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, искусственный интеллект 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 ЭВМ

Студент ИУ7-52Б _____ С.С Беляк
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2024 год

Цель работы – Основной целью работы является ознакомление с принципами функциони- рования, построения и особенностями архитектуры суперскалярных конвейер- ных микропроцессоров. Дополнительной целью работы является знакомство с принципами проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС.

Задание 1

Листинг 1.1. Исходный текст исследуемой программы

```
        .section .text
        .globl _start;
        len = 8 #Размер массива
        enroll = 1 #Количество обрабатываемых элементов за одну итерацию
 elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x1, elem_sz*(len-1) #Адрес последнего элемента
        add x31, x0, x0
lp:
        lw x2, 0(x1)
        add x31, x31, x2 #!
        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
```

Листинг 1.2. Дизассемблерный листинг

```
iscv64-linux-gnu-ld -b elf32-littleriscv -T link.ld task.o -o task.elf
riscv64-linux-gnu-objdump -D -M numeric,no-aliases -t task.elf

task.elf:     формат файла elf32-littleriscv

SYMBOL TABLE:
80000000 l    d  .text 00000000 .text
80000028 l    d  .data 00000000 .data
00000000 l    df *ABS* 00000000 task.o
00000008 l       *ABS* 00000000 len
00000001 l       *ABS* 00000000 enroll
```

```
00000004 l       *ABS* 00000000 elem_sz
80000028 l       .data 00000000 _x
80000010 l       .text 00000000 lp
80000024 l       .text 00000000 lp2
80000000 g       .text 00000000 _start
80000048 g       .data 00000000 _end


Дизассемблирование раздела .text:

80000000 <_start>:
80000000: 00000097           auipc x1,0x0
80000004: 02808093           addi x1,x1,40 # 80000028 <_x>
80000008: 01c08a13           addi x20,x1,28
8000000c: 00000fb3           add x31,x0,x0

80000010 <lp>:
80000010: 0000a103           lw x2,0(x1)
80000014: 002f8fb3           add x31,x31,x2
80000018: 00408093           addi x1,x1,4
8000001c: ff409ae3           bne x1,x20,80000010 <lp>
80000020: 001f8f93           addi x31,x31,1

80000024 <lp2>:
80000024: 0000006f           jal x0,80000024 <lp2>

Дизассемблирование раздела .data:

80000028 <_x>:
80000028: 0001                 c.addi x0,0
8000002a: 0000                 c.unimp
8000002c: 0002                 c.slli64 x0
8000002e: 0000                 c.unimp
80000030: 00000003           lb x0,0(x0) # 0 <enroll-0x1>
80000034: 0004                 0x4
80000036: 0000                 c.unimp
80000038: 0005                 c.addi x0,1
8000003a: 0000                 c.unimp
8000003c: 0006                 c.slli x0,0x1
8000003e: 0000                 c.unimp
80000040: 00000007           0x7
80000044: 0008                 0x8
 ...
```

Листинг 1.3 Псевдокод программы

```
#define len 8
#define enroll 1
#define elem_sz 4
int _x[]={1,2,3,4,5,6,7,8};
void _start() {
    int *x1 = _x;
    int *x20 = x1 + len-1; // т.е. x20 - адрес последнего элемента массива
```

```
8.      int x31 = 0;
9.
10.     do {
11.       int x2 = x1[0];
12.       x31 += x2;
13.       x1 += enroll;
14.     } while(x20 != x1);
15.     x31++;
16.     while(1){}
17. }
```

В конце программы в регистре x31 должна содержаться сумма всех элементов массива, кроме последнего, плюс 1. Таким образом результат будет равен 29 в десятичной системе счисления или 1d в шестнадцатиричной.

