



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 № 2 по дисциплине "Архитектура ЭВМ"

Тема Изучение принципов работы микропроцессорного ядра RISC-V

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Дубровин Е. Н.

Москва — 2021 г.

Содержание

Введение	3
1 Основные теоретические сведения	4
2 Задание 1	5
2.1 Результат выполнения	5
2.2 Вывод	8
3 Задание 2	9
3.1 Результат выполнения	9
3.2 Вывод	9
4 Задание 3	10
4.1 Результат выполнения	10
4.2 Вывод	10
5 Задание 4	11
5.1 Результат выполнения	11
5.2 Вывод	11
6 Задание 5	12
6.1 Проверка результата	12
6.2 Стадии выполнения команды, обозначенной в тексте программы символом	12
6.3 Трасса выполнения программы	14
6.4 Оптимизация программы	15
Заключение	19

Введение

Основной целью данной лабораторной 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.

1 Основные теоретические сведения

Изучение архитектуры суперскалярных конвейерных микропроцессоров используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

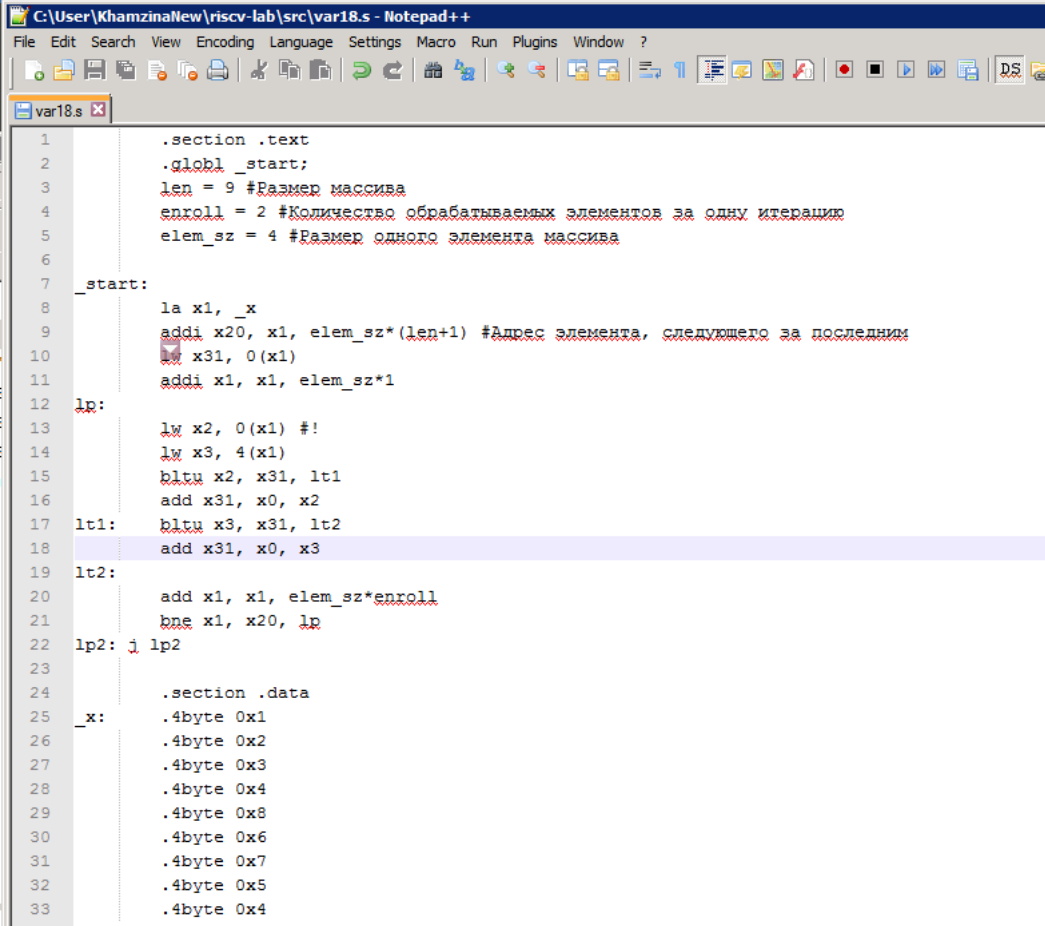
Термин RISC-V является названием для семейства различных систем команд, которые строятся вокруг базового набора команд, путем внесения в него различных расширений. В данной работе исследуется набор команд RV32I, который включает в себя основные команды 32-битной целочисленной арифметики кроме умножения и деления.

