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 № 1

Название Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина Архитектура элекронно-вычислительных машин

Студент:	_____	Фам М. Х.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Попов А. Ю.
	подпись, дата	Фамилия, И. О.

Москва — 2023 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

```
    .section .text
    .globl _start;
    len = 8
    enroll = 4
    elem_sz = 4
_start:
    addi x20, x0, len/enroll
    la x1, _x
loop:
    lw x2, 0(x1)
    add x31, x31, x2
    lw x2, 4(x1)
    add x31, x31, x2
    lw x2, 8(x1)
    add x31, x31, x2
    lw x2, 12(x1)
    add x31, x31, x2
    addi x1, x1, elem_sz*enroll
    addi x20, x20, -1
    bne x20, x0, loop
    addi x31, x31, 1
forever: j forever

    .section .data
_x: .4byte 0x1
    .4byte 0x2
    .4byte 0x3
    .4byte 0x4
    .4byte 0x5
    .4byte 0x6
    .4byte 0x7
    .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 20.

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

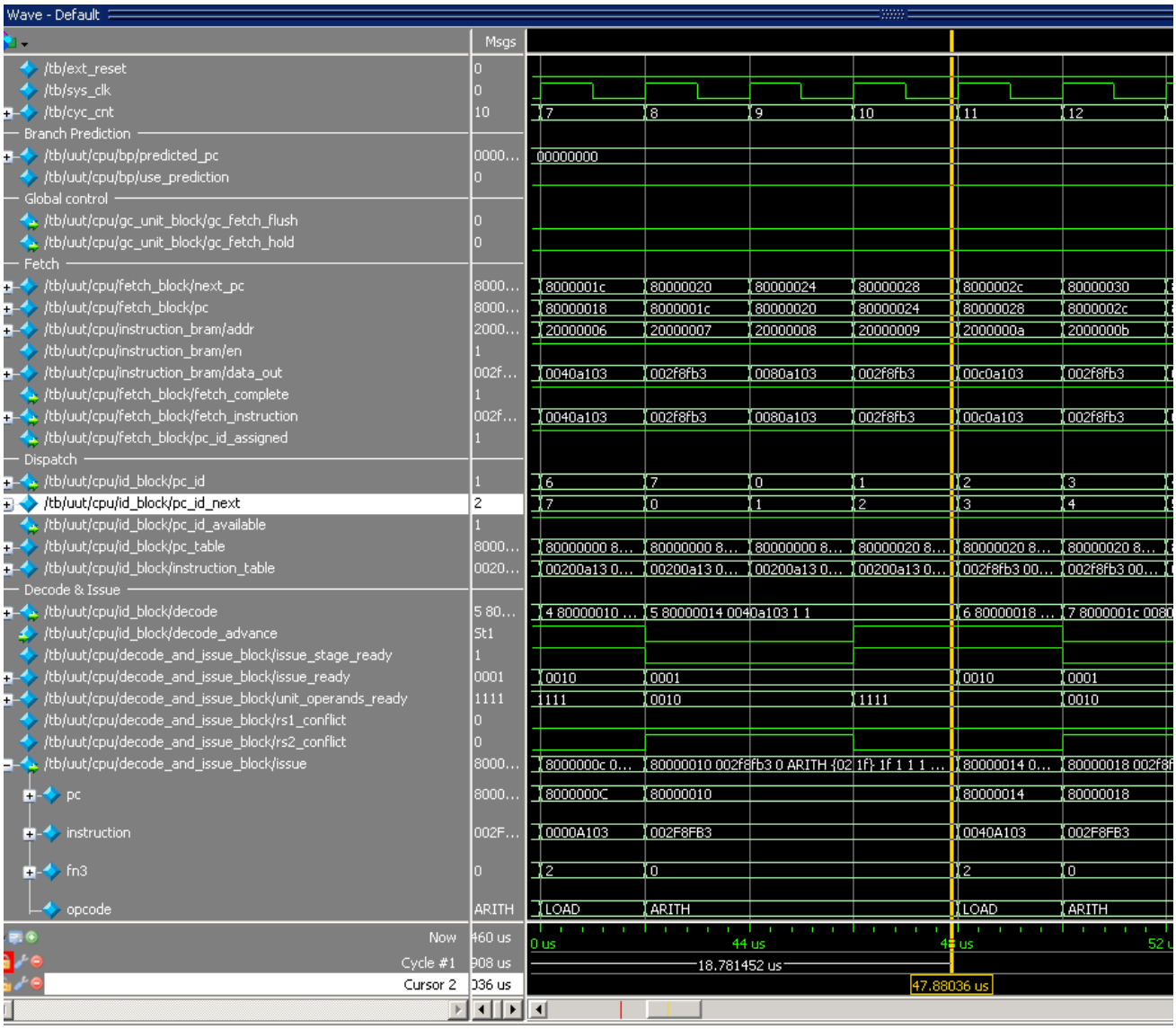

Рисунок 1 – Задание №2

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

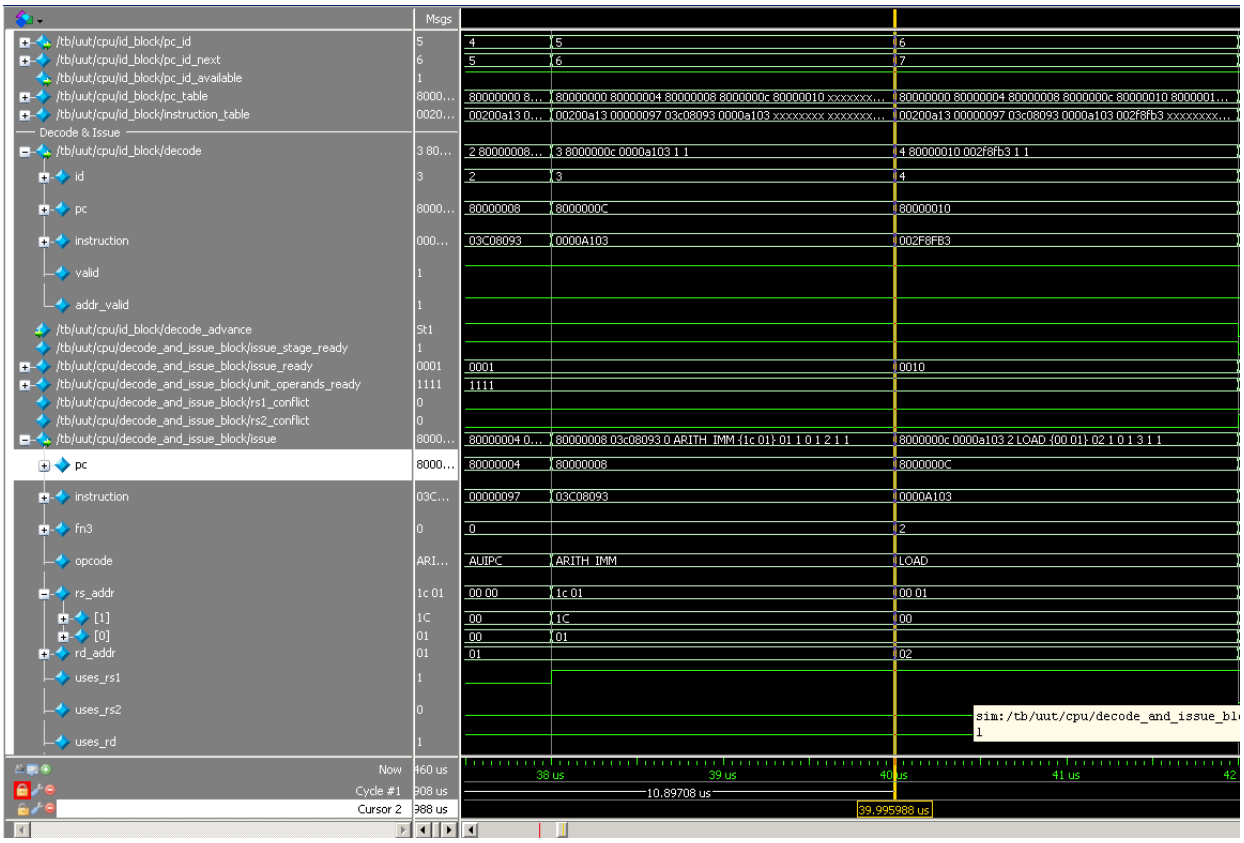

Рисунок 2 – Задание №3

Задание №4

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

Результат представлен на рисунке 3

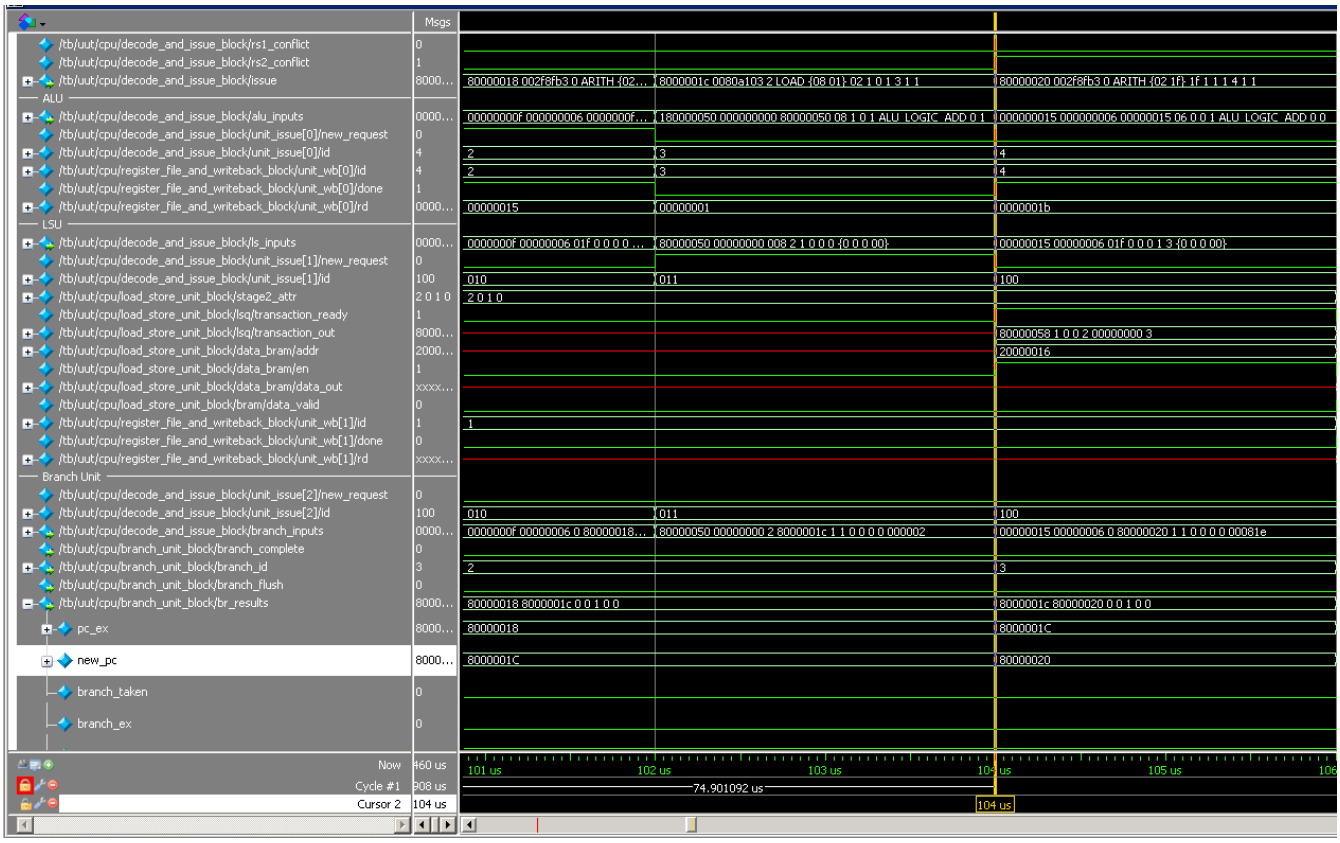

Рисунок 3 – Задание №4

Программа по варианту

Исследуемая программа

Код программы представлен в листинте 4

Листинг 4 – Код программы 20 варианта

