



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

по курсу «архитектура ЭВМ»

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

Вариант № 18

Студент ИУ7-55Б
(Группа)

(Подпись, дата)

Романов С. К.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Дубровин Е. Н.
(И. О. Фамилия)

2022 г.

Оглавление

1 Введение	3
2 Теоретический раздел	4
3 Практический раздел	5
3.1 Задание 0	5
3.2 Задание 1	7
3.3 Задание 2	11
3.4 Задание 3	12
3.5 Задание 4	13
3.6 Задание 5	14
3.6.1 Результаты выполнения	14
3.6.2 Оптимизация программы	17
3.7 Выводы	22

1 Введение

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.

2 Теоретический раздел

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.

3 Практический раздел

3.1 Задание 0

Дизассемблированный код программы (3.1):

Листинг 3.1 – Программа для задания 0

```
1 SYMBOL TABLE:  
2 80000000 l d .text 00000000 .text  
3 80000040 l d .data 00000000 .data  
4 00000000 l df *ABS* 00000000 test.o  
5 00000008 l *ABS* 00000000 len  
6 00000004 l *ABS* 00000000 enroll  
7 00000004 l *ABS* 00000000 elem_sz  
8 80000040 l .data 00000000 _x  
9 8000000c l .text 00000000 loop  
10 8000003c l .text 00000000 forever  
11 80000000 g .text 00000000 _start  
12 80000060 g .data 00000000 _end  
13  
14  
15  
16 Disassembly of section .text:  
17  
18 80000000 <_start>:  
19 80000000: 00200a13 addi x20,x0,2  
20 80000004: 00000097 auipc x1,0x0  
21 80000008: 03c08093 addi x1,x1,60 # 80000040 <_x>  
22  
23 8000000c <loop>:  
24 8000000c: 0000a103 lw x2,0(x1)  
25 80000010: 002f8fb3 add x31,x31,x2  
26 80000014: 0040a103 lw x2,4(x1)  
27 80000018: 002f8fb3 add x31,x31,x2  
28 8000001c: 0080a103 lw x2,8(x1)  
29 80000020: 002f8fb3 add x31,x31,x2  
30 80000024: 00c0a103 lw x2,12(x1)  
31 80000028: 002f8fb3 add x31,x31,x2  
32 8000002c: 01008093 addi x1,x1,16  
33 80000030: fffa0a13 addi x20,x20,-1  
34 80000034: fc0a1ce3 bne x20,x0,8000000c <loop>  
35 80000038: 001f8f93 addi x31,x31,1  
36  
37 8000003c <forever>:  
38 8000003c: 0000006f jal x0,8000003c <forever>
```

39

40 Disassembly of section .data:

3.2 Задание 1

Листинг 3.2 показывает код программы согласно варианту задания. (№18)

Листинг 3.2 – Программа для задания 1 (вар.18)

```
1 # Variant 18
2 .section .text
3     .globl _start;
4     len = 9 #Размер массива
5     enroll = 2 #Количество обрабатываемых элементов за одну итерацию
6     elem_sz = 4 #Размер одного элемента массива
7
8 _start:
9     la x1, _x
10    addi x20, x1, elem_sz*len #Адрес элемента, следующего за последним
11    lw x31, 0(x1)
12    addi x1, x1, elem_sz*1
13 lp:
14    lw x2, 0(x1) #!
15    lw x3, 4(x1)
16    bltu x2, x31, lt1
17    add x31, x0, x2
18 lt1: bltu x3, x31, lt2
19    add x31, x0, x3
20 lt2:
21    add x1, x1, elem_sz*enroll
22    bne x1, x20, lp
23 lp2: j lp2
24
25     .section .data
26 _x: .4byte 0x1
27     .4byte 0x2
28     .4byte 0x3
29     .4byte 0x4
30     .4byte 0x8
31     .4byte 0x6
32     .4byte 0x7
33     .4byte 0x5
34     .4byte 0x4
```