Задание 2

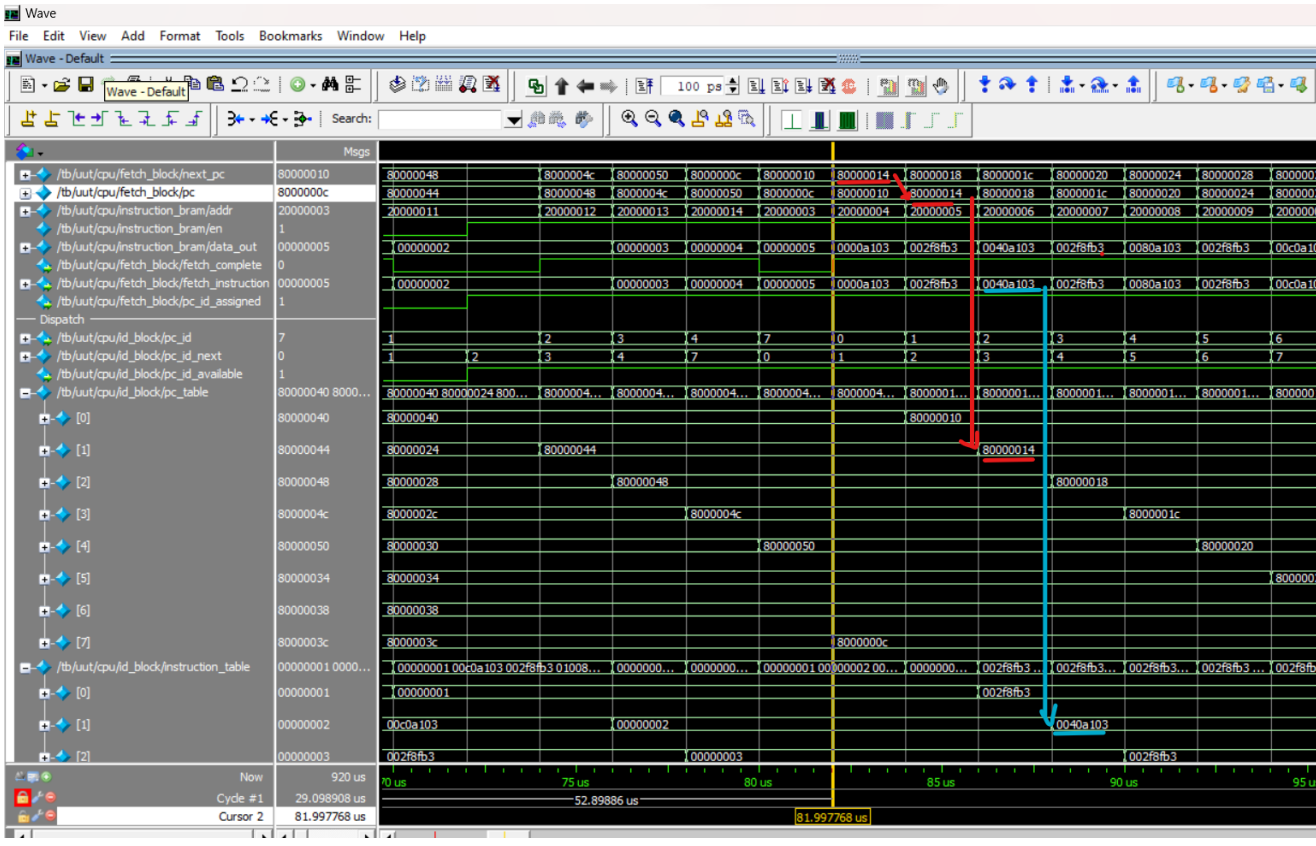

Рисунок 3.1. Временная диаграмма выполнения стадий выборки и диспетчеризации команды с адресом 80000014 для команд, входящих в тело цикла, на 1-ой итерации

Задание 3

Получена временная диаграмма стадии декодирования и планирования на выполнение команды с адресом 80000020 для команд, входящих в тело цикла на 1-ой итерации. Конфликта нет (так как значения rs1_conflict и rs2_conflict равны 0).

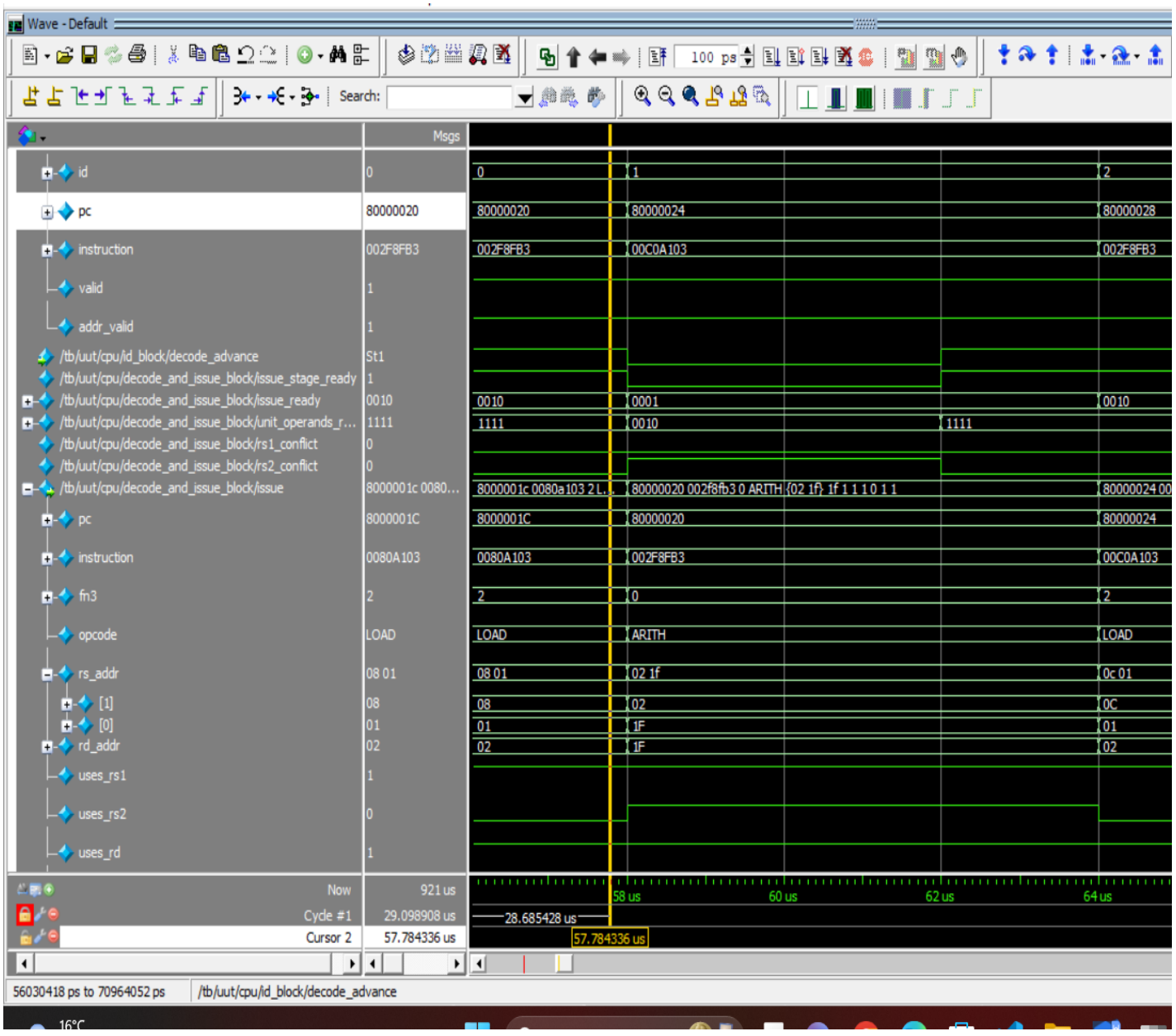

Рисунок 3.2. Временная диаграмма стадии декодирования и планирования на выполнение команды с адресом 80000020 для команд, входящих в тело цикла на 1-ой итерации.

