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 №1

по курсу «Архитектура ЭВМ»

на тему: «Изучение принципов работы микропроцессорного ядра RISC-V»

Вариант № 17

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) М. А. Семенчук (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) А. Ю. Попов (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

1 Задание №1 **4**

1.1 Текст программы по индивидуальному варианту 4

1.2 Дизассемблерный листинг кода программы 5

1.3 Псевдокод, поясняющий работу программы 6

2 Задание №2 **8**

3 Задание №3 **10**

4 Задание №4 **12**

5 Задание №5 **14**

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[1], реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

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

[1]https://gitlab.com/sfu-rcl/Taiga, авторы - Eric Matthews, Lesley Shannon

1 Задание №1

1.1 Текст программы по индивидуальному варианту

Листинг 1.1 – Текст программы по индивидуальному варианту

```
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, x1, elem_sz*len #Адрес элемента, следующего за
            последним
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        bne x1, x20, lp
22 lp2: j lp2
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
33        .4byte 0x9
```

1.2 Дизассемблерный листинг кода программы

Листинг 1.2 – Дизассемблерный листинг кода программы

```
1  SYMBOL TABLE:
2  80000000 l    d  .text  00000000 .text
3  80000038 l    d  .data  00000000 .data
4  00000000 l    df *ABS*  00000000 prog.o
5  00000009 l       *ABS*  00000000 len
6  00000002 l       *ABS*  00000000 enroll
7  00000004 l       *ABS*  00000000 elem_sz
8  80000038 l       .data  00000000 _x
9  80000014 l       .text  00000000 lp
10 80000024 l       .text  00000000 lt1
11 8000002c l       .text  00000000 lt2
12 80000034 l       .text  00000000 lp2
13 80000000 g       .text  00000000 _start
14 8000005c g       .data  00000000 _end
15
16
17
18 Disassembly of section .text:
19
20 80000000 <_start>:
21 80000000:       00000097                auipc   x1,0x0
22 80000004:       03808093                addi    x1,x1,56 #
      80000038 <_x>
23 80000008:       02408a13                addi    x20,x1,36
24 8000000c:       0000af83                lw      x31,0(x1)
25 80000010:       00408093                addi    x1,x1,4
26
27 80000014 <lp>:
28 80000014:       0000a103                lw      x2,0(x1)
29 80000018:       0040a183                lw      x3,4(x1)
30 8000001c:       01f16463                bltu    x2,x31,80000024
      <lt1>
31 80000020:       00200fb3                add     x31,x0,x2
32
33 80000024 <lt1>:
34 80000024:       01f1e463                bltu    x3,x31,8000002c
      <lt2>
35 80000028:       00300fb3                add     x31,x0,x3
```

```
36
37 8000002c <lt2>:
38 8000002c:        00808093                addi    x1,x1,8
39 80000030:        ff4092e3                bne     x1,x20,80000014
      <lp>
40
41 80000034 <lp2>:
42 80000034:        0000006f                jal     x0,80000034 <lp2>
43
44 Disassembly of section .data:
45
46 80000038 <_x>:
47 80000038:        0001                    .insn   2, 0x0001
48 8000003a:        0000                    .insn   2, 0x
49 8000003c:        0002                    .insn   2, 0x0002
50 8000003e:        0000                    .insn   2, 0x
51 80000040:        00000003                lb      x0,0(x0) # 0
      <enroll-0x2>
52 80000044:        0004                    .insn   2, 0x0004
53 80000046:        0000                    .insn   2, 0x
54 80000048:        0005                    .insn   2, 0x0005
55 8000004a:        0000                    .insn   2, 0x
56 8000004c:        0006                    .insn   2, 0x0006
57 8000004e:        0000                    .insn   2, 0x
58 80000050:        00000007                .insn   4, 0x0007
59 80000054:        0008                    .insn   2, 0x0008
60 80000056:        0000                    .insn   2, 0x
61 80000058:        0009                    .insn   2, 0x0009
62          ...
```