```
1           .section .text
2   .globl _start;
3   len = 9
4   enroll = 4
5   elem_sz = 4
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          add x1, x1, elem_sz*enroll
16          addi x20, x20, -1 #!
17          bltu x2, x31, lt1
18          add x31, x0, x2
19  lt1:    bltu x3, x31, lt2
20          add x31, x0, x3
21  lt2:    lw x4, 8(x1)
22          lw x5, 12(x1)
23          bltu x4, x31, lt3
24          add x31, x0, x4
25  lt3:    bltu x5, x31, lt4
26          add x31, x0, x5
27  lt4:
28          bne x20, x0, lp
29  lp2: j lp2
30
31          .section .data
32          _x:     .4byte 0x1
33          .4byte 0x2
```

```
34          .4byte 0x3
35          .4byte 0x4
36          .4byte 0x5
37          .4byte 0x6
38          .4byte 0x5
39          .4byte 0x8
40          .4byte 0x9
```

Дизассемблерный код представлен на листинге 5.

Листинг 5 – Дизассемблированный код 20 варианта

```
1     Disassembly of section .text:
2
3     80000000 <_start>:
4     80000000: 00000097          auipc x1,0x0
5     80000004: 05408093          addi x1,x1,84 # 80000054 <_x>
6     80000008: 00200a13          addi x20,x0,2
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: 0080a203          lw x4,8(x1)
23    80000034: 00c0a283          lw x5,12(x1)
24    80000038: 01f26463          bltu x4,x31,80000040 <lt3>
25    8000003c: 00400fb3          add x31,x0,x4
26
27    80000040 <lt3>:
28    80000040: 01f2e463          bltu x5,x31,80000048 <lt4>
29    80000044: 00500fb3          add x31,x0,x5
30
31    80000048 <lt4>:
```

```
80000048: 01008093          addi x1,x1,16
8000004c: fc0a14e3          bne x20,x0,80000014 <lp>

80000050 <lp2>:
80000050: 0000006f          jal x0,80000050 <lp2>
```

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

Листинг 6 – Псевдокод программы 20 варианта

```
    #define len 8
#define enroll 4
#define elem_sz 4
int _x[]={1,2,3,4,5,6,5,8,9};
void _start() {
    int *x1 = _x;
    int x20 = (len - 1) / enroll;
    int x31 = x1[0];
    x1 += 1;