Задание 4

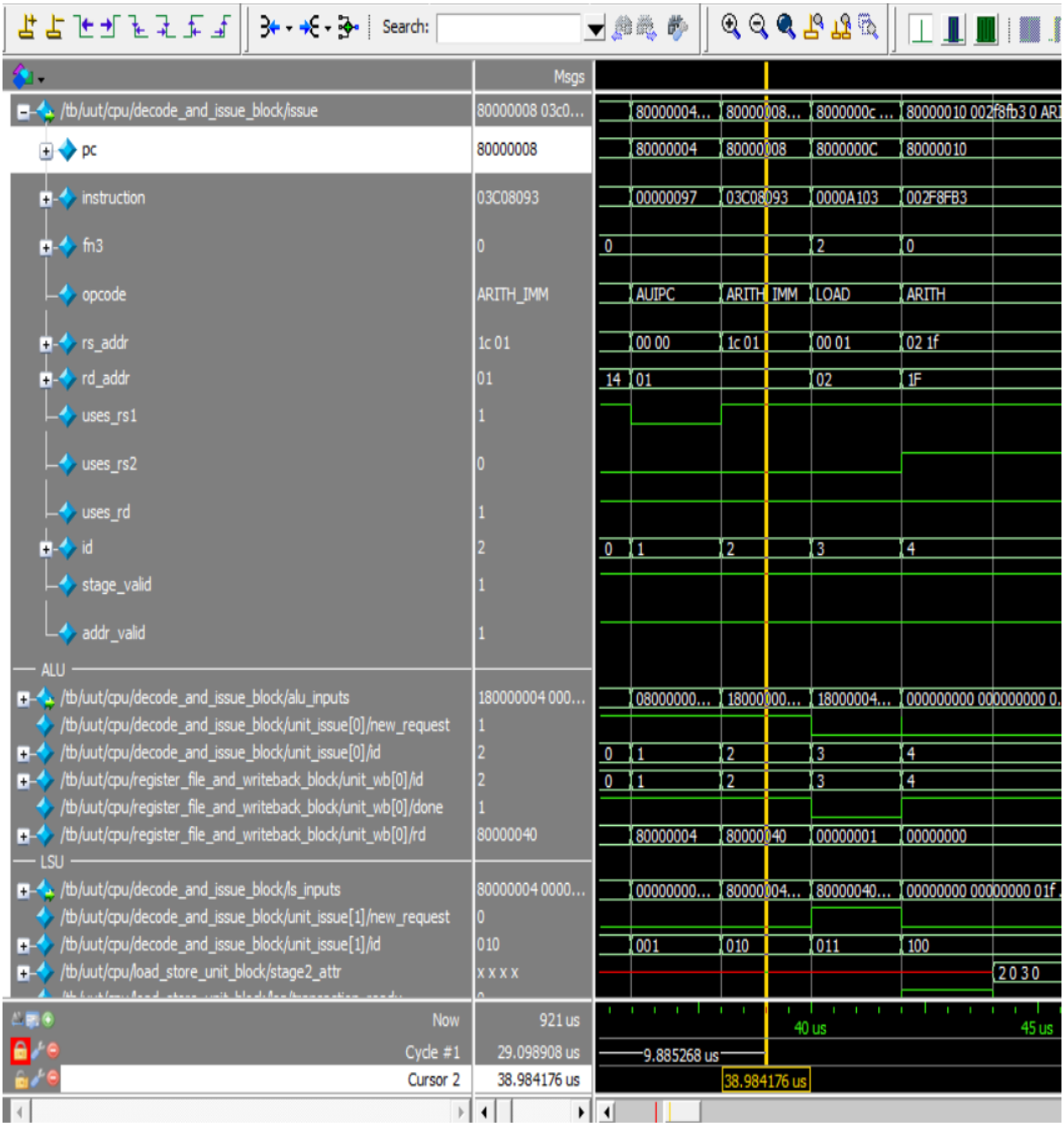

Рисунок 3.3. Временная диаграмма стадии выполнения команды с адресом 80000008.