Псевдокод 0 программы выше (3.2):

```
len ← 9                                ▷ Размер массива
enroll ← 2      ▷ Количество обрабатываемых элементов за одну итерацию
elem_sz ← 4                                ▷ Размер одного элемента массива
_x ← 0x1, 0x2, 0x3, 0x4, 0x8, 0x6, 0x7, 0x5, 0x4
x1 ← _x
x_end ← x1 + len
x31 ← x1[0]
x1 ← x1 + 1
while x1 < x_end do
    x2 ← x[0]
    x3 ← x[1]
    if x2 > x31 then
        x31 ← x2
    end if
    if x3 > x31 then
        x31 ← x3
    end if
    x1 ← x1 + 2
end while
```

Анализируя исходный текст программы значение в регистре x31 в конце выполнения программы равно числу 8 (максимальное число)

Дизассемблированная код программы выше (3.2) находится в листинге 3.3:

Листинг 3.3 – Дизассемблированный код программы для задания 1

```
1 SYMBOL TABLE: 0005 c.addi x0,1
2 80000000 1 d .text 00000000 .text c.unimp
3 80000038 1 d .data 00000000 .data c.slli x0,0x1
4 00000000 1 df *ABS* 00000000 task.o c.unimp
5 00000009 1 *ABS*0700000000 len .4byte 0x7
6 00000002 1 *ABS* 00000000 enroll .2byte 0x8
7 00000004 1 *ABS* 00000000 elem_sz
8 80000038 1 .dataop00000000 _x --reverse-bytes=4 test.elf test.bin
9 80000014 1 .textbi00000000 lp
10 80000024 1 .texttt.00000000 lt1
11 8000002c 1 .text-500000000 lt22/riscv-lab/src | on main ?1 ls
```

```

12      80000034 l .text 00000000 lp2
13      80000000 g .textt.00000000 _start
14      8000005c g .data-500000000 _end/riscv-lab/src | on main ?1 cat test.hex
15      ok
16      00200a13
17      00000097
18      Disassembly of section .text:
19      0000a103
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      002f8fb3
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      00000001
33      80000024 <lt1>:
34      80000024: 01f1e463 bltu x3,x31,8000002c <lt2>
35      80000028: 00300fb3 add x31,x0,x3
36      00000005
37      8000002c <lt2>:
38      8000002c: 00808093 addi x1,x1,8
39      80000030: ff4092e3 bne x1,x20,80000014 <lp>
40      ~/gith/bmstu-comparch-5th-sem/lab_02/riscv-lab/src | on main ?1
41      80000034 <lp2>:
42      80000034: 0000006f jal x0,80000034 <lp2>
43
44      Disassembly of section .data:
45
46      80000038 <_x>:
47      80000038: 0001 c.addi x0,0
48      8000003a: 0000 c.unimp
49      8000003c: 0002 c.slli64 x0
50      8000003e: 0000 c.unimp
51      80000040: 00000003 lb x0,0(x0) # 0 <enroll-0x2>
52      80000044: 0004 .2byte 0x4
53      80000046: 0000 c.unimp
54      80000048: 0008 .2byte 0x8
55      8000004a: 0000 c.unimp
56      8000004c: 0006 c.slli x0,0x1

```

```
57      8000004e: 0000 c.unimp
58      80000050: 00000007 .4byte 0x7
59      80000054: 0005 c.addi x0,1
60      80000056: 0000 c.unimp
61      80000058: 0004 .2byte 0x4
62      ...
```

3.3 Задание 2

В данной секции представлены результаты выполнения выборки и диспетчеризации.

В соответствии с таблицей, приведенной в репозитории, необходимо было получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с указанным адресом (Вариант №18, адрес - 80000024, 2-я итерация). Рисунок 3.1 показывает результат выполнения.

