



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

Название Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина Архитектура электронно-вычислительных машин

Студент:

_____ Мансуров В. М.
подпись, дата Фамилия, И.О.

Преподаватель:

_____ Попов А. Ю.
подпись, дата Фамилия, И. О.

Москва — 2022 г.

Содержание

Цель работы	3
1 Основные теоретические сведения	4
1.1 Модель памяти	4
1.2 Система команд	4
2 Общая программа	5
3 Результаты исследования программы	9
3.1 Задание №1	9
3.2 Задание №2	14
3.3 Задание №3	17
3.4 Задание №4	18
3.5 Задание №5	19
Заключение	27
Приложение	28

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.

1 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.

1.1 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.

1.2 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.

2 Общая программа

Рассмотрим пример небольшой программы для RV32I, которым мы будем пользоваться далее для исследования процесса выполнения команд.

Данная программа выполняет суммирование значений элементов массива слов и увеличивает это значение на 1 - Листинг 2.1.

Листинг 2.1 – Пример программы

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

1. Объявление секции *.text*, содержащей исполняемый код.
2. Объявление символа *_start*, имеющего глобальную видимость. Символ *_start* это специальный символ, обозначающий точку входа в программу.
3. Метка.
4. Объявление констант.
5. Арифметические выражения над константами могут использоваться в командах на месте непосредственного операнда.
6. Загрузка в *x1* адреса символа *_x* (то есть, начала массива).
7. Загрузка в *x2* числа по адресу, содержащемуся в *x1* по смещению 0.
8. Добавление к *x31* (который хранит результат) значения *x2*.
9. Смещение указателя *x1*.
10. Уменьшение счетчика цикла.
11. Условный переход на метку *loop*.
12. Бесконечный цикл.
13. Объявление секции данных.
14. Начало описания массива.

Дизассемблерный код представлен на листинге 2.2.

Листинг 2.2 – Дизассемблированный код примера 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 2.3.

Листинг 2.3 – Псевдокод общей программы

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4 int _x[]={1,2,3,4,5,6,7,8};
5 void _start() {
6     int x20 = len/enroll;
7     int *x1 = _x;
8
9     do {
10         int x2 = x1[0];
11         x31 += x2;
12         x2 = x1[1];
13         x31 += x2;
14         x2 = x1[2];
15         x31 += x2;
16         x2 = x1[3];
17         x31 += x2;
18         x1 += enroll;
19         x20--;
20     } while(x20 != 0);
21     x31++;
22     while(1){}
23 }
```


3 Результаты исследования программы

Задания выполнялись по варианту 9.

3.1 Задание №1

Условие задания

В процессе выполнения задания необходимо выполнить следующие действия:

1. Ознакомиться с теоретической частью, внимательно изучить примеры.
2. Перейти в подкаталог *src* командой *cd riscv – lab/src*.
3. Выполнить сборку, запустив команду *gmake*. Убедиться, что был создан файл *test.hex*, содержащий шестнадцатеричное представление программы, а в окне терминала отобразился дизассемблерный листинг. Сравнить дизассемблерный листинг с тем, который приведен в примере.
4. Создать новый файл, содержащий текст программы по индивидуальному варианту (см. Индивидуальные варианты). Поместить его в каталог *src*. Текст программы сохранить в файле с расширением *.s*.
5. Изучить текст программы по индивидуальному варианту. Поместить в отчете псевдокод, соответствующий данной программе.
6. Анализируя исходный текст программы, ответьте на вопрос: какое значение должно содержаться в регистре *x31* в конце выполнения программы?
7. Изменить в *Makefile* строку *SRC* = так, чтобы ее содержимое соответствовало имени файла с текстом программы без расширения *.s*.
8. Выполнить компиляцию командой *gmake*. В процессе будет создан файл с расширением *.hex*, хранящий содержимое памяти команд и данных, а в окне терминала отобразится дизассемблерный листинг, который необходимо поместить в отчет вместе с исходным текстом.

Результаты выполнения

Листинг 3.1 – Код программы 9 варианта

```
1      .section .text
2      .globl _start;
3      len = 8
4      enroll = 4
5      elem_sz = 4
6
7 _start:
8      addi x20, x0, len/enroll
9      la x1, _x
10     lp:
11     lw x2, 0(x1)
12     add x31, x31, x2 #!
13     lw x3, 4(x1)
14     add x31, x31, x3
15     lw x4, 8(x1)
16     lw x5, 12(x1)
17     add x31, x31, x4
18     add x31, x31, x5
19     addi x1, x1, elem_sz*enroll
20     addi x20, x20, -1
21     bne x20, x0, lp
22     addi x31, x31, 1
23 lp2: j lp2
24
25     .section .data
26 _x: .4 byte 0x1
27     .4 byte 0x2
28     .4 byte 0x3
29     .4 byte 0x4
30     .4 byte 0x5
31     .4 byte 0x6
32     .4 byte 0x7
33     .4 byte 0x8
```

Выполнение команды *gmake*, после присвоения *SRC* = название файла, где содержится код программы варианта 9, в мое случае этот файл был назван *lab_02_09*, это показано на рисунке 3.1