Задание 5

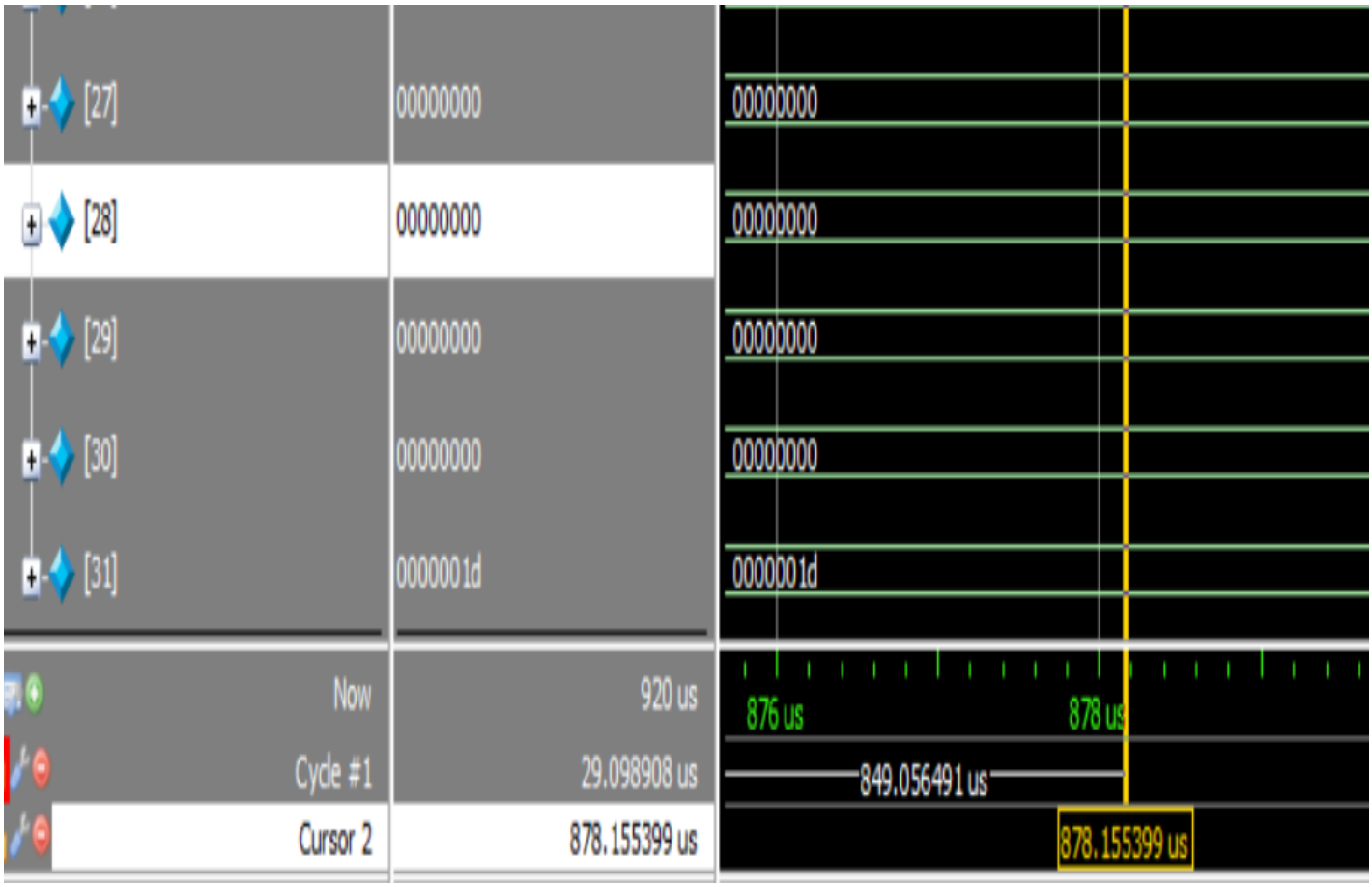

Рисунок 3.4 – Значение регистра x31

Временные диаграммы сигналов, соответствующие всем стадиям выполнения команды add x31, x31, x2 #!.

Из дизассемблированного кода можно узнать адрес данной команды:

```
1.  80000014: 002f8fb3          add x31,x31,x2
```

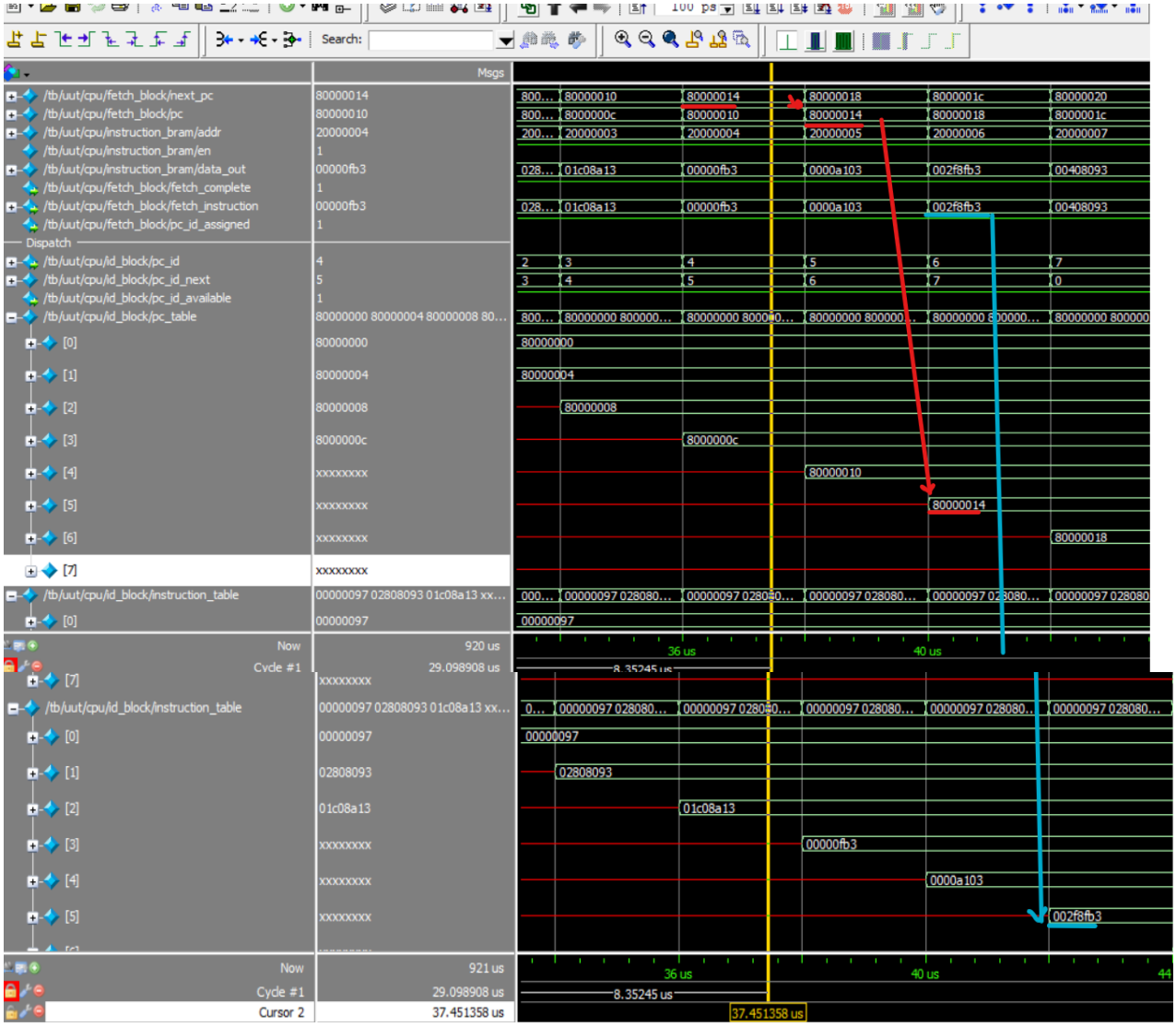

Рисунок 3.5. Временная диаграмма выполнения стадий выборки и диспетчеризации команды с адресом 80000014 для команд, входящих в тело цикла, на 1-ой итерации.