    do {
        int x2 = x1[0];
        int x3 = x1[1];
        x1 += enroll;
        x20--;
        if (x2 > x31)
        x31 = x2;
        if (x3 > x31)
        x31 = x3;
        int x4 = x1[2];
        int x5 = x1[3];
        if (x4 > x31)
        x31 = x4;
        if (x5 > x31)
        x31 = x5;
    } while(x20 != 0);
    while(1){}
}
```

Трасса работы программы

Трасса работы представлена на рисунке 7.

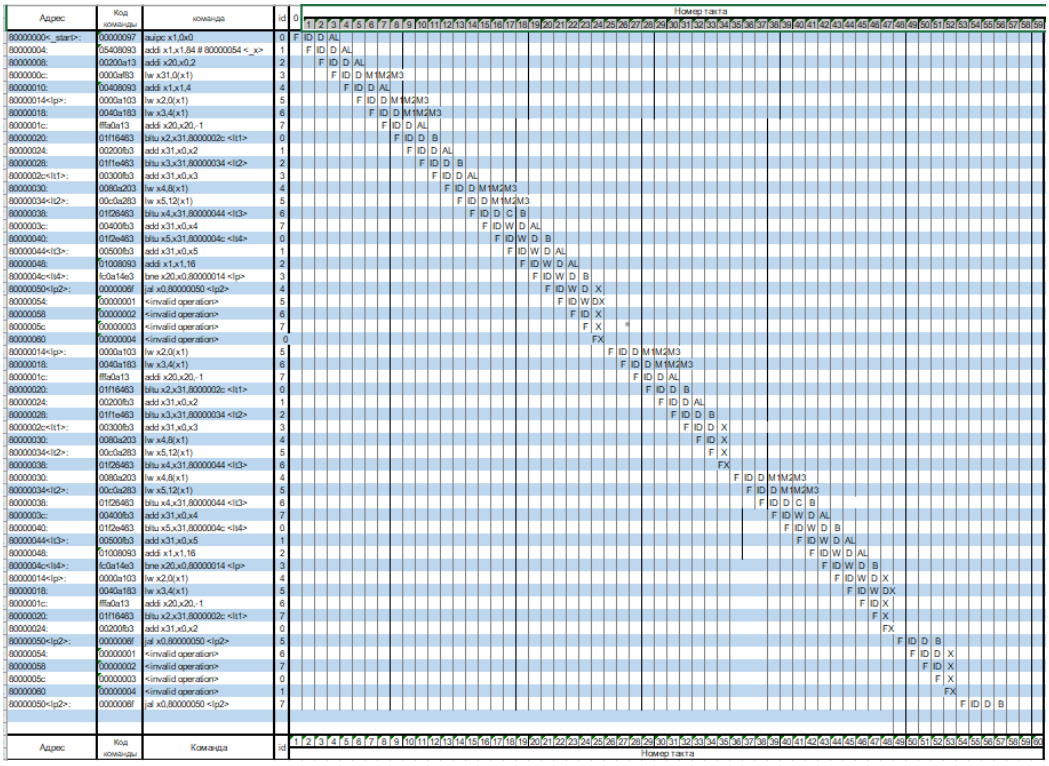

Рисунок 4 – Трасса выполнения программы

Временные диаграммы

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

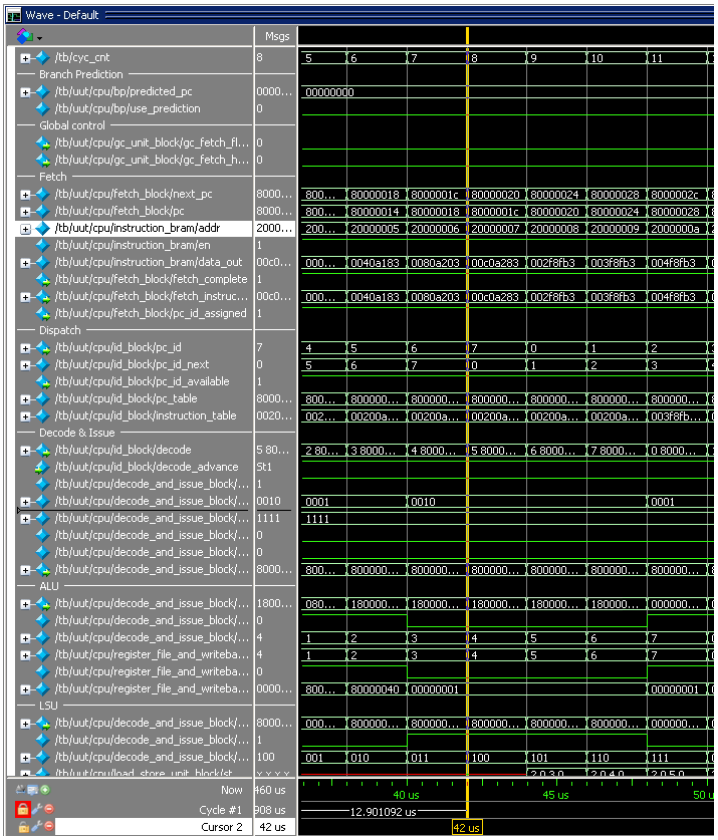

Рисунок 5 – Временные диаграммы сигналов

Вывод и предложение по оптимизации

Конфликт возникает из-за того, что команда загрузки памяти не завершил выполнение, а команда ветвления использовал x4 в качестве параметра.

Оптимизировать программы можно тем, что вставить одну команду между командой загрузки памяти и командой ветвления.

В итоге, можно будет уменьшить программу на 2 такта в программе, то есть на $2/48 = 4\%$ программа будет работать быстрее.

Оптимизированная программа

Код программы представлен в листинте 7

Листинг 7 – Код программы 20 варианта(оптимизированный)