1.3 Псевдокод, поясняющий работу программы

Листинг 1.3 – "Псевдокод поясняющий работу программы"

```
1 #define len 9 // Размер массива
2 #define enroll 2 // Количество обрабатываемых элементов за одну
      итерацию
3 #define elem_sz 4 // Размер одного элемента массива
4
5 int _x[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
6
7 int _start() {
```

```
8       int *x1 = _x;
9       int *x20 = x1 + elem_sz * len; // Адрес элемента, следующего
            за последним
10      int x31 = x1[0];
11      x1 += elem_sz;
12      do {
13          int x2 = x1[0];
14          int x3 = x1[1];
15          if (x2 >= x31)
16              x31 = x2;
17          if (x3 >= x31)
18              x31 = x3;
19          x1 += elem_sz * enroll;
20      } while (x1 != x20);
21      while (1);
22  }
```

Приведенный код находит **максимальный** элемент массива, используя попарное сравнение элементов. Следовательно, после работы программы регистр **x31** будет содержать число **9**.

2 Задание №2

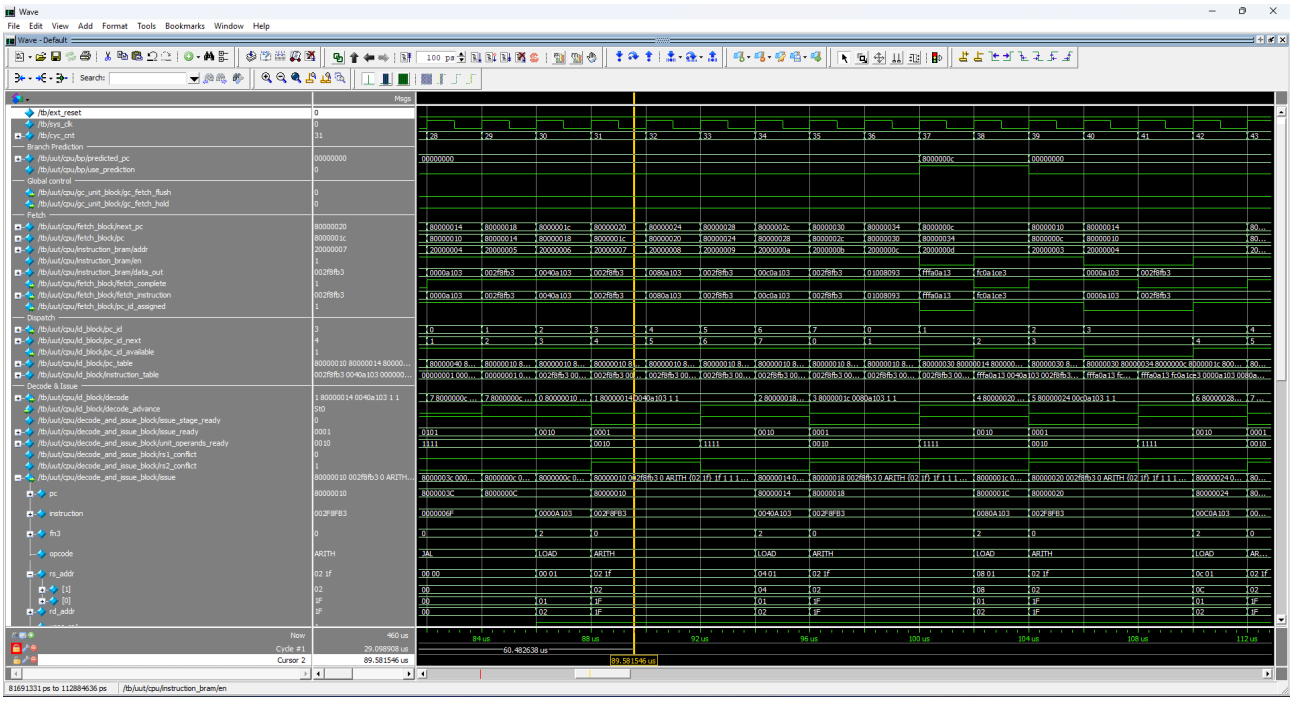

Рисунок 2.1 – Временная диаграмма выполнения стадий выборки и диспетчеризации команды с **адресом 80000020 на 2-й итерации**