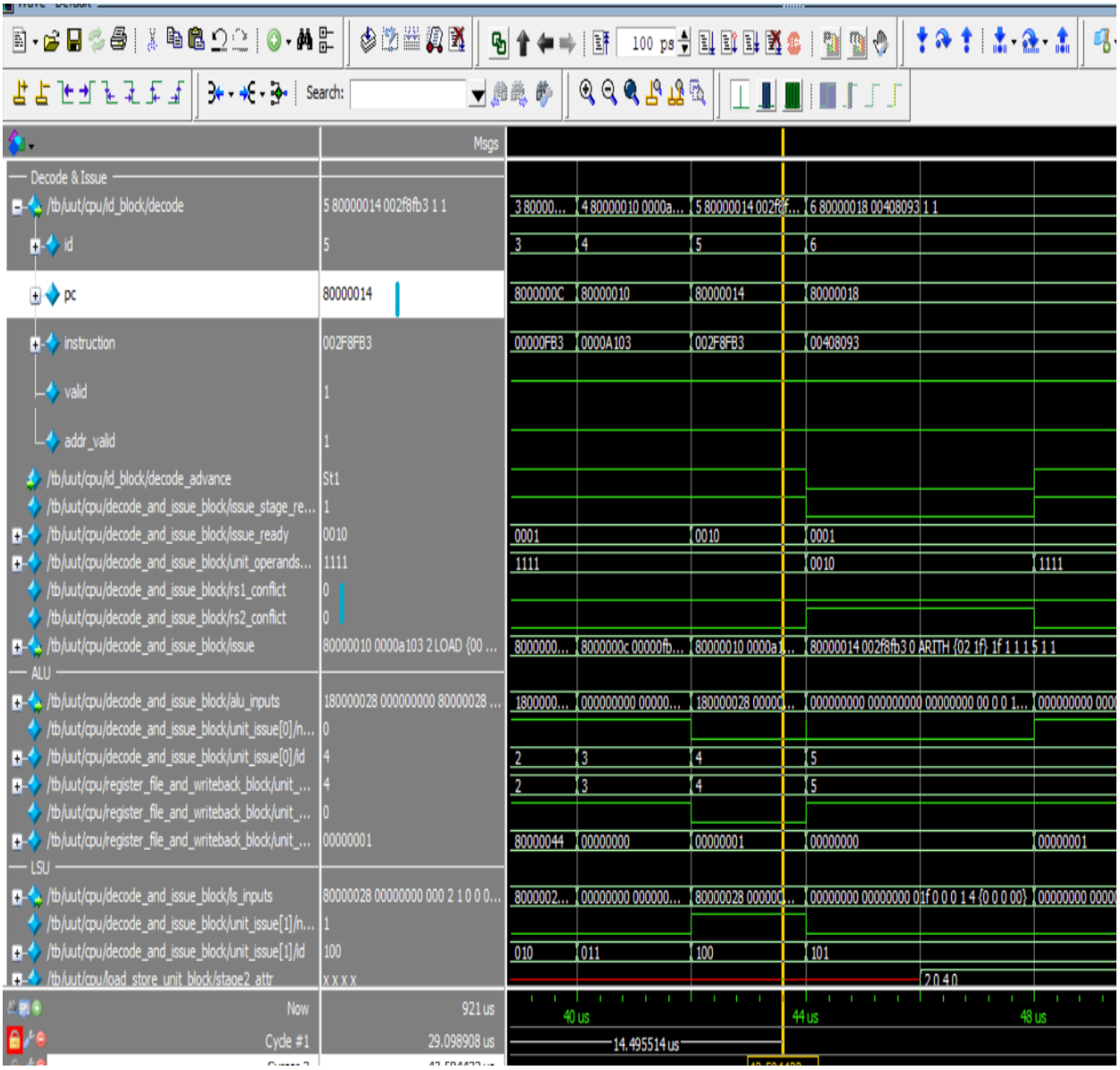

Рисунок 3.6. Временная диаграмма стадии декодирования и планирования на выполнение команды с адресом 80000014 для команд, входящих в тело цикла на 1-ой итерации.

