00

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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная Инженерия**

О Т Ч Е Т

по лабораторной работе № 2

Название: Изучение принципов работы микропроцессорного ядра RISCV

Дисциплина: Архитектура ЭВМ

Студент	ИУ7-51Б		Кузнецова А.В.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Попов А. Ю.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022

Цели 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.

Для достижения поставленных целей в настоящей лабораторной работе используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

RISC-V является открытым современным набором команд, который может использоваться для построения как микроконтроллеров, так и высокопроизводительных микропроцессоров. В связи с такой широкой областью применения в систему команд введена вариативность. 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. В рамках данного набора команд мы не будем рассматривать системные команды, связанные с таймерами, системными регистрами, управлением привилегиями, прерываниями и исключениями.

Задание №1

Скомпилированная тестовая программа:

```
user640@WIN-4QT89R26C78 MINGW32 /c/User/ShatskiyR/riscv-lab/src (main)
$ gmake
riscv64-unknown-elf-as --march=rv32i test.s -o test.o
riscv64-unknown-elf-ld -b elf32-littleriscv -T link.ld test.o -o test.elf
riscv64-unknown-elf-objdump -D -M numeric,no-aliases -t test.elf

test.elf:     file format elf32-littleriscv

SYMBOL TABLE:
80000000 l    d  .text  00000000 .text
80000040 l    d  .data  00000000 .data
00000000 l    df *ABS*  00000000 test.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 loop
8000003c l       .text  00000000 forever
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 <loop>:
8000000c:       0000a103                lw      x2,0(x1)
80000010:       002f8fb3                add     x31,x31,x2
80000014:       0040a103                lw      x2,4(x1)
80000018:       002f8fb3                add     x31,x31,x2
8000001c:       0080a103                lw      x2,8(x1)
80000020:       002f8fb3                add     x31,x31,x2
80000024:       00c0a103                lw      x2,12(x1)
80000028:       002f8fb3                add     x31,x31,x2
8000002c:       01008093                addi    x1,x1,16
80000030:       fffa0a13                addi    x20,x20,-1
80000034:       fc0a1ce3                bne     x20,x0,8000000c <loop>
80000038:       001f8f93                addi    x31,x31,1

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

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
80000058:       00000007                0x7
8000005c:       0008                    c.addi4spn      x10,x2,0
        ...
riscv64-unknown-elf-objcopy -O binary --reverse-bytes=4 test.elf test.bin
xxd -g 4 -c 4 -p test.bin test.hex
rm test.bin test.o test.elf
```

Рисунок 1. Скомпилированная программа test

```
80000000 <_start>:
80000000:       00200a13                addi    x20,x0,2
80000004:       00000097                auipc   x1,0x0 (1)
80000008:       03c08093                addi    x1,x1,60 # 80000040 <_x>

8000000c <loop>:
8000000c:       0000a103                lw      x2,0(x1)
80000010:       002f8fb3                add     x31,x31,x2
80000014:       0040a103                lw      x2,4(x1)
80000018:       002f8fb3                add     x31,x31,x2
8000001c:       0080a103                lw      x2,8(x1)
80000020:       002f8fb3                add     x31,x31,x2
80000024:       00c0a103                lw      x2,12(x1)
80000028:       002f8fb3                add     x31,x31,x2
8000002c:       01008093                addi    x1,x1,16
80000030:       fffa0a13                addi    x20,x20,-1
80000034:       fc0a1ce3                bne     x20,x0,8000000c <loop>
80000038:       001f8f93                addi    x31,x31,1

8000003c <forever>:
8000003c:       0000006f                jal     x0,8000003c <forever>
```

Рисунок 2. Код тестовой программа

Рассматриваемый вариант – 14

Ниже приведен листинг программы этого варианта.

Вариант 14

```
        .section .text
        .globl _start;
        len = 9 #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 элемента, следующего за последним
        lw x31, 0(x1)
        add x1, x1, elem_sz*1
lp:
        lw x2, 0(x1)
        bltu x2, x31, lt
        add x31, x0, x2 #!
lt:
        add x1, x1, elem_sz*enroll
        bne x1, x20, lp
lp2: j lp2

        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x8
        .4byte 0x6
        .4byte 0x7
        .4byte 0x5
        .4byte 0x4
```

Рисунок 3. Код программы в.14

Код на языке C, соответствующий данной программе:

```
#include <stdio.h>
#include <stdlib.h>

#define len 9
#define enroll 1
#define elem_sz 4

int _x[] = {1, 2, 3, 4, 8, 6, 7, 5, 4};

int main() {
   int *x20 = _x + len + 1;
   int *x1 = _x;
   int x31 = x1[0];
   x1 += 1;

   do {
      int x2 = x1[0];
      if (x2 >= x31) {
         x31 = x2;
      }
      x1 += enroll;
   } while(x1 != x20);

   printf("%d\n", x31);

   while (1) {};

   return 0;
}
```

В x31 в конце программы будет находиться число 8.