```

user29@WIN-40TB9R26C78 MINGW32 /c/User/mansurov/riscv-lab/src (main)
$ make
riscv64-unknown-elf-as --march=rv32i lab_02_09.s -o lab_02_09.o
lab_02_09.s: Assembler messages:
lab_02_09.s: Warning: end of file not at end of a line; newline inserted
riscv64-unknown-elf-ld -b elf32-littleriscv -T link.ld lab_02_09.o -o lab_02_09.elf
riscv64-unknown-elf-objdump -D -M numeric,no-aliases -t lab_02_09.elf

lab_02_09.elf:      file format elf32-littleriscv

SYMBOL TABLE:
80000000 l    d .text 00000000 .text
80000040 l    d .data 00000000 .data
00000000 l    df *ABS* 00000000 lab_02_09.o
00000008 l    *ABS* 00000000 len
00000004 l    *ABS* 00000000 enroll
00000004 l    *ABS* 00000000 elem_sz
80000040 l    .data 00000000 _x
8000000c l    .text 00000000 lp
8000003c l    .text 00000000 lp2
80000000 g    .text 00000000 _start
80000060 g    .data 00000000 _end

Disassembly of section .text:

80000000 <_start>:
80000000:    00200a13          addi    x20,x0,2
80000004:    00000097          auipc   x1,0x0
80000008:    03c08093          addi    x1,x1,60 # 80000040 <_x>

8000000c <lp>:
8000000c:    0000a103          lw      x2,0(x1)
80000010:    002f8fb3          add     x31,x31,x2
80000014:    0040a183          lw      x3,4(x1)
80000018:    003f8fb3          add     x31,x31,x3
8000001c:    0080a203          lw      x4,8(x1)
80000020:    00c0a283          lw      x5,12(x1)
80000024:    004f8fb3          add     x31,x31,x4
80000028:    005f8fb3          add     x31,x31,x5
8000002c:    01008093          addi    x1,x1,16
80000030:    fffa0a13          addi    x20,x20,-1
80000034:    fc0a1ce3          bne     x20,x0,8000000c <lp>
80000038:    001f8f93          addi    x31,x31,1

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 <elem_sz-0x4>
8000004c:    0004          c.addi4spn x9,x2,0
8000004e:    0000          unimp
80000050:    0005          c.addi  x0,1
80000052:    0000          unimp
80000054:    0006          0x6
80000056:    0000          unimp

```

Рисунок 3.1 – Скриншот запуска make для программы по варианту 9

Листинг 3.2 – Дизассемблированный код 9 варианта

```

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:      0040a183      add     x31,x31,x2
11 80000014:      0080a203      lw      x4,8(x1)
12 80000018:      00c0a283      add     x31,x31,x3
13 8000001c:      002f8fb3      lw      x3,4(x1)
14 80000020:      003f8fb3      add     x31,x31,x4
15 80000024:      004f8fb3      lw      x5,12(x1)
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 С, представленному на листинге 3.3.