2 Задание 1

Дизассемблировать программу по индивидуальному варианту.

2.1 Результат выполнения

На рисунке 2.1 представлен исходный текст исследуемой программы для варианта № 18.



```
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+1) #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 0x8
30         .4byte 0x6
31         .4byte 0x7
32         .4byte 0x5
33         .4byte 0x4
```

Рисунок 2.1 – Текст программы

Дизассемблерный листинг данной программы показан на рисунке 2.2.

```

MINGW32:/c:/User/KhamzinaNew/riscv-lab/src

var18.elf:      file format elf32-littleriscv

SYMBOL TABLE:
80000000 l      d .text 00000000 .text
80000038 l      d .data 00000000 .data
00000000 l      df *ABS* 00000000 var18.o
00000009 l      *ABS* 00000000 len
00000002 l      *ABS* 00000000 enroll
00000004 l      *ABS* 00000000 elem_sz
80000038 l      .data 00000000 _x
80000014 l      .text 00000000 lp
80000024 l      .text 00000000 lt1
8000002c l      .text 00000000 lt2
80000034 l      .text 00000000 lp2
80000000 g      .text 00000000 _start
8000005c g      .data 00000000 _end

Disassembly of section .text:

80000000 <_start>:
80000000:      00000097      auipc    x1,0x0
80000004:      03808093      addi     x1,x1,56 # 80000038 <_x>
80000008:      02808a13      addi     x20,x1,40
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      c.addi   x0,0
8000003a:      0000      unimp
8000003c:      0002      0x2
8000003e:      0000      unimp
80000040:      00000003      lb       x0,0(x0) # 0 <enroll-0x2>
80000044:      0004      c.addi4spn x9,x2,0
80000046:      0000      unimp
80000048:      0008      c.addi4spn x10,x2,0
8000004a:      0000      unimp
8000004c:      0006      0x6
8000004e:      0000      unimp
80000050:      00000007      0x7
80000054:      0005      c.addi   x0,1
80000056:      0000      unimp
80000058:      0004      c.addi4spn x9,x2,0
...
riscv64-unknown-elf-objcopy -O binary --reverse-bytes=4 var18.elf var18.bin
xxd -g 4 -c 4 -p var18.bin var18.hex

```

Рисунок 2.2 – Дизассемблированный листинг исходной программы

В листинге 2.1 приведен псевдокод на языке C, поясняющий работу представленной программы.

Листинг 2.1 – Псевдокод программы на языке C

```
1 #define LEN 9
2 #define ENROLL 2
3 #define ELEM_SZ 4
4
5 int _x[] = {1, 2, 3, 4, 8, 6, 7, 5, 4};
6
7 int main(void)
8 {
9     int* x1 = _x;
10    int x20 = ELEM_SZ * (LEN + 1);
11    int x31 = _x[0];
12    x1 += ELEM_SZ * 1
13
14    do
15    {
16        int x2 = x1[0];
17        int x3 = x1[4];
18
19        if ( !(x2 < x31) )
20        {
21            x31 = x2;
22        }
23
24        if ( !(x3 < x31) )
25        {
26            x31 = x3;
27        }
28
29        x1 += ELEM_SZ * ENROLL;
30
31    } while ( x1 != x20 );
32
33    while ( 1 ) {}
34 }
```

2.2 Вывод

Проанализировав исходный текст программы, можно сделать вывод о том, что в регистре `x31` в конце выполнения программы должно содержаться значение `0x8`.

3 Задание 2

Получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 8000002 на 2-ой итерации.