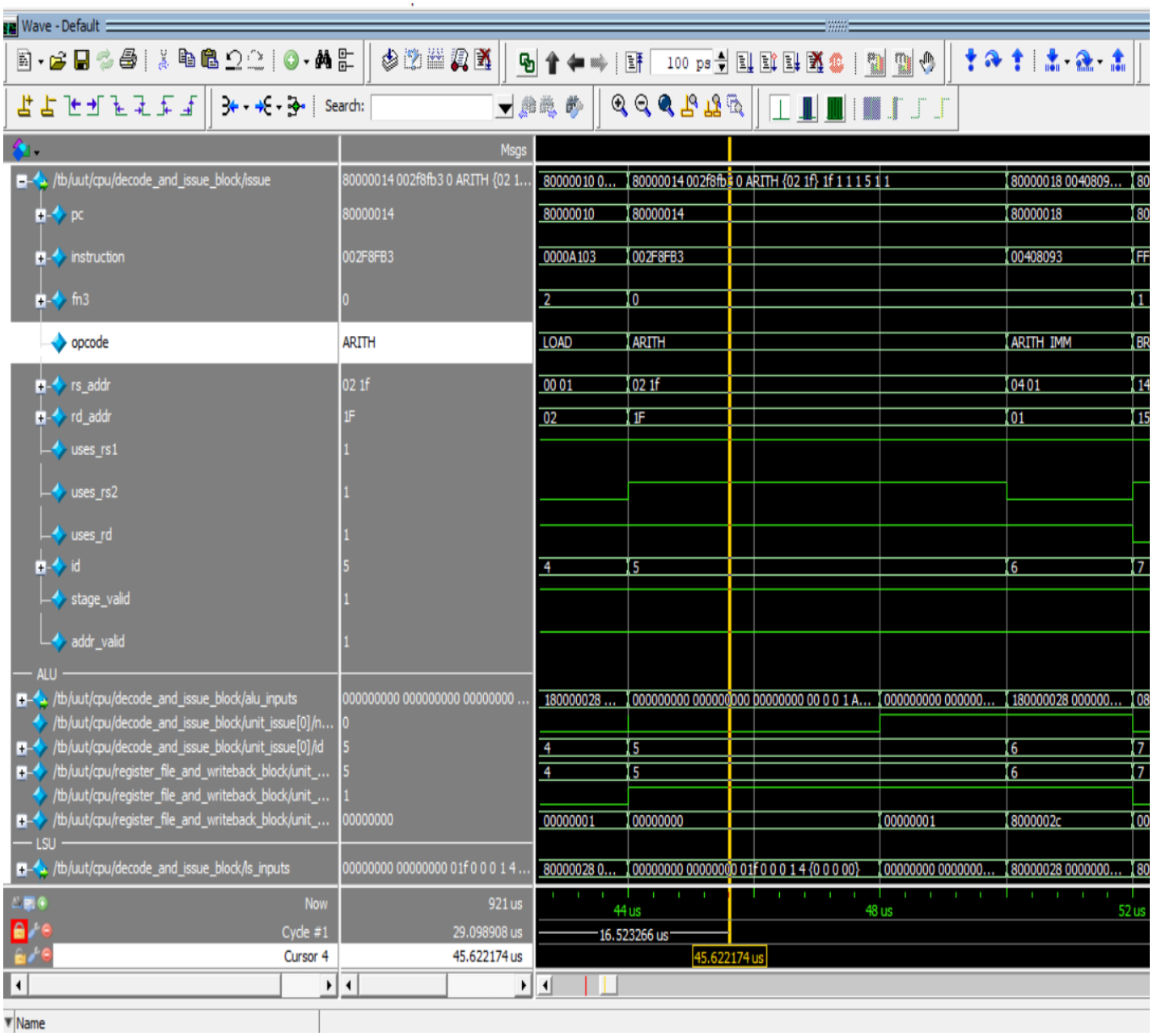

Рисунок 3.7. Временная диаграмма стадии выполнения команды с адресом 80000014 для команд, входящих в тело цикла на 1-ой итерации.