3 Задание №3

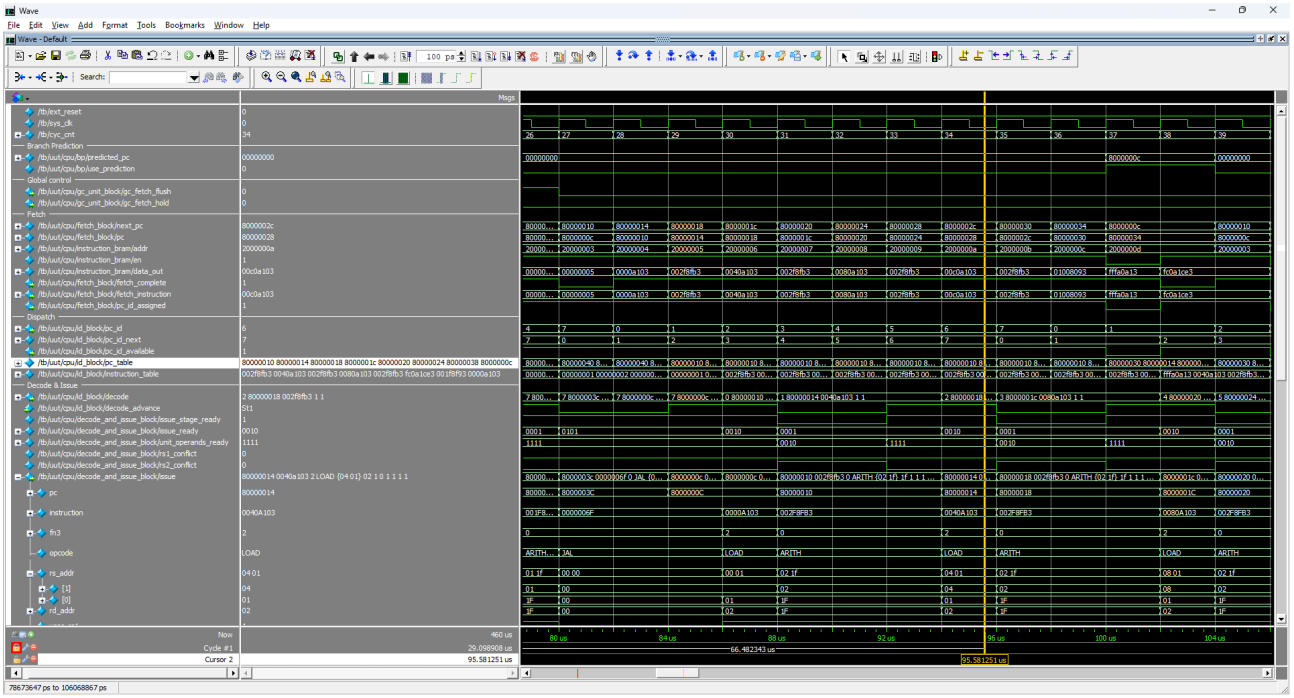

Рисунок 3.1 – Временная диаграмма выполнения стадии декодирования и планирования на выполнение **команды 8000002c на 2-й итерации**