```
1           .section .text
2  .globl _start;
3  len = 9
4  enroll = 4
5  elem_sz = 4
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 add x1, x1, elem_sz*enroll
16 addi x20, x20, -1 #!
17 bltu x2, x31, lt1
18 add x31, x0, x2
19 lt1:     bltu x3, x31, lt2
20 add x31, x0, x3
21 lt2:     lw x4, 8(x1)
22 lw x5, 12(x1)
23 bltu x4, x31, lt3
24 add x31, x0, x4
25 lt3:     bltu x5, x31, lt4
26 add x31, x0, x5
27 lt4:
28 bne x20, x0, lp
29 lp2:     j lp2
30
31 .section .data
32 _x:      .4byte 0x1
33 .4byte 0x2
34 .4byte 0x3
35 .4byte 0x4
36 .4byte 0x5
```

```
37 .4byte 0x6
38 .4byte 0x5
39 .4byte 0x8
40 .4byte 0x9
```

Дизассемблерный код представлен на листинге 8.

Листинг 8 – Дизассемблированный код 20 варинта (оптимизированный)

```
1  80000000 <_start>:
2  80000000:  00000097          auipc x1,0x0
3  80000004:  05408093          addi x1,x1,84 # 80000054 <_x>
4  80000008:  00200a13          addi x20,x0,2
5  8000000c:  0000af83          lw x31,0(x1)
6  80000010:  00408093          addi x1,x1,4
7
8  80000014 <lp>:
9  80000014:  0000a103          lw x2,0(x1)
10 80000018:  0040a183          lw x3,4(x1)
11 8000001c:  fffa0a13          addi x20,x20,-1
12 80000020:  01f16463          bltu x2,x31,80000028 <lt1>
13 80000024:  00200fb3          add x31,x0,x2
14
15 80000028 <lt1>:
16 80000028:  01f1e463          bltu x3,x31,80000030 <lt2>
17 8000002c:  00300fb3          add x31,x0,x3
18
19 80000030 <lt2>:
20 80000030:  0080a203          lw x4,8(x1)
21 80000034:  00c0a283          lw x5,12(x1)
22 80000038:  01008093          addi x1,x1,16
23 8000003c:  01f26463          bltu x4,x31,80000044 <lt3>
24 80000040:  00400fb3          add x31,x0,x4
25
26 80000044 <lt3>:
27 80000044:  01f2e463          bltu x5,x31,8000004c <lt4>
28 80000048:  00500fb3          add x31,x0,x5
29
30 8000004c <lt4>:
31 8000004c:  fc0a14e3          bne x20,x0,80000014 <lp>
32
33 80000050 <lp2>:
34 80000050:  0000006f          jal x0,80000050 <lp2>
```

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

Листинг 9 – Псевдокод программы 20 варинта (оптимизированный)

```
#define len 8
#define enroll 4
#define elem_sz 4
int _x[]={1,2,3,4,5,6,5,8,9};
void _start() {
    int *x1 = _x;
    int x20 = (len - 1) / enroll;
    int x31 = x1[0];
    x1 += 1;