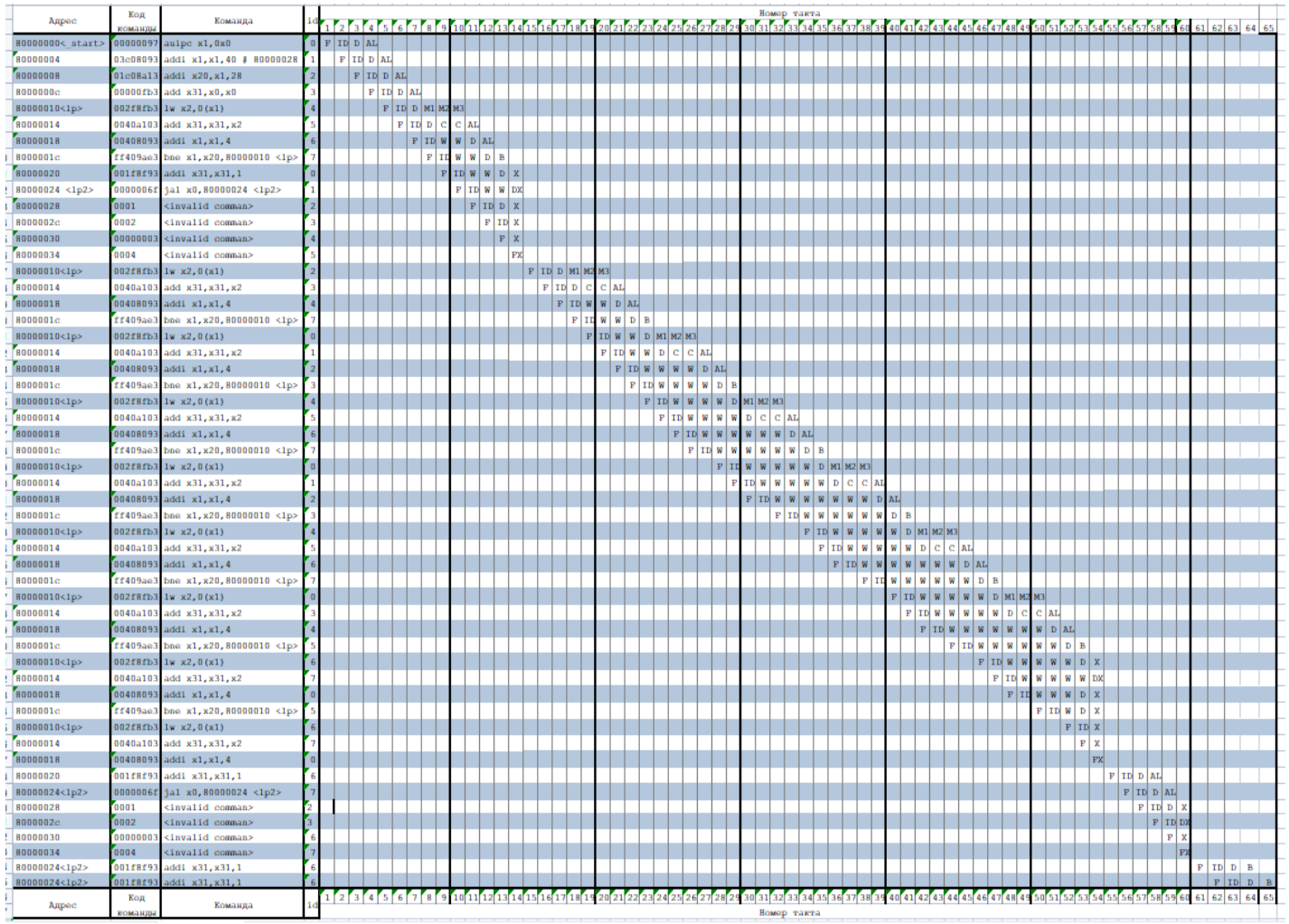

Рисунок 3.8. Трасса неоптимизированной программы

Оптимизация программы

Для выполнения команды `lw x2, 0(x1)` необходимо 3 такта, так как происходит обращение к памяти. Только по прошествии этих 3-ёх тактов можно будет работать с регистром x2. Для оптимизации программы можно поместить команду инкремента счётчика цикла (`addi x1, x1, elem_sz*enroll`) между командами, меняющими регистр x2: `lw x2, 0(x1)` и `add x31, x31, x2.`

В результате изменений, время выполнения уменьшилось с 65 до 58 тактов, предположение оказалосьверным.

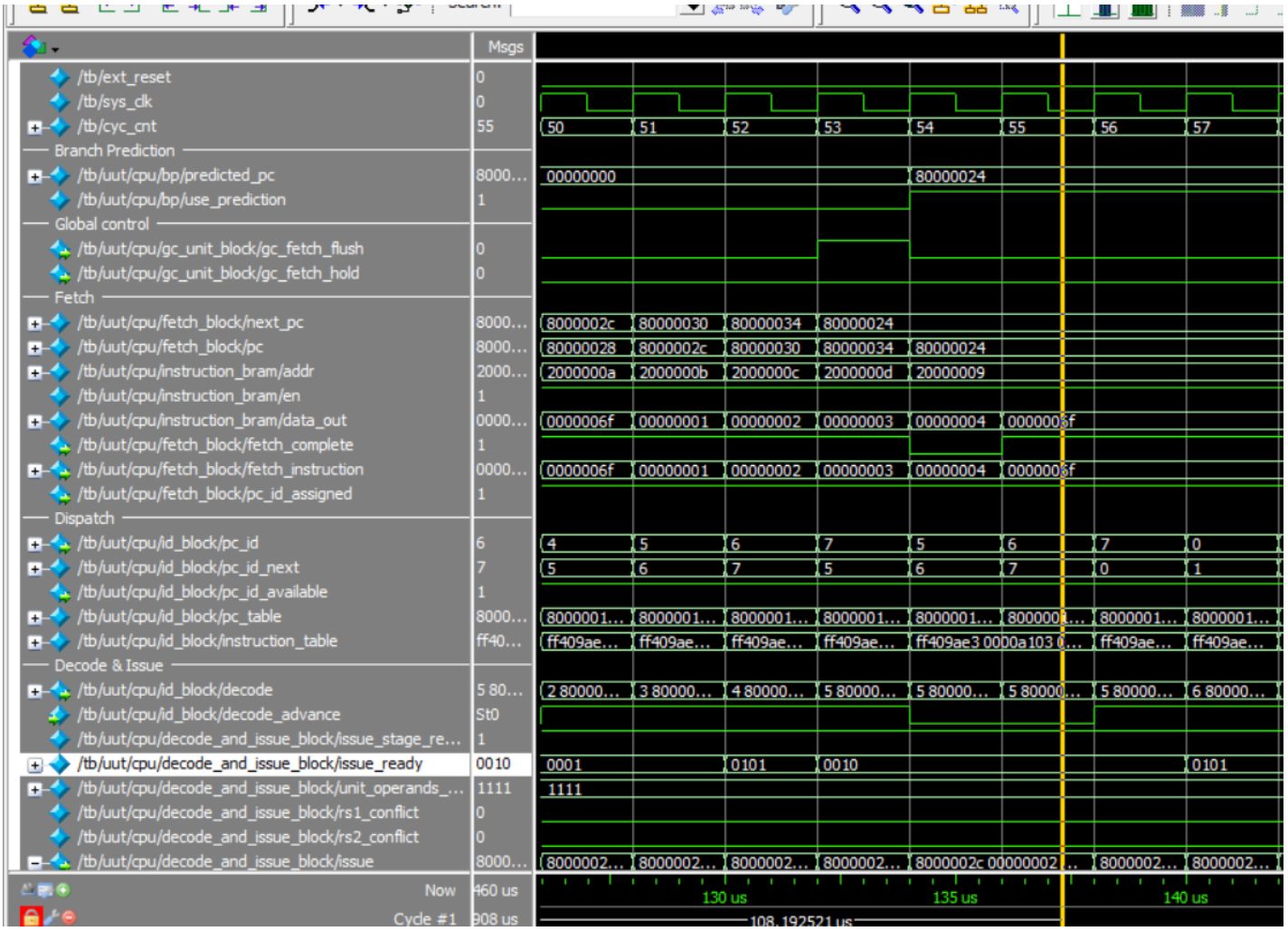

Рисунок 3.9. Завершение программы с оптимизацией

Листинг 2.1. Оптимизированный вариант программы.

```
        .section .text
        .globl _start;
        len = 8 #Размер массива
        enroll = 1 #Количество обрабатываемых элементов за одну итерацию
  elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x1, elem_sz*(len-1) #Адрес последнего элемента
 add x31, x0, x0
lp:
        lw x2, 0(x1)
        addi x1, x1, elem_sz*enroll
        add x31, x31, x2 #!
        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
```