4 Задание №4

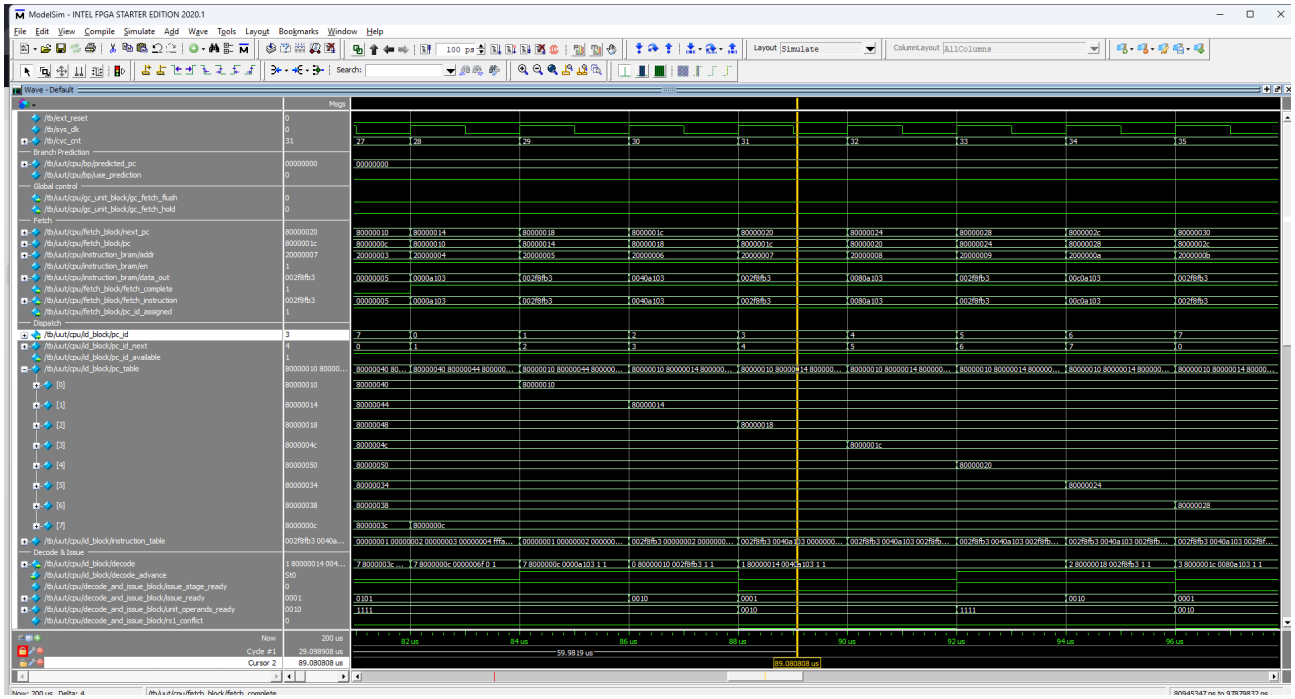

Рисунок 4.1 – Временная диаграмма команды add x31,x31,x2 2-ой итерации на стадии выборки и диспетчеризации

