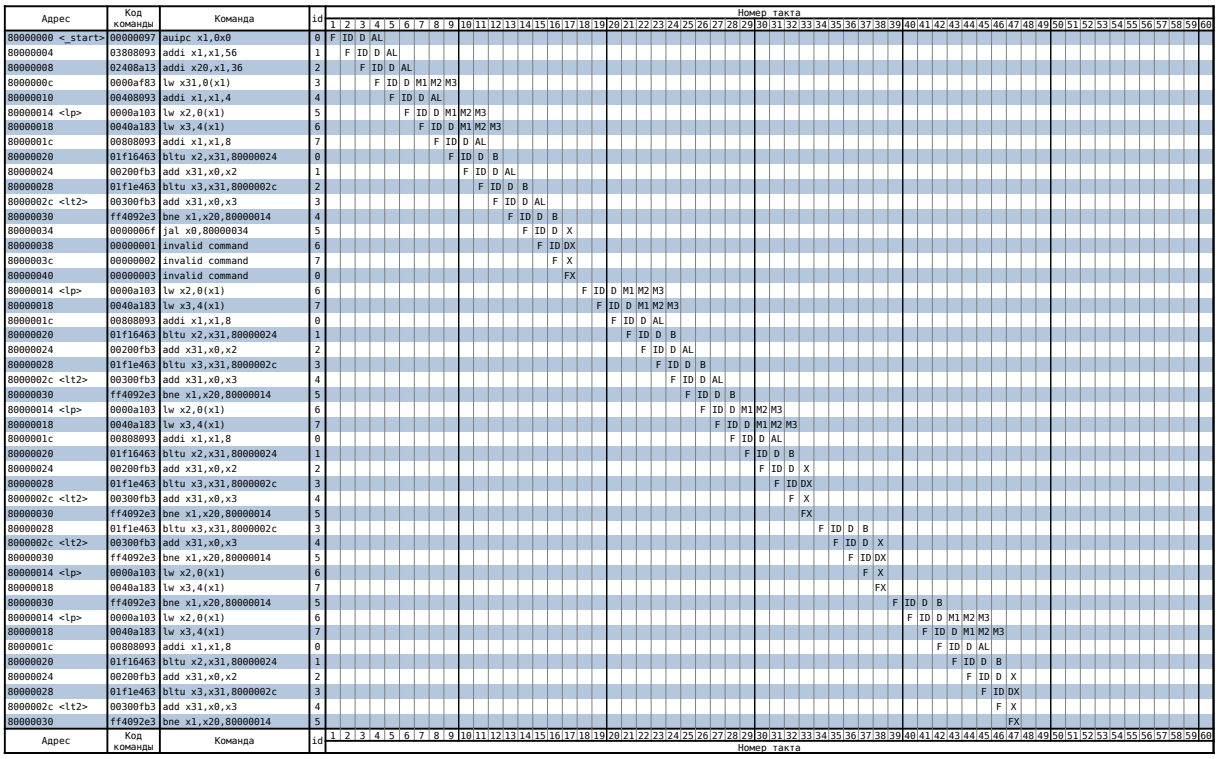

Рисунок 7 – Трасса выполнения оптимизированной программы

Можно заметить, что конфликтов больше нет, а трасса стала на два такта короче (на 4%).

7 Вывод

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

На основе изученных материалов удалось оптимизировать программу.