Листинг 3.3 – Псевдокод программы 9 варианта

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4 int _x[]={1,2,3,4,5,6,7,8};
5 void _start() {
6     int x20 = len/enroll;
7     int *x1 = _x;
8
9     do {
10         int x2 = x1[0];
11         x31 += x2;
12         int x3 = x1[1];
13         x31 += x3;
14         int x4 = x1[2];
15         x31 += x4;
16         int x5 = x1[3];
17         x31 += x5;
18         x1 += enroll;
19         x20--;
20     } while(x20 != 0);
21     x31++;
22     while(1){}
23 }
```

После выполнения программы по варианту 9 в $x31$ будет записано число 38 (подсчитано вручную и написанной программы на языке С по псевдокоду).

3.2 Задание №2

Условие задания

В ходе выполнения данного задания необходимо выполнить следующие действия:

1. Открыть проект *riscv-lab/taiga/taiga.qpf* в среде *Intel Quartus*. При запуске *Intel Quartus*, если потребуется, выбрать опцию *Run the Quartus Prime software*.
2. Выполнить синтез проекта выбрав пункт меню Processing → Start → Start Analysis & Synthesis.
3. Запустить симуляцию в среде Modelsim. Для этого выбрать в меню Quartus пункт Tools → Run Simulation Tool → RTL Simulation.