3.1 Результат выполнения

На рисунке 3.1 представлена диаграмма, соответствующая этапам выборки и диспетчеризации требуемой команды.

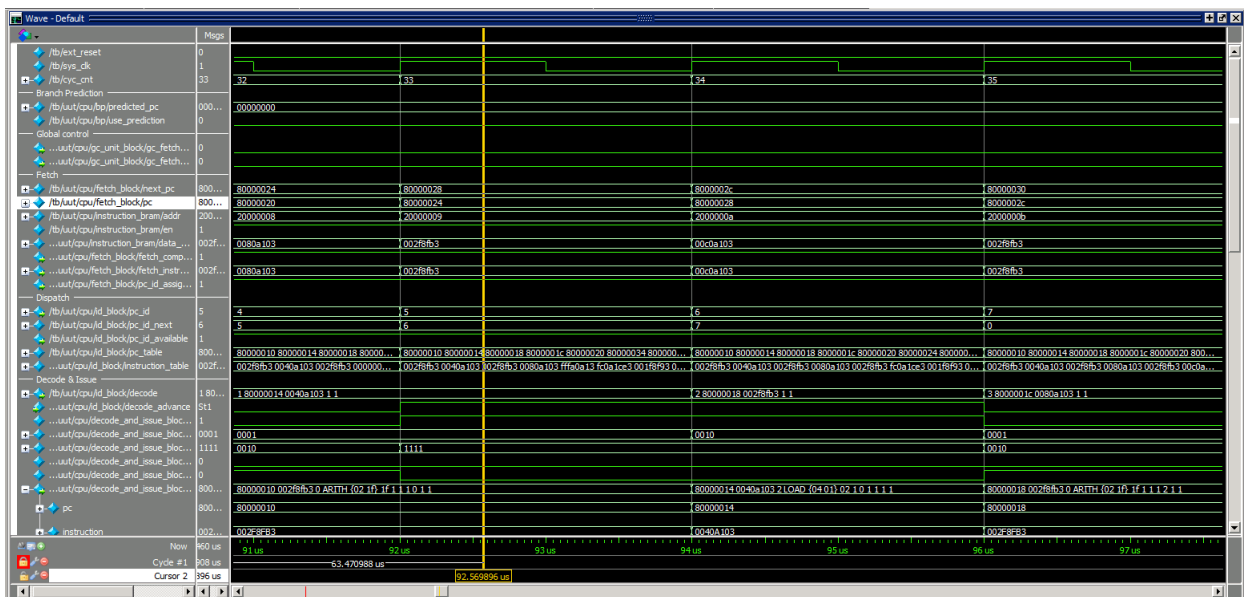


Рисунок 3.1 – Диаграмма, соответствующая этапам выборки и диспетчеризации

3.2 Вывод

Выборка и диспетчеризация данной команды происходят на 33 и 34 тактах соответственно.

4 Задание 3

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 80000030 на 2-ой итерации.

4.1 Результат выполнения

Диаграмма, соответствующая этапам декодирования и планирования требуемой команды, показана на рисунке 4.1.