Листинг 2.1. Дизассемблерный листинг оптимизированной программы

```
myTest2.elf:     file format elf32-littleriscv

SYMBOL TABLE:
80000000 l    d  .text  00000000 .text
80000028 l    d  .data  00000000 .data
00000000 l    df *ABS*  00000000 myTest2.o
00000008 l       *ABS*  00000000 len
00000001 l       *ABS*  00000000 enroll
00000004 l       *ABS*  00000000 elem_sz
80000028 l       .data  00000000 _x
80000010 l       .text  00000000 lp
80000024 l       .text  00000000 lp2
80000000 g       .text  00000000 _start
80000048 g       .data  00000000 _end


Disassembly of section .text:

80000000 <_start>:
```

```
80000000:       00000097                auipc   x1,0x0
80000004:       02808093                addi    x1,x1,40 # 80000028 <_x>
80000008:       01c08a13                addi    x20,x1,28
8000000c:       00000fb3                add     x31,x0,x0

80000010 <lp>:
80000010:       0000a103                lw      x2,0(x1)
80000014:       00408093                addi    x1,x1,4
80000018:       002f8fb3                add     x31,x31,x2
8000001c:       ff409ae3                bne     x1,x20,80000010 <lp>
80000020:       001f8f93                addi    x31,x31,1

80000024 <lp2>:
80000024:       0000006f                jal     x0,80000024 <lp2>

Disassembly of section .data:

80000028 <_x>:
80000028:       0001                    c.addi  x0,0
8000002a:       0000                    c.unimp
8000002c:       0002                    c.slli64        x0
8000002e:       0000                    c.unimp
80000030:       00000003                lb      x0,0(x0) # 0 <enroll-0x1>
80000034:       0004                    .2byte  0x4
80000036:       0000                    c.unimp
80000038:       0005                    c.addi  x0,1
8000003a:       0000                    c.unimp
8000003c:       0006                    c.slli  x0,0x1
8000003e:       0000                    c.unimp
80000040:       00000007                .4byte  0x7
80000044:       0008                    .2byte  0x8
```

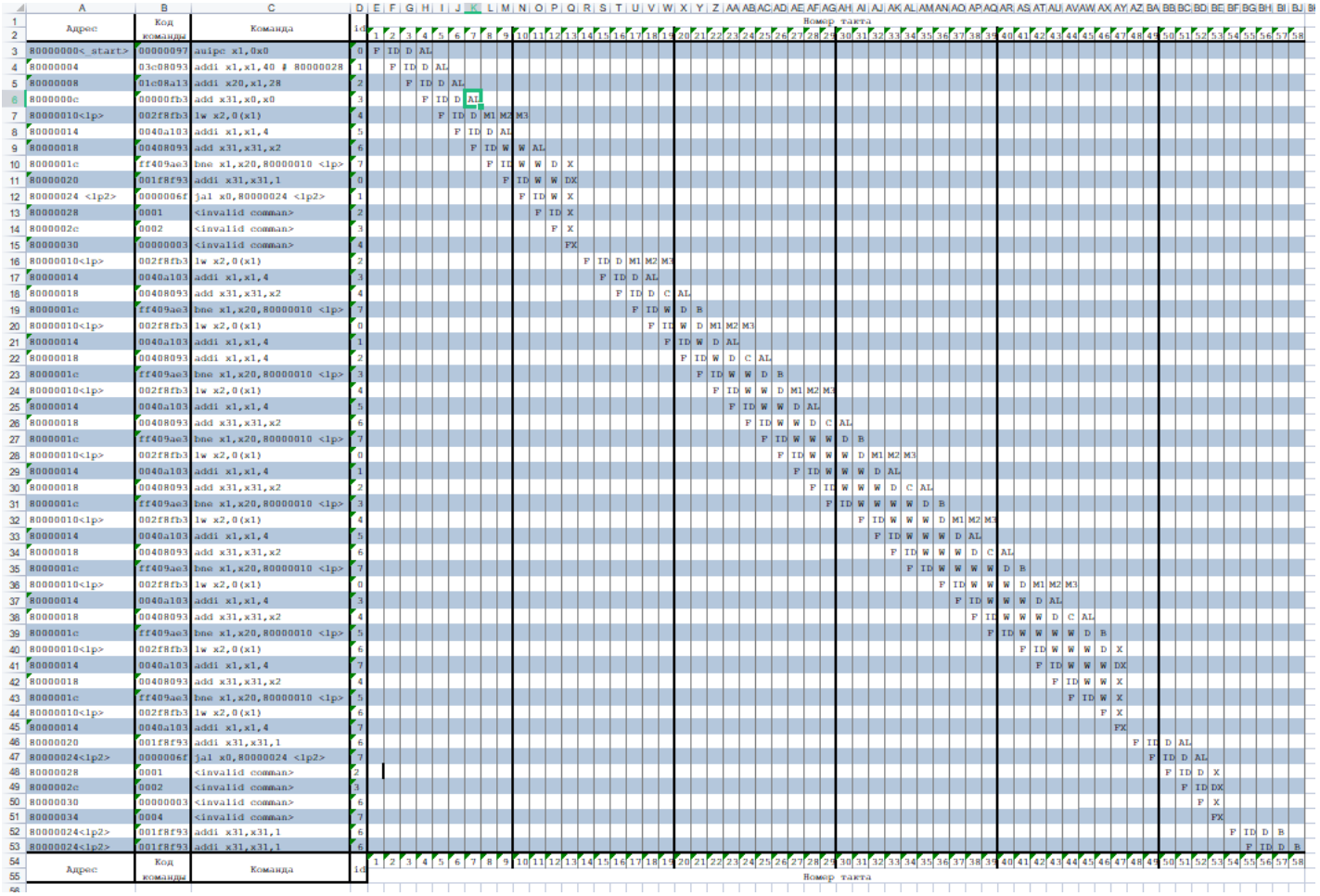

Рисунок 3.10. Трасса оптимизированной программы

Выводы

Выполнение лабораторной работы позволило ознакомиться с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров. В ходе выполнения лабораторной работы были изучены основны микроархитектуры ядра Taiga. Были получены временные диаграммы стадий команд конвейера Taiga. Также удалось оптимизировать программу, уменьшив время выполнения с 65 до 58 тактов.