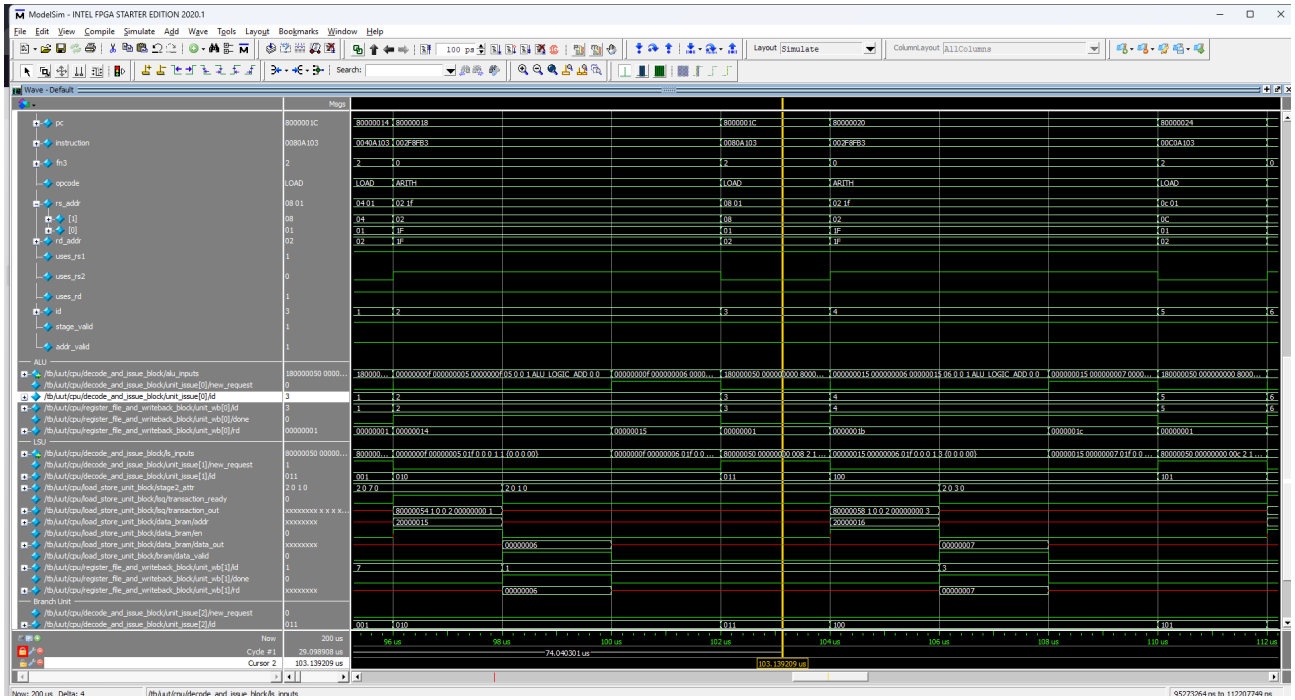

Рисунок 4.2 – Временная диаграмма команды add x31,x31,x2 2-ой итерации на стадии выполнения

5 Задание №5

Значение регистра ‘x31‘ на момент окончания выполнения программы равно 9, как предполагалось ранее.

Программа достаточно хорошо оптимизирована, так как планирование производилось более чем на 75% тактах. Если же попробовать, к примеру, поменять в цикле порядок загрузки значения в регистр и ветвления, то эффективность программы снизится.