4. Запустить симуляцию, набрав в командной строке Modelsim команду *run 460us*.
5. Изучить список сигналов, приведенных в окне Wave.
6. Получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 8000002с на первой итерации.

Результаты выполнения

После выполнения пунктов 1-4 запускается симуляция в среде Modelsim, как видно на рисунке 3.2.

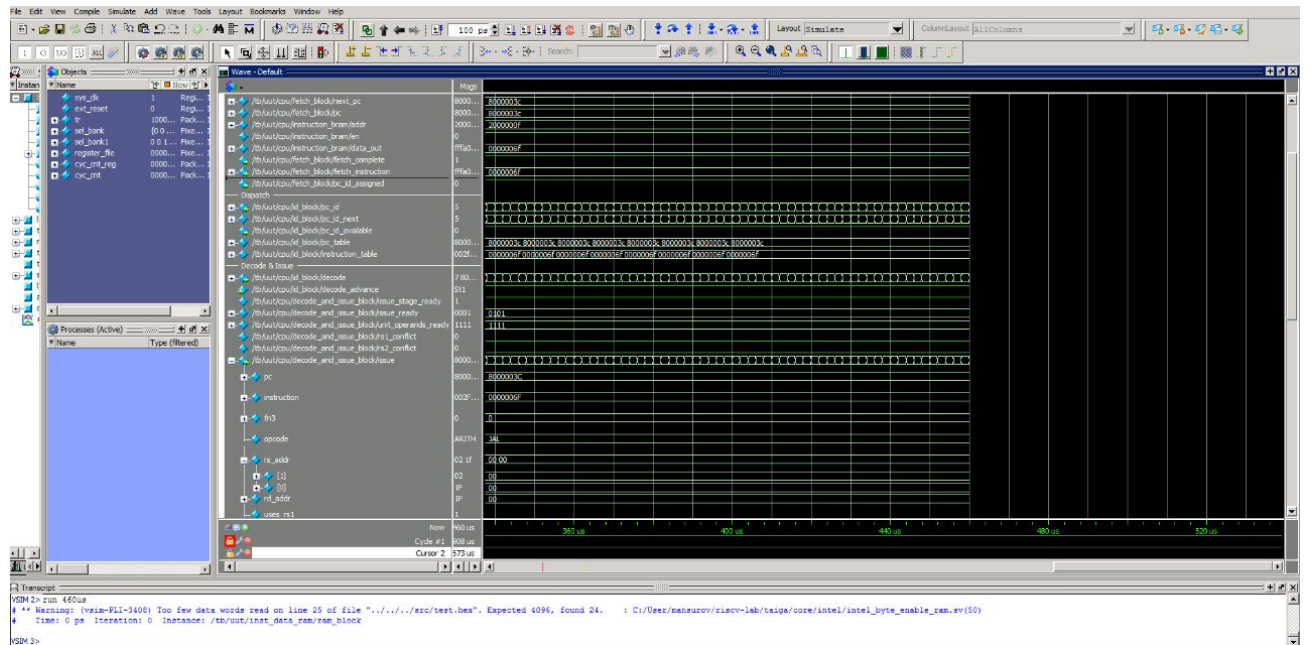


Рисунок 3.2 – Скриншот запуска симуляция в среде Modelsim

На рисунке 3.3 снимок экране симуляции в среде Modelsim на стадии выборки и диспетчеризации команды с адресом 8000002с на первой итерации.

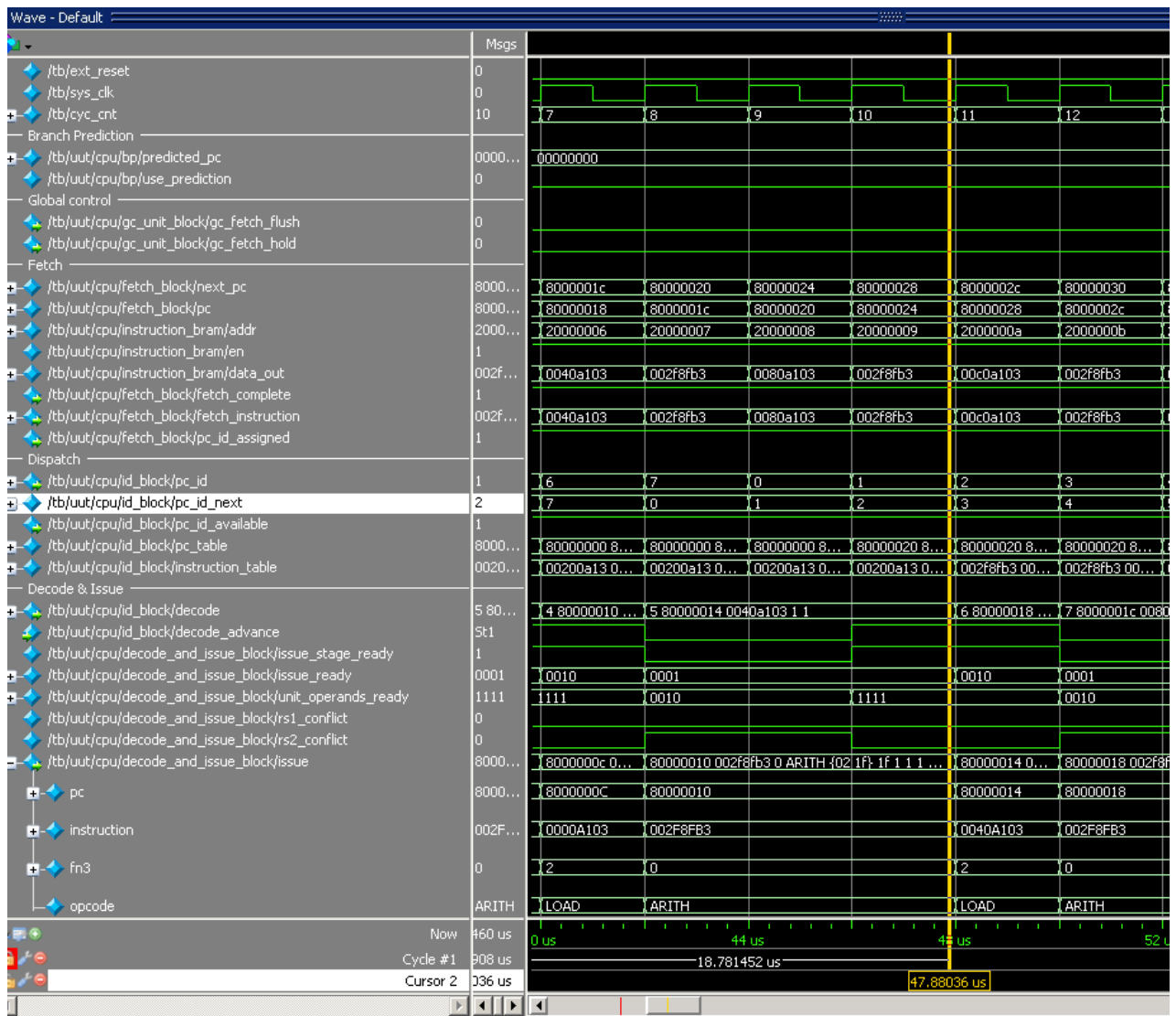


Рисунок 3.3 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 8000002с на первой итерации.

3.3 Задание №3

Условие задания

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 8000000с на второй итерации.

Результаты выполнения

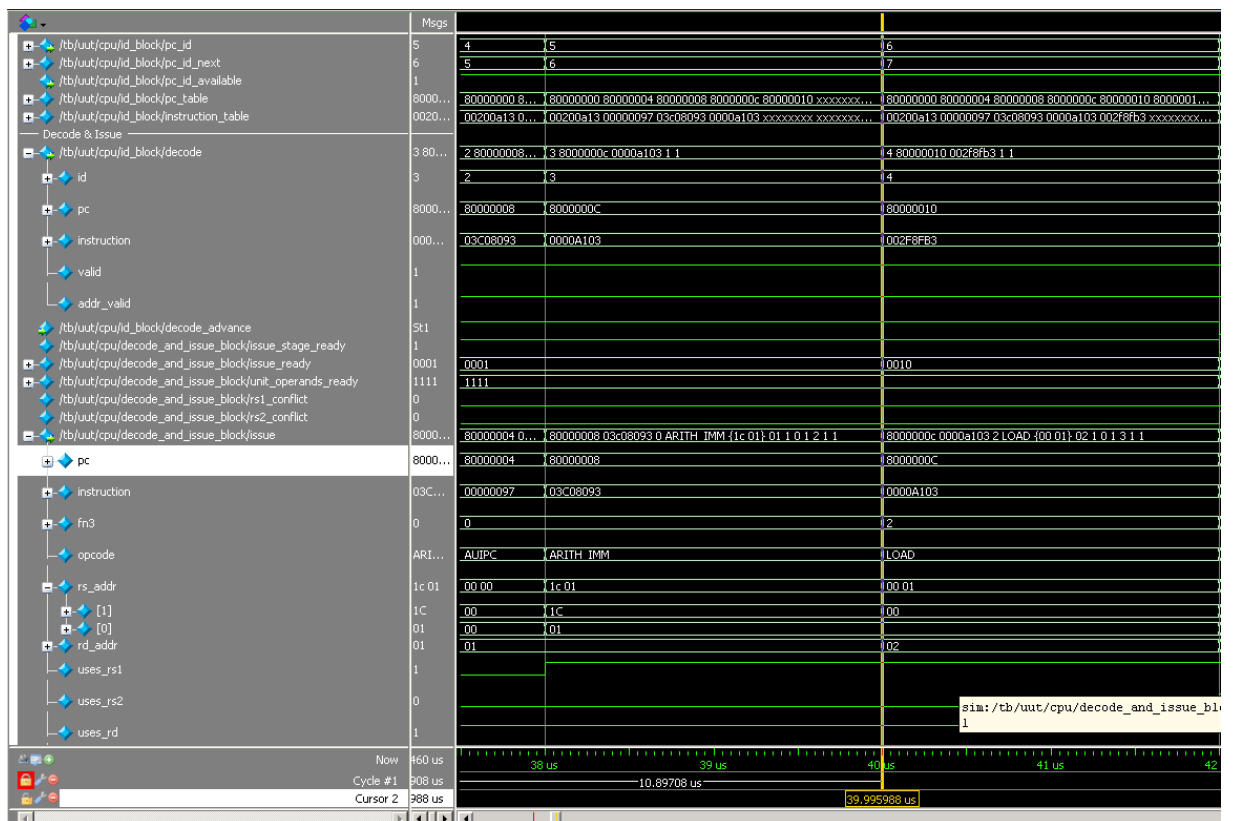


Рисунок 3.4 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 8000000с на второй итерации.

3.4 Задание №4

Условие задания