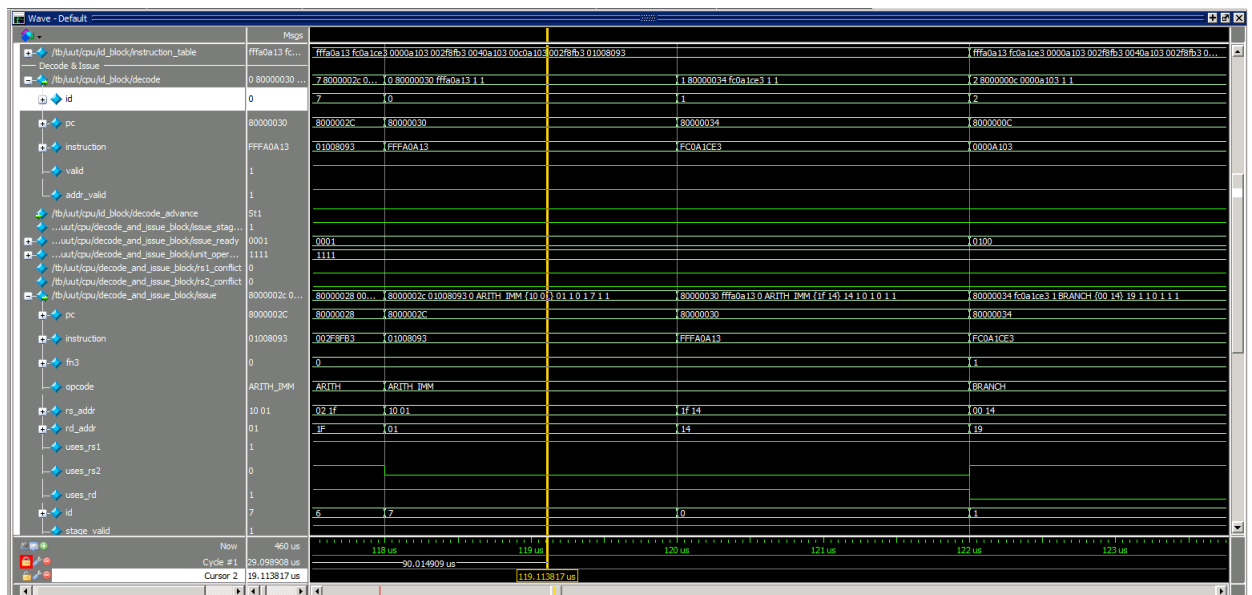


Рисунок 4.1 – Диаграмма, соответствующая этапам декодирования и планирования

4.2 Вывод

Так как данная команда выполняет арифметическую операцию, значения сигналов rs1_conflict и rs2_conflict равны 0, конфликт не возникает.

5 Задание 4

Получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды с адресом 8000001с на 2-ой итерации.

5.1 Результат выполнения

На рисунке 5.1 представлена диаграмма, соответствующая этапу выполнения требуемой команды.