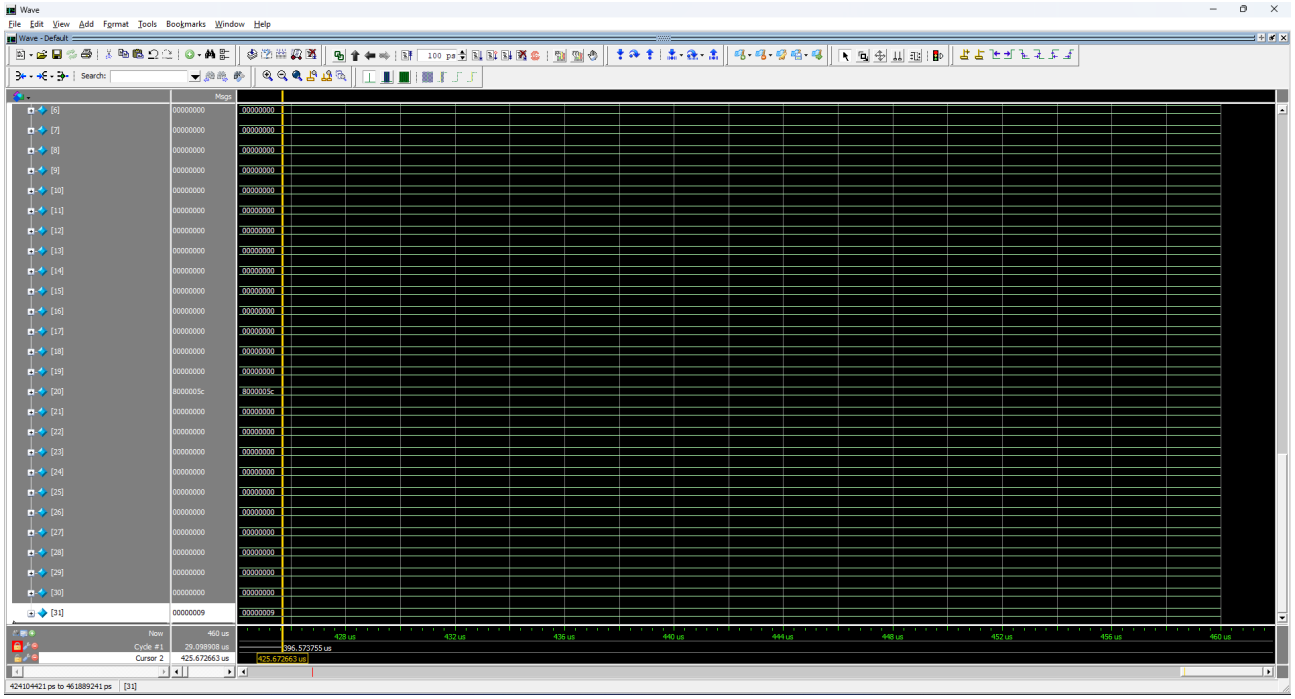

Рисунок 5.1 – Значение регистра x31 на момент окончания выполнения программы

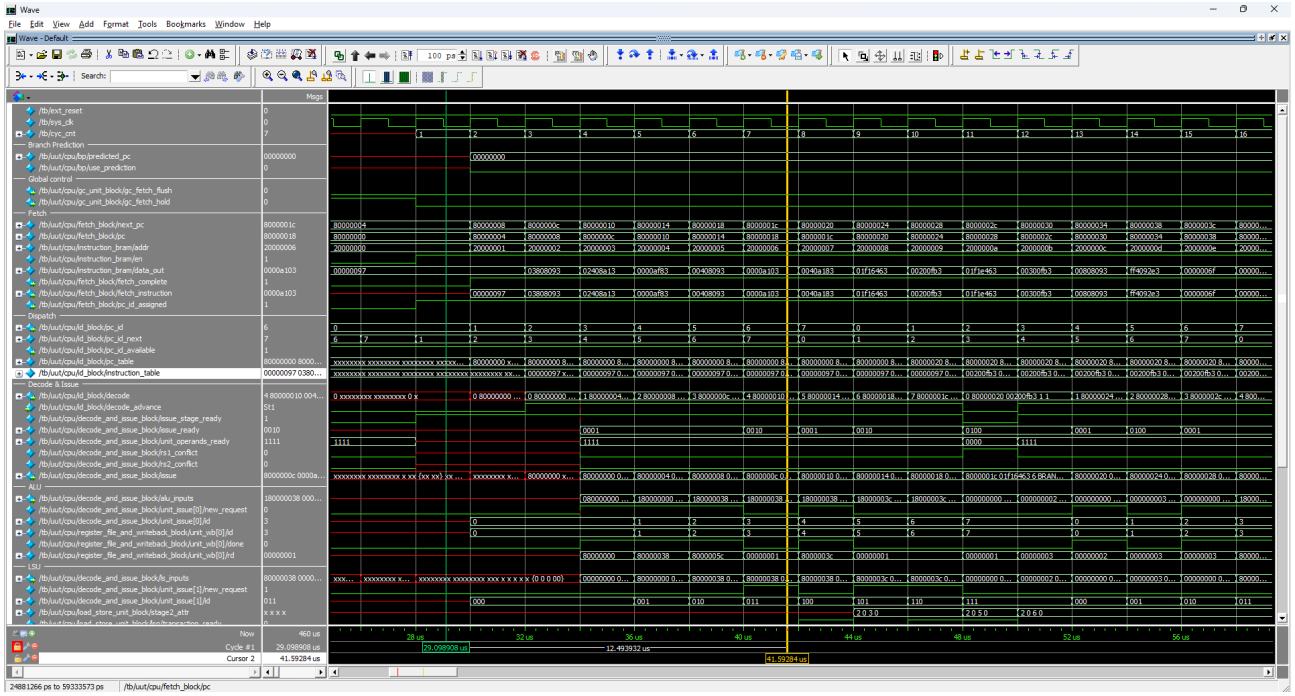

Рисунок 5.2 – Временная диаграмма сигналов команды lw x3, 4(x1) на стадии выборки и диспетчеризации