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

Результаты выполнения

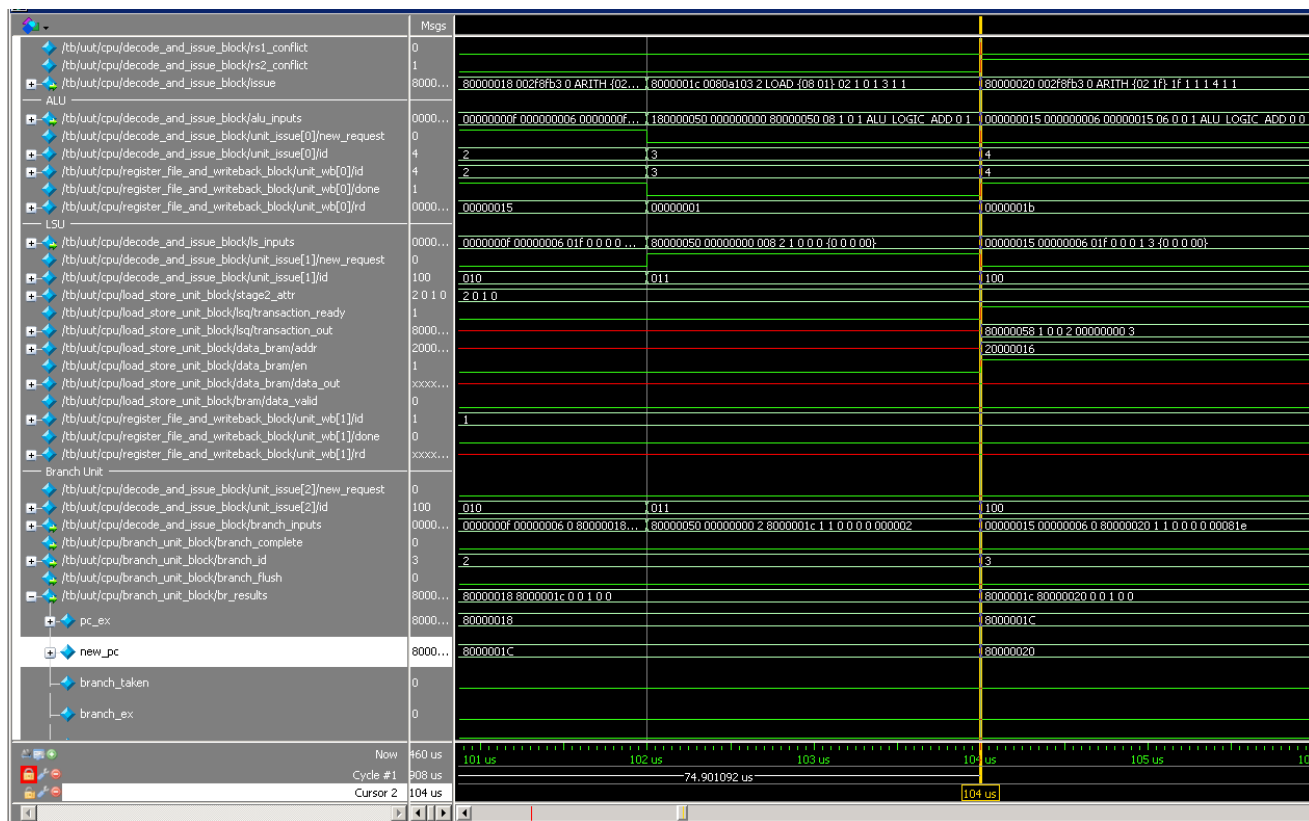


Рисунок 3.5 – Скриншот запуска симуляция в среде Modelsim – команды с адресом 80000020 на первой итерации.

3.5 Задание №5

Условие задания

В процессе выполнения задания необходимо выполнить следующие действия:

1. Исправить 76-ю строку файла *taiga/examples/zedboard/taiga_wrapper.sv* так, чтобы там был указан путь к файлу *.hex*, соответствующему программе по индивидуальному варианту. Сохранить файл.
2. Перекомпилировать исправленный файл. Для этого в окне программы Modelsim найти вкладку Library, в этой вкладке найти модуль work → taiga_wrapper. В контекстном меню модуля выбрать пункт Recompile.
3. Ввести в командой строке Modelsim команду *restart; run 460us* для перезапуска симуляции.
4. Получить временную диаграмму сигналов выполнения программы индивидуального варианта.
5. Сравнить значение регистра x31 (сигнал */tb/register_file[31]*) на момент окончания выполнения программы с тем, который был получен в задании №1.
6. Получить снимок экрана, содержащий временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом *#!*.