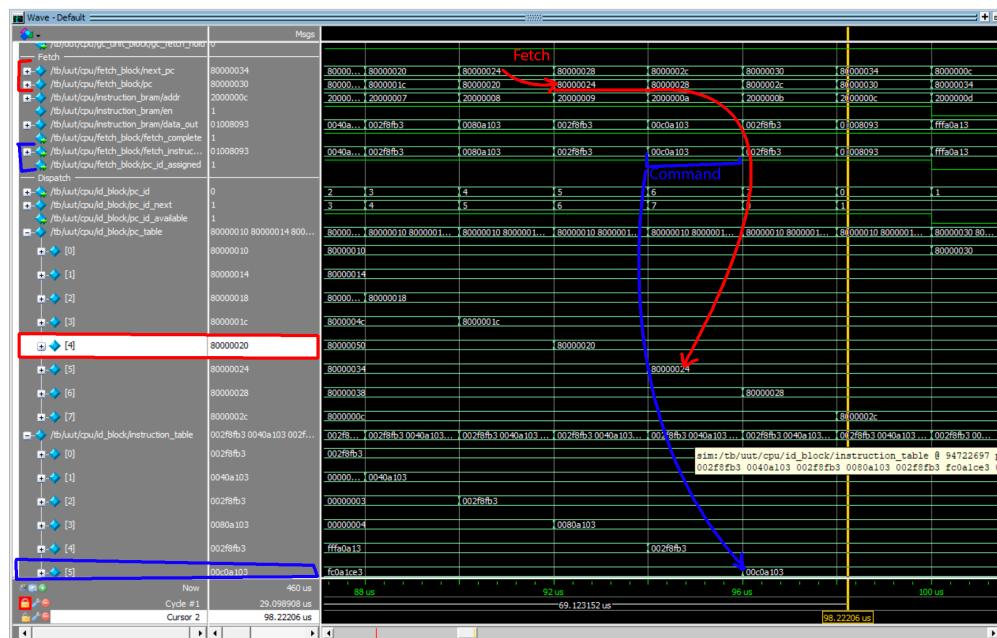


Рисунок 3.1 – Диспетчирезация команды с адресом 80000024

3.4 Задание 3

В данной секции представлены результаты выполнения декодирования и планирования.

В соответствии с таблицей, приведенной в репозитории, необходимо получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с указанным адресом (Вариант №18, адрес - 80000030, 2-я итерация). Рисунок 3.2 показывает результат выполнения.

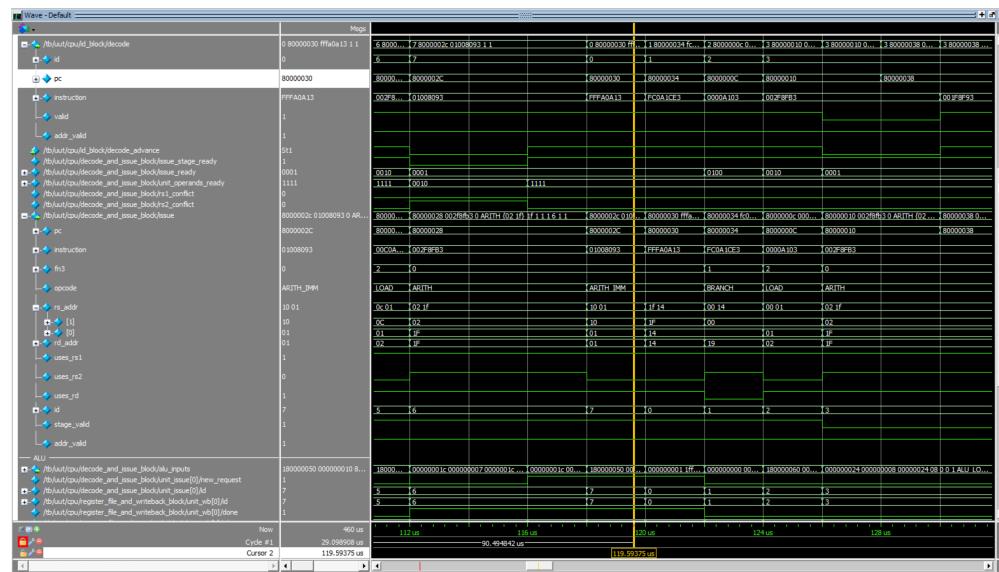


Рисунок 3.2 – Декодирование команды с адресом 80000030

Конфликта нет

3.5 Задание 4

В соответствии с таблицей, приведенной в репозитории, получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды с указанным адресом (Вариант №18, адрес - 8000001c, 2-я итерация). Рисунок 3.3 показывает результат выполнения.