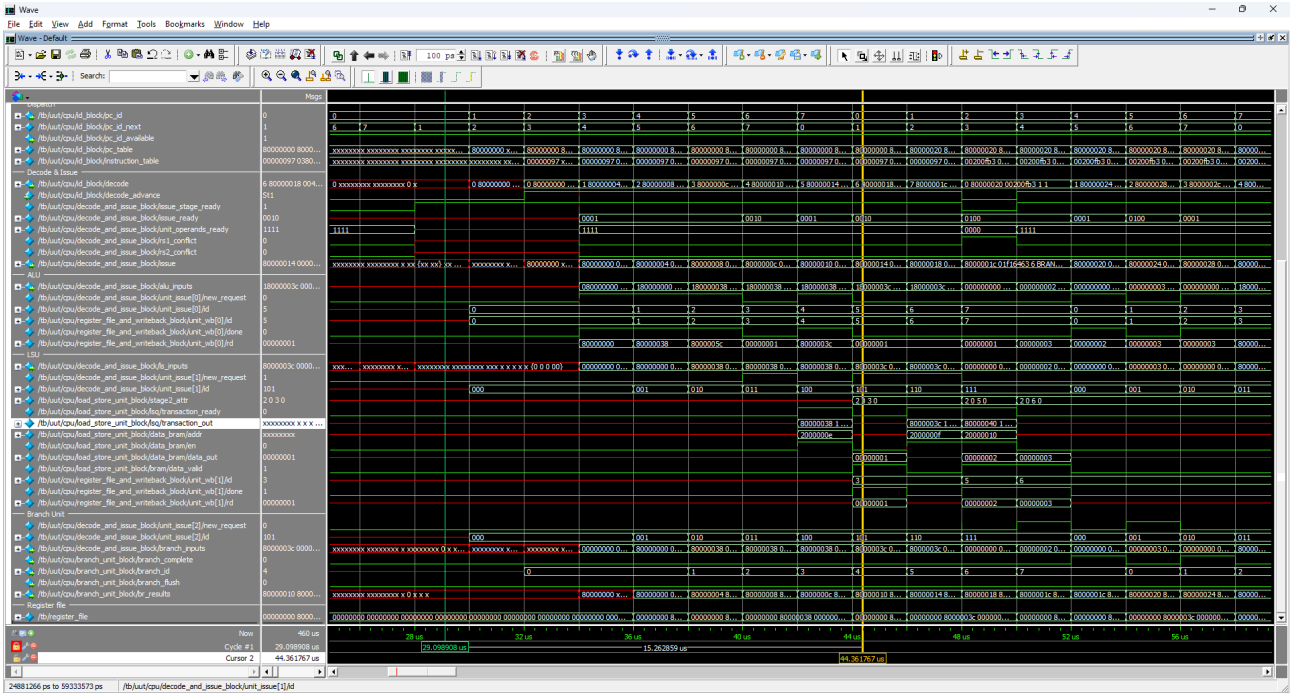

Рисунок 5.3 – Временная диаграмма сигналов команды lw x3, 4(x1) на стадии декодирования