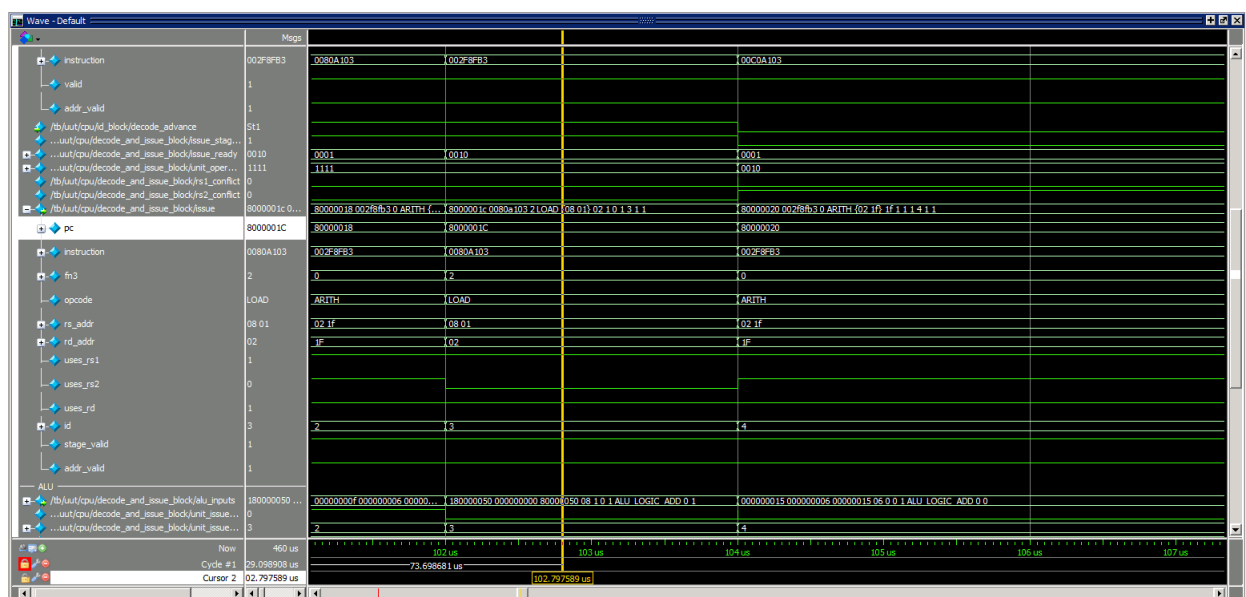


Рисунок 5.1 – Диаграмма, соответствующая этапу выполнения

5.2 Вывод

Данная команда является командой загрузки и выполняется 3 такта. В следующем такте возникает конфликт из-за обращения к памяти, в которую происходит загрузка в текущем такте.

6 Задание 5

6.1 Проверка результата

Значение регистра x31 на момент окончания выполнения программы, показанное на рисунке 6.1, равно значению, полученному в Задании 1.

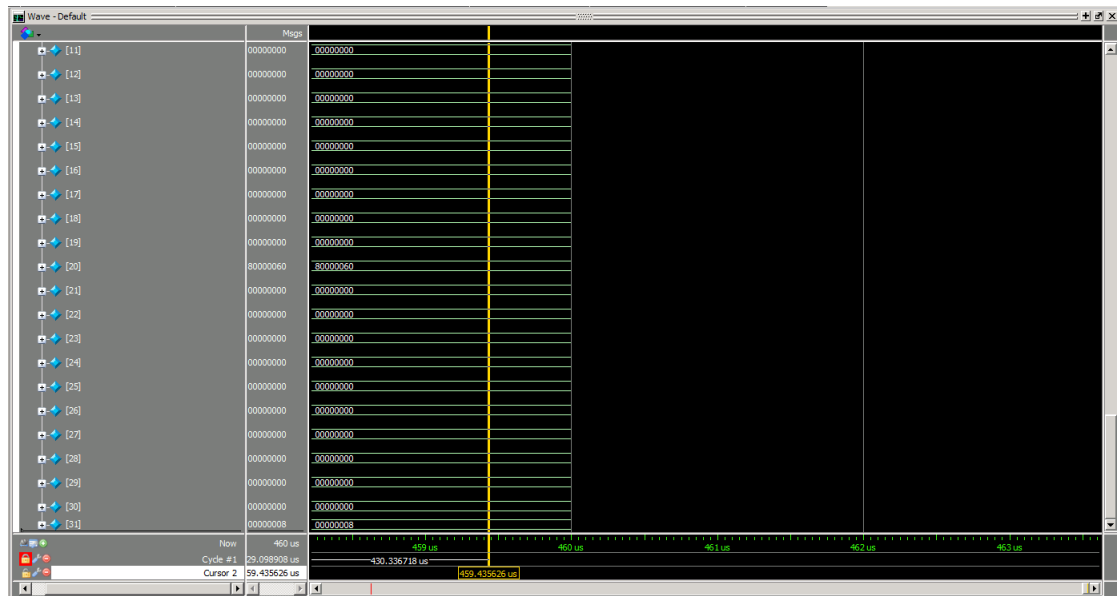


Рисунок 6.1 – Результат выполнения программы

6.2 Стадии выполнения команды, обозначенной в тексте программы символом

Для программы по индивидуальному варианту № 18 заданным символом обозначена команда $lwx2, 0(x1)$, находящаяся по адресу 0x80000014.

На рисунках 6.2-6.4 показаны временные диаграммы сигналов данной команды для стадий выборки и диспетчеризации, декодирования и планирования на выполнение и выполнения.