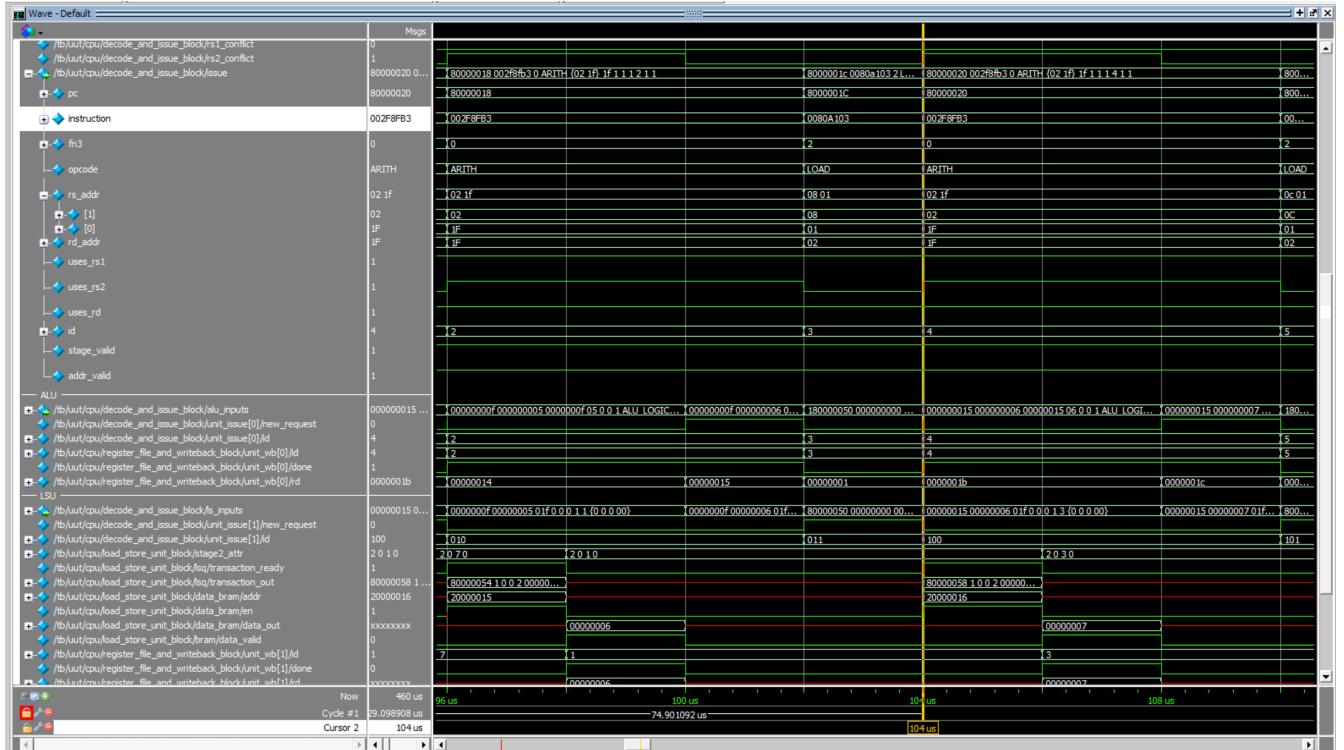


Рисунок 3.3 – Выполнение команды с адресом 8000001c

3.6 Задание 5

3.6.1 Результаты выполнения

В данной секции представлены результаты выполнения программы в соответствии с вариантом.

Рисунок 3.4 показывает результат значение регистра x31 после выполнения программы, что соответствует ранее полученному результату.



Рисунок 3.4 – Значение регистра x31

Далее представлены временные диаграммы сигналов, соответствующих всем стадиям выполнения команды в соответствии с вариантом.

А именно:

- Рисунок 3.5 показывает результат выполнения стадии выборки и диспетчирезации.