7. Анализируя диаграмму заполнить трассу выполнения программы. Рекомендуется использовать для этого файл *pipeline.ods*, содержащий трассу тестового примера.
8. Сделать вывод об эффективности выполнения программы и о путях оптимизации.
9. Провести оптимизацию программы путем перестановки команд для устранения конфликтов.

- ## Результаты выполнения

Трасса работы представлена на рисунке 3.6.

[illegible]

Рисунок 3.6 – Трасса работы программы.

Временные диаграммы

Временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом `#!` (`add x31, x31, x2`) с адресом `80000010` представлены на рисунке 3.7 – 3.9.

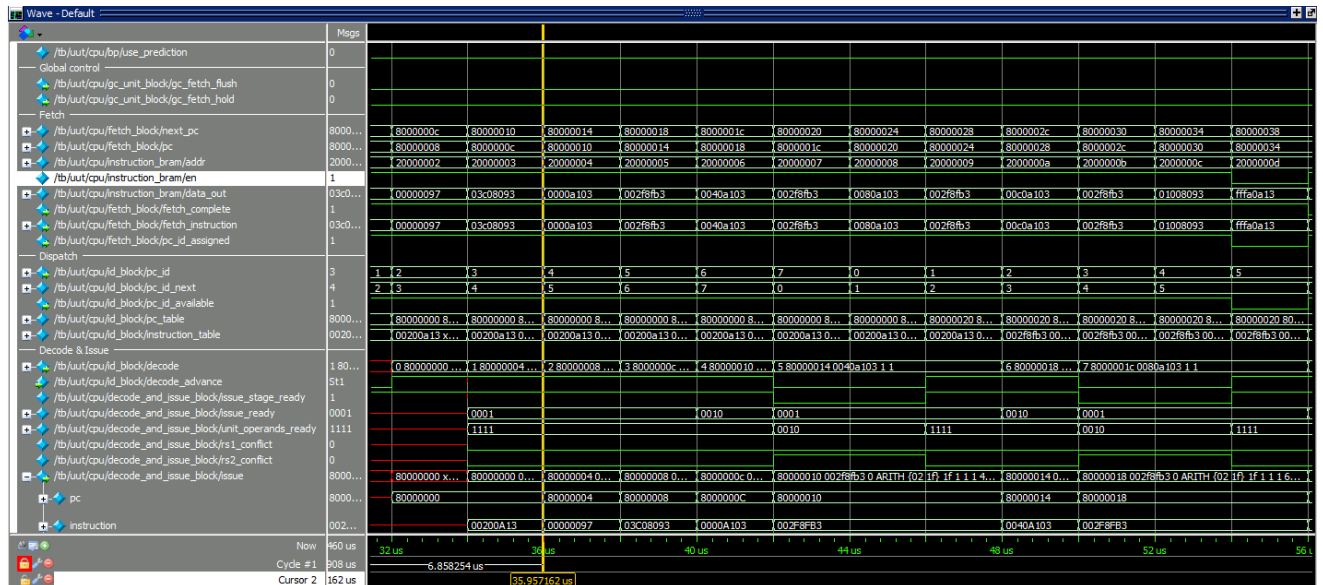


Рисунок 3.7 – Временные диаграммы сигналов выборки и диспетчеризации (Fetch & Dispatch) с адресом `80000010`.

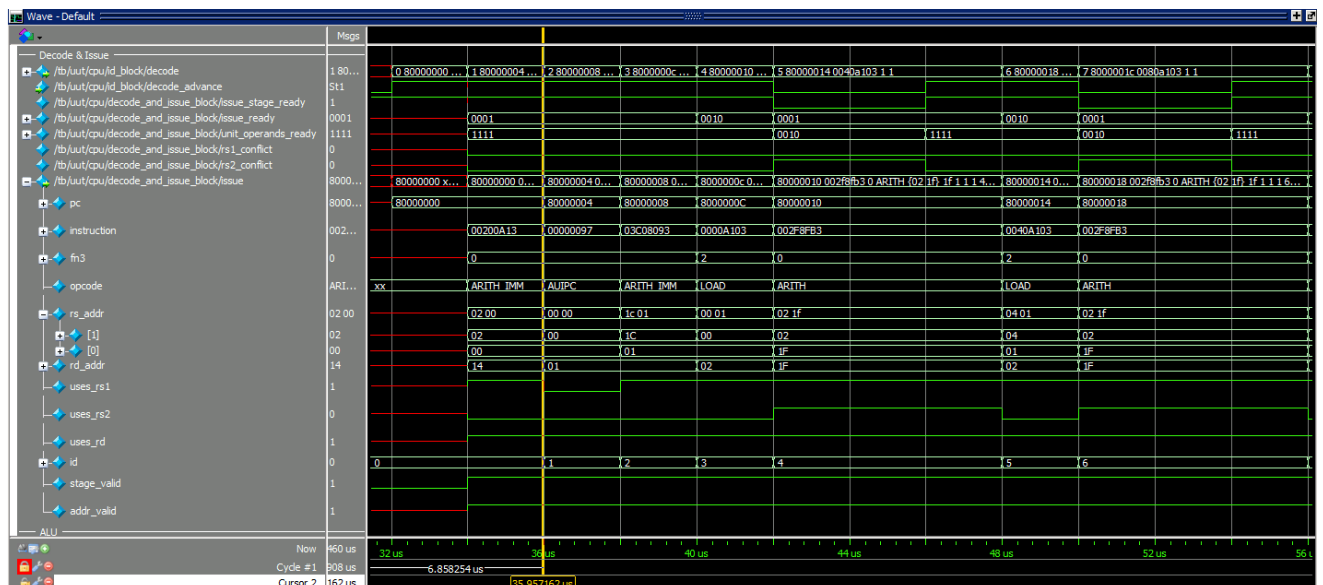


Рисунок 3.8 – Временные диаграммы сигналов декодирования и планирования (Decode & Issue) с адресом `80000010`.

Оптимизированная программа

Код программы представлен в листинге 3.4

Листинг 3.4 – Код программы 9 варианта(оптимизированный)