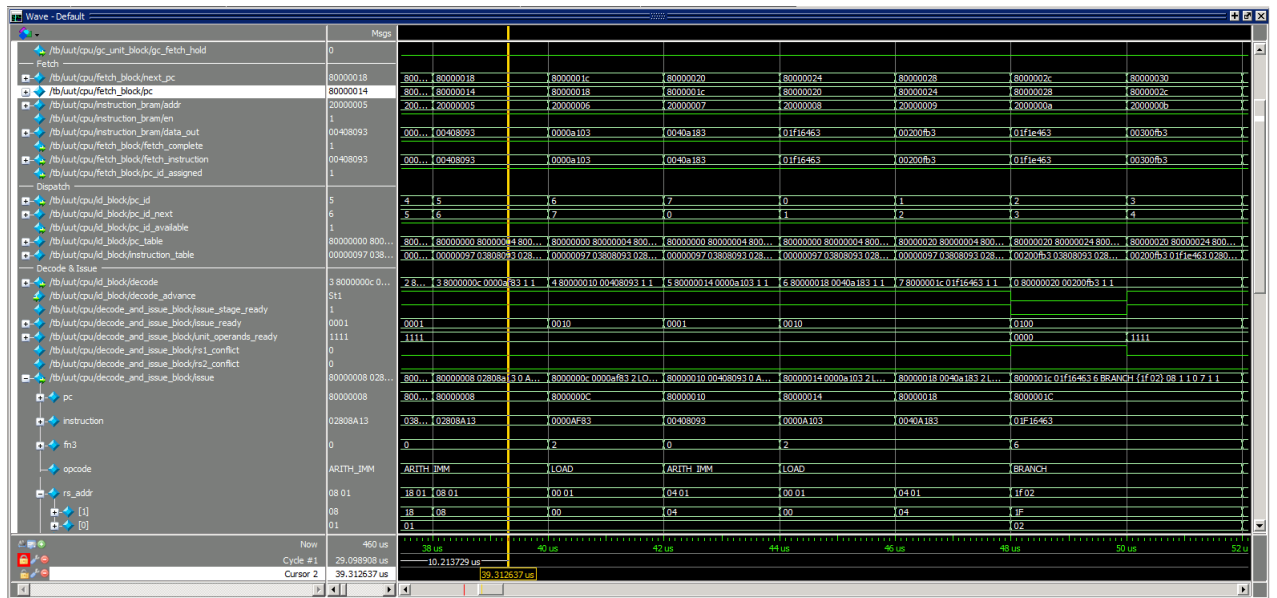


Рисунок 6.2 – Выборка и диспетчеризация команды

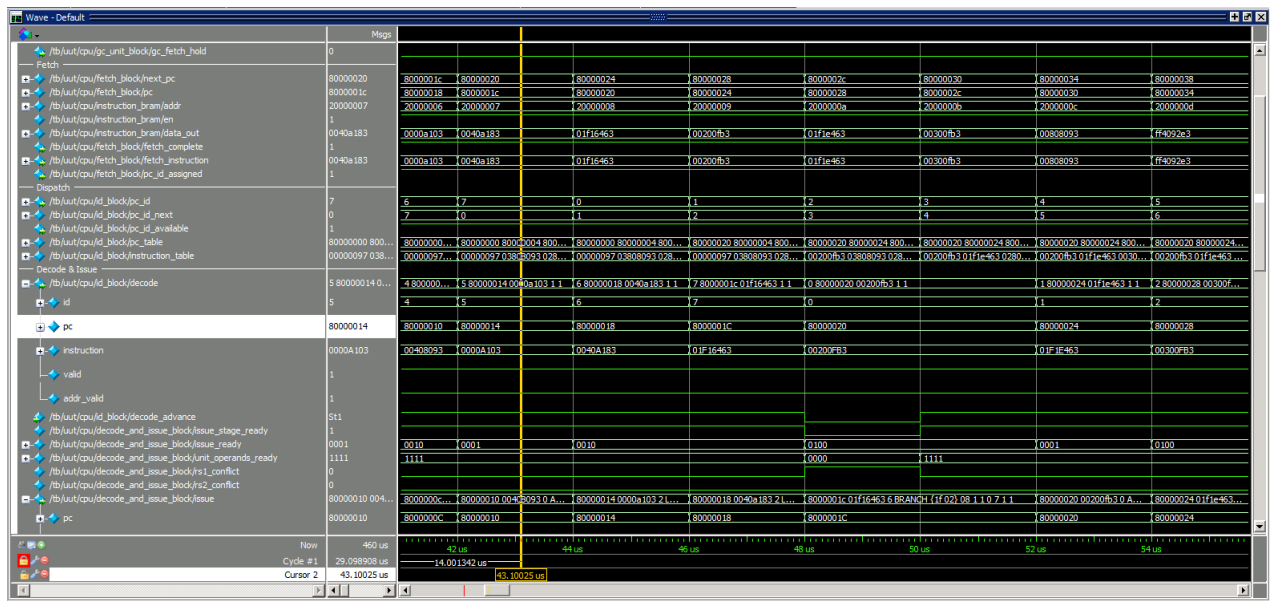


Рисунок 6.3 – Декодирование команды

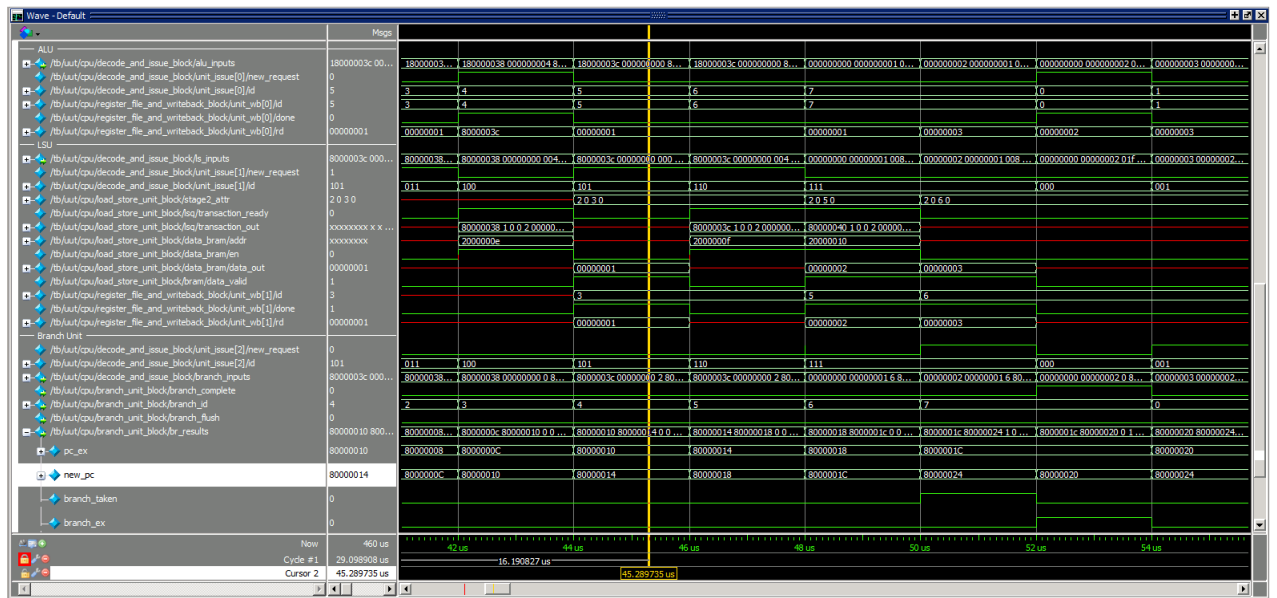


Рисунок 6.4 – Выполнение команды

6.3 Трасса выполнения программы

Анализируя трассу выполнения программы, представленной на рисунке 6.5, можно сделать вывод о том, что конфликты возникают тогда, когда блок обращения к памяти не успевает загрузить необходимое для выполнения команды условного перехода (*bltux2, x31*) значение. Это происходит потому, что блок обращения к памяти выполняет загрузку за три такта, а команда условного перехода следует за загрузкой нужной переменной через одну команду, таким образом блок обращения к памяти только завершает третий такт обработки.