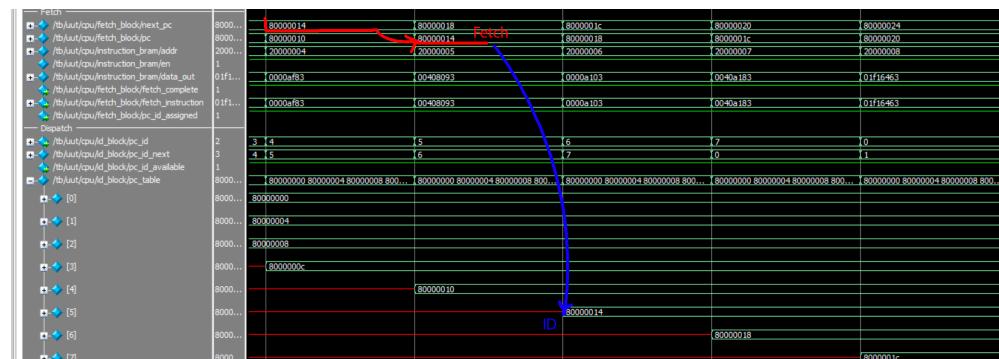


Рисунок 3.5 – Выборка и диспетчирезация

- Рисунок 3.6 показывает результат выполнения стадии декодирования и выполнения.
- Рисунок 3.7 показывает результат выполнения стадии выполнения в блоке LSU.

Таблица 3.8 показывает трассу выполнения программы.

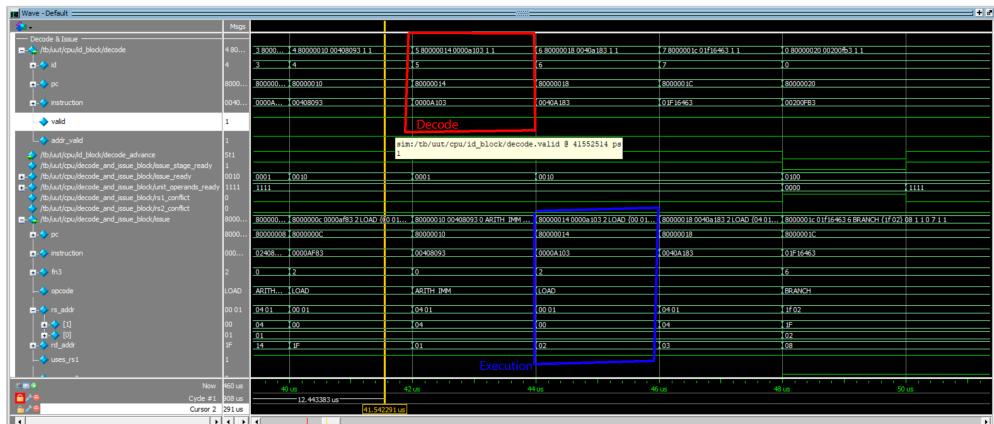


Рисунок 3.6 – Декодирование и выполнение



Рисунок 3.7 – Декодирование и выполнение

Рисунок 3.8 – Трасса неоптимизированной программы

3.6.2 Оптимизация программы

В данной секции представлены выводы об эффективности программы, а также возможных путях ее улучшения.

Анализируя трассу, мы можем увидеть, что двойное условие внутри цикла создает множество конфликтов ветвления, что приводит к тому, что ветвление не может быть корректно предсказано. Сделаем предположение, что будет лучше, мы будем использовать одно условие внутри цикла (и загрузку одной переменной).

В результате (рисунок 3.10) мы получим следующее время выполнения программы:

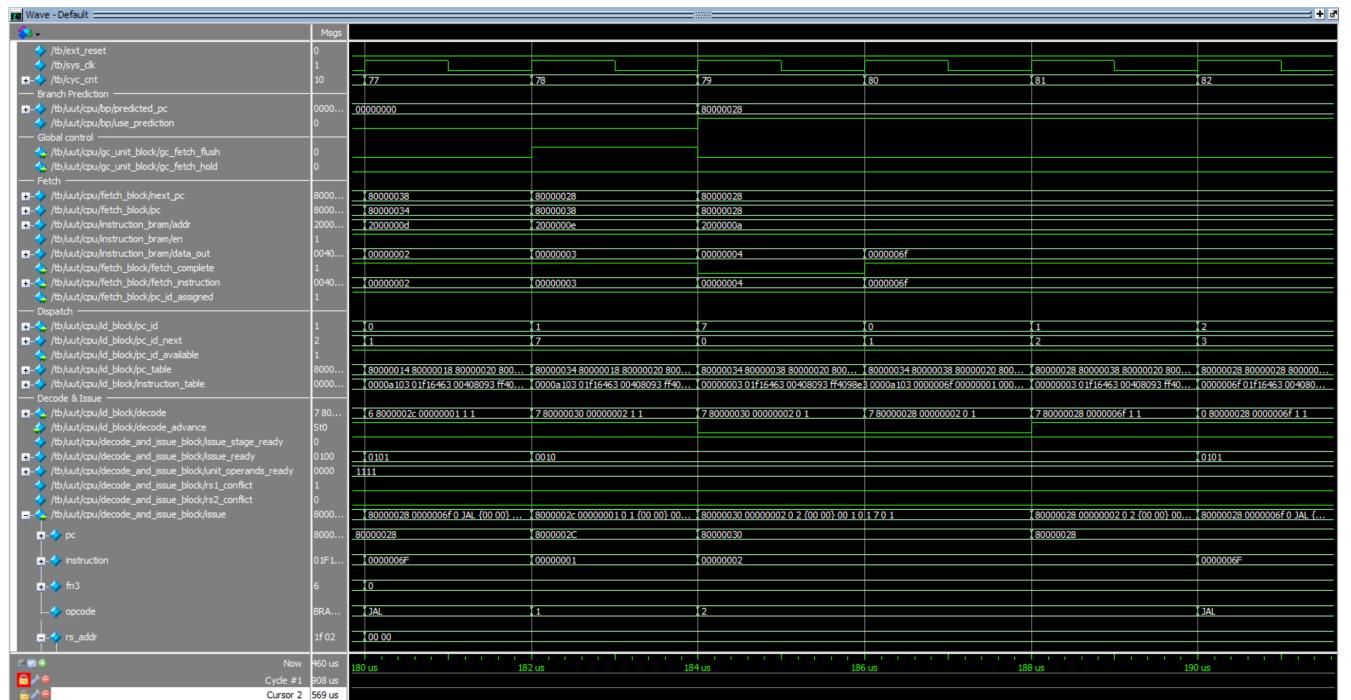


Рисунок 3.9 – Версия программы с плохой оптимизацией

Время выполнения увеличилось с 70 тиков до 82, предположение оказалось неверным. Код неудачной программы прилагается в листинге 3.4.

Листинг 3.4 – Программа для задания 6 (вар.18; неопт.)

```
1 # Variant 18
2 .section .text
3     .globl _start;
4     len = 9 #Размер массива
5     enroll = 2 #Количество обрабатываемых элементов за одну итерацию
6     elem_sz = 4 #Размер одного элемента массива
7
8 _start:
9     la x1, _x
10    addi x20, x1, elem_sz*len #Адрес элемента, следующего за последним
11    lw x31, 0(x1)
12    addi x1, x1, elem_sz*1
13 lp:
14    lw x2, 0(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 0x8
30     .4byte 0x6
31     .4byte 0x7
32     .4byte 0x5
33     .4byte 0x4
```

Вглядываясь более внимательно, мы можем заметить, что внутри цикла происходит конфликт между операциями загрузки и ветвления. Попробуем поместить другое вычисление между этими двумя операциями (в данном случае увеличение регистра, отвечающего за индексацию, который стоял в конце цикла).

В результате (рисунок 3.10) мы получим следующее время выполнения

программы:

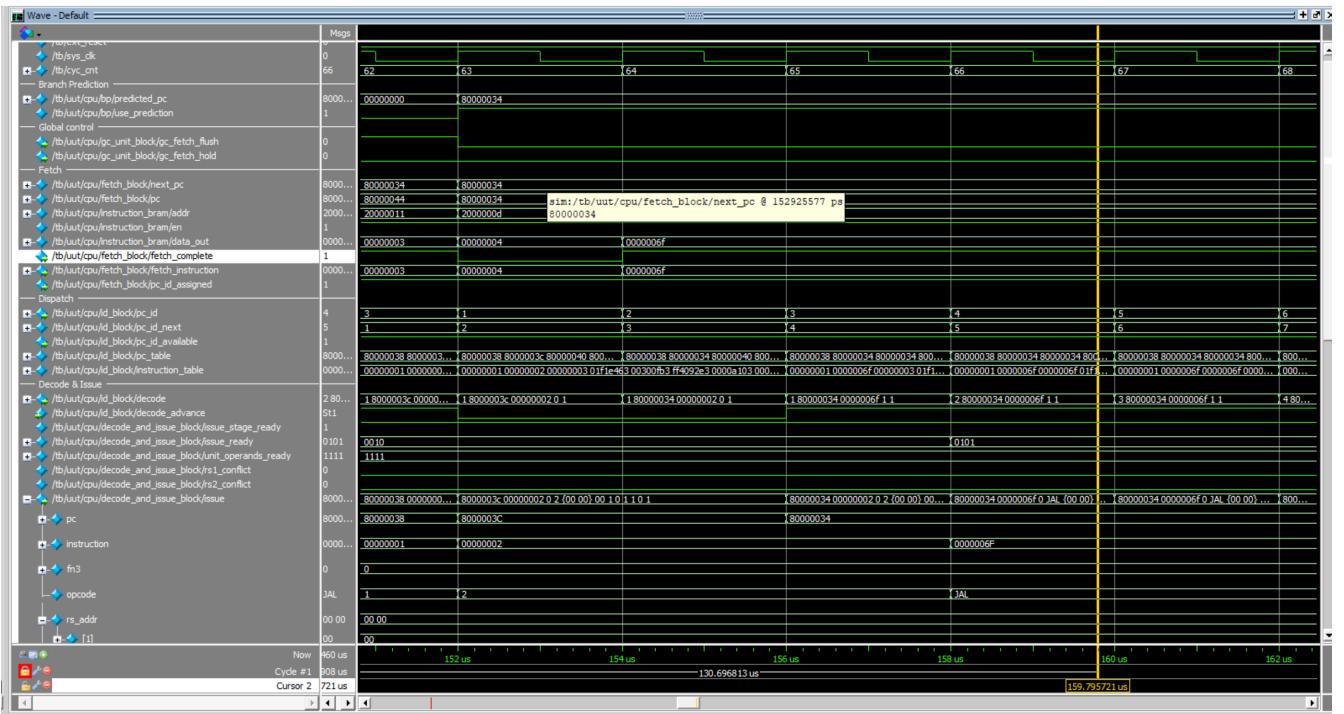


Рисунок 3.10 – Версия программы с хорошей оптимизацией

Время выполнения уменьшилось с 70 тиков до 66, предположение оказалось верным. Код удачной программы прилагается в листинге 3.5.

Листинг 3.5 – Программа для задания 6 (вар.18; опт.)

```
1 # Variant 18
2 .section .text
3     .globl _start;
4     len = 9 #Размер массива
5     enroll = 2 #Количество обрабатываемых элементов за одну итерацию
6     elem_sz = 4 #Размер одного элемента массива
7
8 _start:
9     la x1, _x
10    addi x20, x1, elem_sz*len #Адрес элемента, следующего за последним
11    lw x31, 0(x1)
12    addi x1, x1, elem_sz*1
13 lp:
14    lw x2, 0(x1) #!
15    lw x3, 4(x1)
16    add x1, x1, elem_sz*enroll
17    bltu x2, x31, lt1
18    add x31, x0, x2
19 lt1: bltu x3, x31, lt2
20    add x31, x0, x3
21 lt2:
22    bne x1, x20, lp
23 lp2: j lp2
24
25     .section .data
26 _x: .4byte 0x1
27     .4byte 0x2
28     .4byte 0x3
29     .4byte 0x4
30     .4byte 0x8
31     .4byte 0x6
32     .4byte 0x7
33     .4byte 0x5
34     .4byte 0x4
```

Таблица 3.11 содержит трассу выполнения программы с оптимизацией.

Рисунок 3.11 – Трасса выполнения программы с оптимизацией

3.7 Выводы

В ходе выполнения лабораторной работы были изучены основные архитектурные особенности процессора RISC-V, а также основные инструкции и регистры процессора. Также удалось оптимизировать программу, уменьшив время выполнения с 70 тиков до 66 тиков за счет уменьшения количества конфликтов