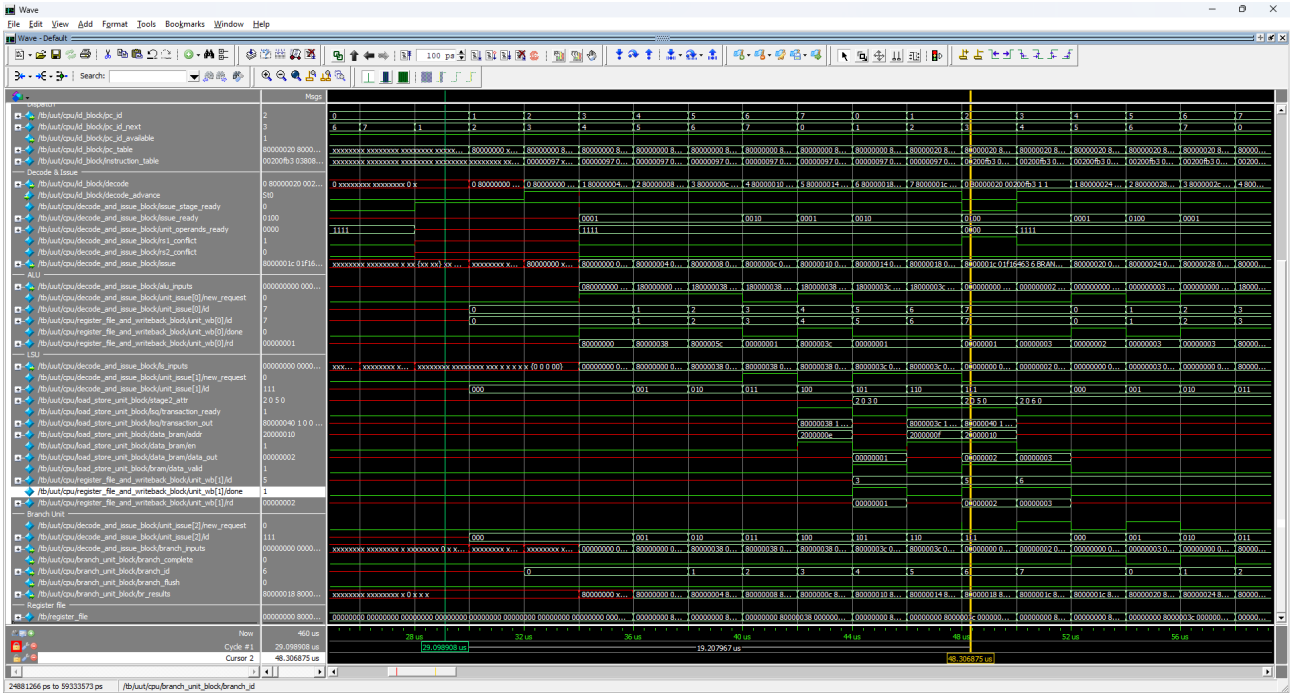

Рисунок 5.4 – Временная диаграмма сигналов команды lw x3, 4(x1) на стадии выполнения

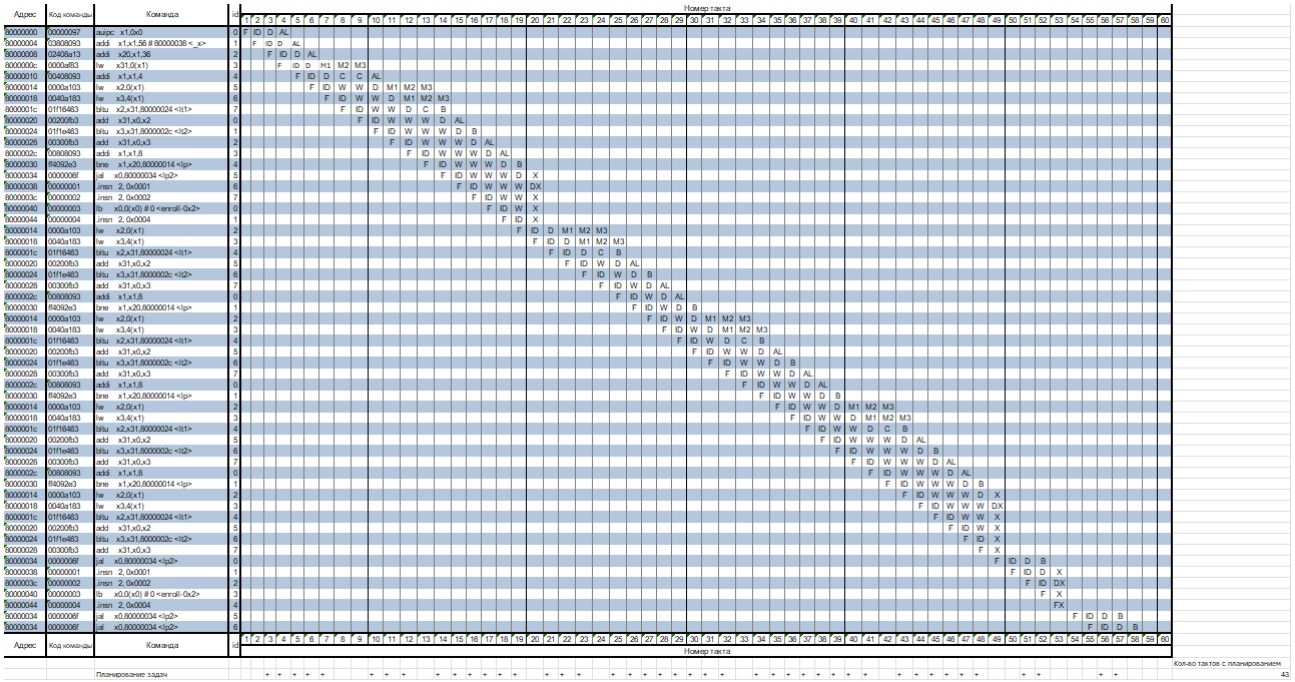

Рисунок 5.5 – Трасса работы программы