[illegible]

Рисунок 6.5 – Трасса выполнения программы

6.4 Оптимизация программы

Для оптимизации программы необходимо переставить команды для устранения конфликтов: число конфликтов можно сократить, если вынести команды загрузки значений `x2` и `x3` из цикла, в таком случае конфликт возникнет только при первой итерации цикла.

На рисунке 6.6 приведен текст оптимизированной программы по индивидуальному варианту, на рисунке 6.7 - ее дизассемблерный листинг.

```
C:\User\KhamzinaNew\riscv-lab\src\var18_fixed.s - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
var18_fixed.s
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+1) #Адрес элемента, следующего за последним
10         lw x31, 0(x1)
11         addi x1, x1, elem_sz*1
12         lw x2, 0(x1)
13         lw x3, 4(x1)
14
15     lp:
16         bltu x2, x31, lt1
17         add x31, x0, x2
18         bltu x3, x31, lt2
19         add x31, x0, x3
20
21     lt1:
22         add x1, x1, elem_sz*enroll
23         lw x2, 0(x1)
24         lw x3, 4(x1)
25         bne x1, x20, lp
26
27     lp2: j lp2
28
29     .section .data
30     _x:
31         .4byte 0x1
32         .4byte 0x2
33         .4byte 0x3
34         .4byte 0x4
35         .4byte 0x8
36         .4byte 0x6
37         .4byte 0x7
38         .4byte 0x5
39         .4byte 0x4
```

Рисунок 6.6 – Оптимизированный код программы


```

MINGW32/c/User/KhamzinaNew/riscv-lab/src
var18_fixed.elf: file format elf32-littleriscv

SYMBOL TABLE:
80000000 l d .text 00000000 .text
80000040 l d .data 00000000 .data
00000000 l df "ABS" 00000000 var18_fixed.o
00000009 l "ABS" 00000000 len
00000002 l "ABS" 00000000 enroll
00000004 l "ABS" 00000000 elem_sz
80000040 l .data 00000000 _x
8000001c l .text 00000000 lp
80000024 l .text 00000000 lt1
8000002c l .text 00000000 lt2
8000003c l .text 00000000 lp2
80000000 g .text 00000000 _start
80000064 g .data 00000000 _end

Disassembly of section .text:

80000000 <_start>:
80000000: 00000097 auipc x1,0x0
80000004: 04008093 addi x1,x1,64 # 80000040 <_x>
80000008: 02808a13 addi x20,x1,40
8000000c: 0000af83 lw x31,0(x1)
80000010: 00408093 addi x1,x1,4
80000014: 0000a103 lw x2,0(x1)
80000018: 0040a183 lw x3,4(x1)

8000001c <lp>:
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: 0000a103 lw x2,0(x1)
80000034: 0040a183 lw x3,4(x1)
80000038: ff4092e3 bne x1,x2,8000001c <lp>

8000003c <lp2>:
8000003c: 0000006f jal x0,8000003c <lp2>

Disassembly of section .data:

80000040 <_x>:
80000040: 0001 c.addi x0,0
80000042: 0000 unimp
80000044: 0002 0x2
80000046: 0000 unimp
80000048: 00000003 lb x0,0(x0) # 0 <enroll-0x2>
8000004c: 0004 c.addi4spn x9,x2,0
8000004e: 0000 unimp
80000050: 0008 c.addi4spn x10,x2,0
80000052: 0000 unimp
80000054: 0006 0x6
80000056: 0000 unimp
80000058: 00000007 0x7
8000005c: 0005 c.addi x0,1
8000005e: 0000 unimp
80000060: 0004 c.addi4spn x9,x2,0
...
riscv64-unknown-elf-objcopy -O binary --reverse-bytes=4 var18_fixed.elf var18_fi

```

Рисунок 6.7 – Дизассемблерный листинг оптимизированной программы

Трасса выполнения оптимизированной программы приведена на рисунке 6.8.

Заключение

В ходе лабораторной работы были изучены принципы функционирования и построения, а также особенности архитектуры суперскалярных конвейерных микропроцессоров на примере микропроцессорного ядра Taiga, реализующего систему команд семейства RISC-V. Таким образом, цель данной работы была достигнута.