```
1 .section .text
2 .globl _start;
3 len = 8
4 enroll = 4
5 elem_sz = 4
6
7 _start:
8 addi x20, x0, len/enroll
9 la x1, _x
10 lp:
11 lw x2, 0(x1)
12 lw x3, 4(x1)
13 lw x4, 8(x1)
14 lw x5, 12(x1)
15 add x31, x31, x2 #!
16 add x31, x31, x3
17 add x31, x31, x4
18 add x31, x31, x5
19 addi x1, x1, elem_sz*enroll
20 addi x20, x20, -1
21 bne x20, x0, lp
22 addi x31, x31, 1
23 lp2: j lp2
24
25 .section .data
26 _x:      .4 byte 0x1
27 .4 byte 0x2
28 .4 byte 0x3
29 .4 byte 0x4
30 .4 byte 0x5
31 .4 byte 0x6
32 .4 byte 0x7
33 .4 byte 0x8
```

Дизассемблерный код представлен на листинге 3.5.

Листинг 3.5 – Дизассемблированный код 9 варианта (оптимизированный)

```
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:      0040a183      lw      x3,4(x1)
11 80000014:      0080a203      lw      x4,8(x1)
12 80000018:      00c0a283      lw      x5,12(x1)
13 8000001c:      002f8fb3      add     x31,x31,x2
14 80000020:      003f8fb3      add     x31,x31,x3
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 3.6.