    do {
        int x2 = x1[0];
        int x3 = x1[1];
        x1 += enroll;
        if (x2 > x31)
        x31 = x2;
        if (x3 > x31)
        x31 = x3;
        int x4 = x1[2];
        int x5 = x1[3];
        x20--;
        if (x4 > x31)
        x31 = x4;
        if (x5 > x31)
        x31 = x5;
    } while(x20 != 0);
    while(1){}
}
```

Трасса работы программы

Трасса работы представлена на рисунке 8.

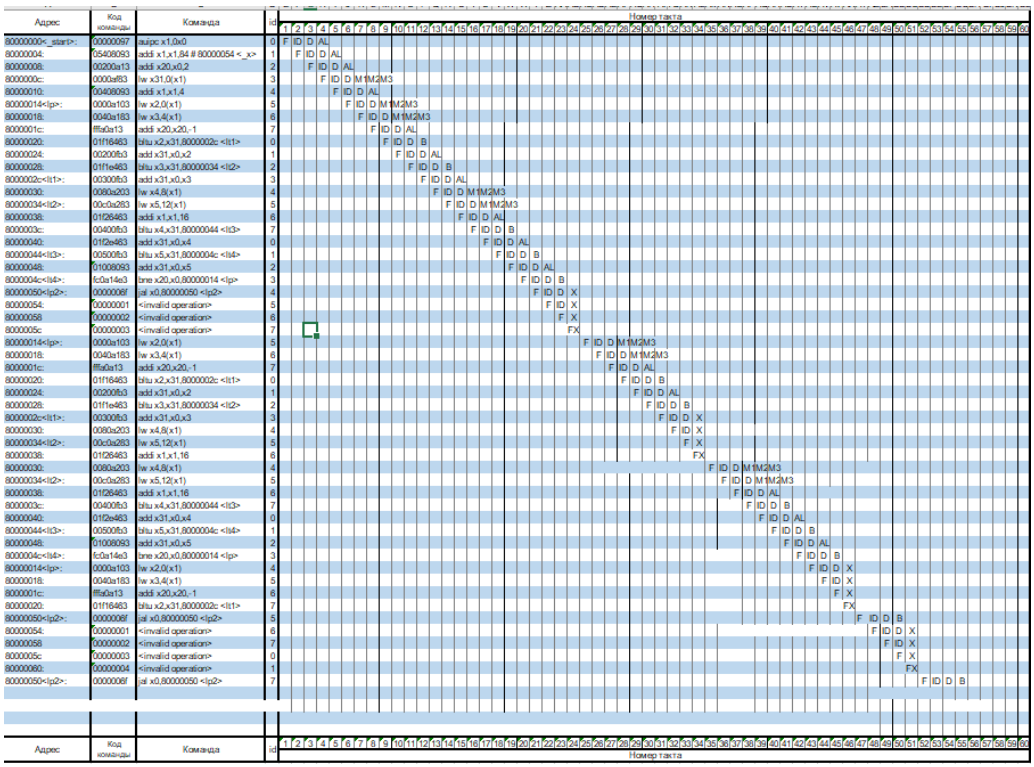

Рисунок 6 – Трасса выполнения программы

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.

Приложение

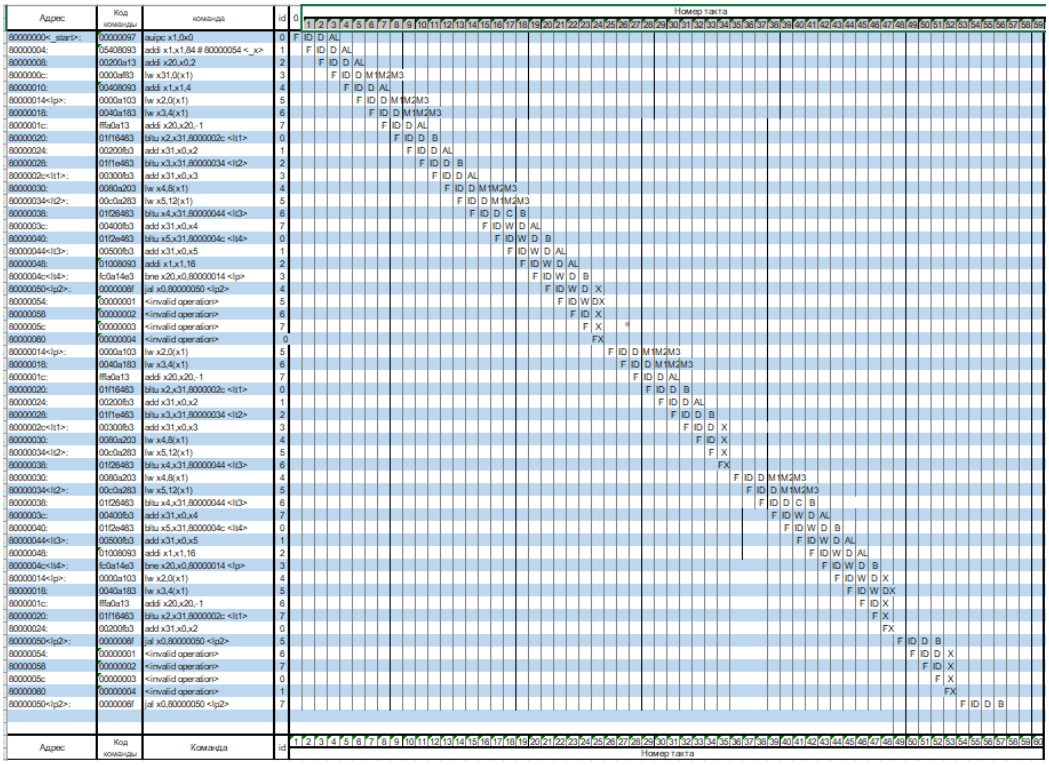

Рисунок 7 – Трасса выполнения программы

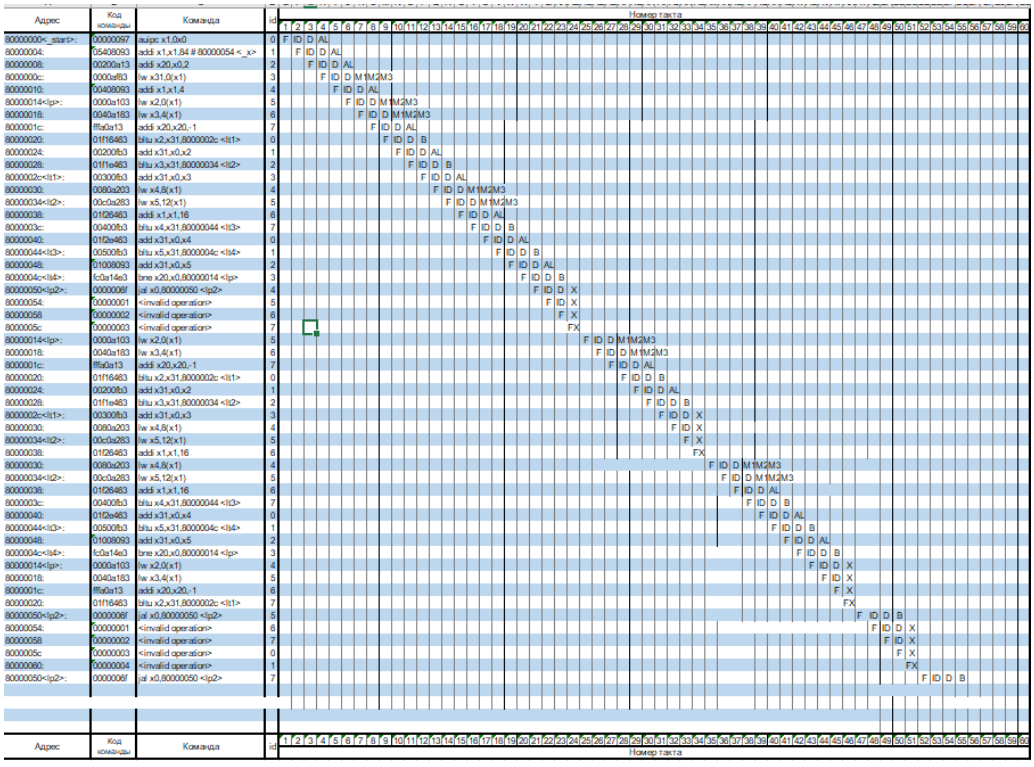

Рисунок 8 – Трасса выполнения программы