Листинг 3.6 – Псевдокод программы 9 варианта (оптимизированный)

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4 int _x[]={1,2,3,4,5,6,7,8};
5 void _start() {
6     int x20 = len/enroll;
7     int *x1 = _x;
8
9     do {
10         int x2 = x1[0];
11         int x3 = x1[1];
12         int x4 = x1[2];
13         int x5 = x1[3];
14         x31 += x2;
15         x31 += x3;
16         x31 += x4;
17         x31 += x5;
18         x1 += enroll;
19         x20--;
20     } while(x20 != 0);
21     x31++;
22     while(1){}
23 }
```

Трасса работы оптимизированной программы

Трасса работы представлена на рисунке 3.10.

Адрес	Код команды	Команда	id	Номер такта																																												
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42			
80000000<_start>	00200a13	addi x20,x0,2	0	F	ID	D	AL																																									
80000004	00000097	auipc x1,0x0	1		F	ID	D	AL																																								
80000008	03c08093	addi x1,x1,60#80000040<x>	2			F	ID	D	AL																																							
8000000c<loop>	0000a103	lw x2,0(x1)	3				F	ID	D	M1	M2	M3																																				
80000010	0040a183	lw x3,4(x1)	4					F	ID	D	M1	M2	M3																																			
80000014	0080a203	lw x4,8(x1)	5						F	ID	D	M1	M2	M3																																		
80000018	00c0a283	lw x5,12(x1)	6							F	ID	D	M1	M2	M3																																	
8000001c	002f8fb3	add x31,x31,x2	7								F	ID	D	AL																																		
80000020	003f8fb3	add x31,x31,x3	8									F	ID	D	AL																																	
80000024	004f8fb3	add x31,x31,x4	9										F	ID	D	AL																																
80000028	005f8fb3	add x31,x31,x5	10											F	ID	D	AL																															
8000002c	01008093	addi x1,x1,16	11												F	ID	D	AL																														
80000030	fffa0a13	addi x20,x20,-1	12													F	ID	D	AL																													
80000034	fc0a1ce3	bne x20,x0,8000000c <lp>	13														F	ID	D	B																												
80000038	001f8f93	addi x31,x31,1	14															F	ID	D	X																											
8000003c	0000006f	jal x0,8000003c <lp2>	15																F	ID	DX																											
80000040	00000001	<invalid operation>	16																	F	X																											
80000044	00000002	<invalid operation>	17																		F	X																										
8000000c<loop>	0000a103	lw x2,0(x1)	18																		F	ID	D	M1	M2	M3																						
80000010	0040a183	lw x3,4(x1)	19																			F	ID	D	M1	M2	M3																					
80000014	0080a203	lw x4,8(x1)	20																				F	ID	D	M1	M2	M3																				
80000018	00c0a283	lw x5,12(x1)	21																					F	ID	D	M1	M2	M3																			
8000001c	002f8fb3	add x31,x31,x2	22																					F	ID	D	AL																					
80000020	003f8fb3	add x31,x31,x3	23																						F	ID	D	AL																				
80000024	004f8fb3	add x31,x31,x4	24																							F	ID	D	AL																			
80000028	005f8fb3	add x31,x31,x5	25																								F	ID	D	AL																		
8000002c	01008093	addi x1,x1,16	26																									F	ID	D	AL																	
80000030	fffa0a13	addi x20,x20,-1	27																										F	ID	D	AL																
80000034	fc0a1ce3	bne x20,x0,8000000c <lp>	28																											F	ID	D	B															
8000000c<loop>	0000a103	lw x2,0(x1)	29																												F	ID	D	X														
80000010	0040a183	lw x3,4(x1)	30																													F	ID	DX														
80000014	0080a203	lw x4,8(x1)	31																														F	X														
80000018	00c0a283	lw x5,12(x1)	32																																													
80000038	001f8f93	addi x31,x31,1	33																																													
8000003c	0000006f	jal x0,8000003c <lp2>	34																																													
80000040	00000001	<invalid operation>	35																																													
80000044	00000002	<invalid operation>	36																																													
80000048	00000003	<invalid operation>	37																																													
8000004c	00000004	<invalid operation>	38																																													
8000003c	0000006f	jal x0,8000003c <lp2>	39																																													
			40																																													
			41																																													
			42																																													
Адрес	Код команды	Команда	id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42			
				Номер такта																																												

Рисунок 3.10 – Трасса работы оптимизированной программы.

Заключение

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

[illegible]

Рисунок 3.11 – Трасса работы программы.

[illegible]

Рисунок 3.12 – Трасса работы оптимизированной программы.