Ниже приведен дизассемблерный листинг программы.

```
Disassembly of section .text:

80000000 <_start>:
80000000:       00000097                auipc   x1,0x0
80000004:       02c08093                addi    x1,x1,44 # 8000002c <_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:       01f16463                bltu    x2,x31,80000020 <lt>
8000001c:       00200fb3                add     x31,x0,x2

80000020 <lt>:
80000020:       00408093                addi    x1,x1,4
80000024:       ff4098e3                bne     x1,x20,80000014 <lp>

80000028 <lp2>:
80000028:       0000006f                jal     x0,80000028 <lp2>

Disassembly of section .data:

8000002c <_x>:
8000002c:       0001                    c.addi  x0,0
8000002e:       0000                    unimp
80000030:       0002                    0x2
80000032:       0000                    unimp
80000034:       00000003                lb      x0,0(x0) # 0 <enroll-0x1>
80000038:       0004                    c.addi4spn      x9,x2,0
8000003a:       0000                    unimp
8000003c:       0008                    c.addi4spn      x10,x2,0
8000003e:       0000                    unimp
80000040:       0006                    0x6
80000042:       0000                    unimp
80000044:       00000007                0x7
80000048:       0005                    c.addi  x0,1
8000004a:       0000                    unimp
8000004c:       0004                    c.addi4spn      x9,x2,0
```

Рисунок 4. Скомпилированная программа

Задание 2

Адрес команды: 80000014

Номер итерации: 2

Выборка и диспетчеризация.

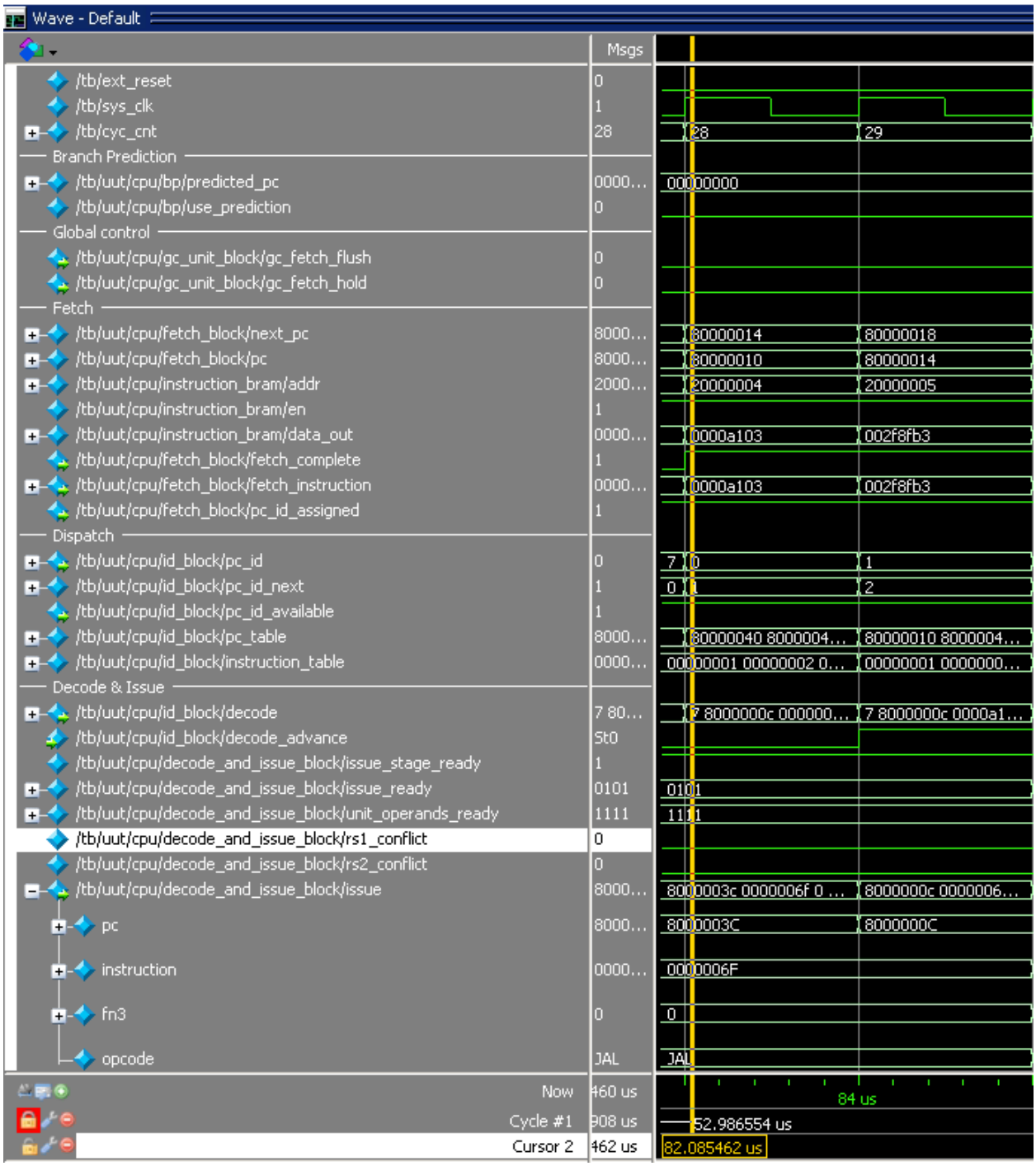

Рисунок 5. Выборка и диспетчеризация

Задание 3

Адрес команды: 80000020

Номер итерации: 2

Декодирование и планирование.

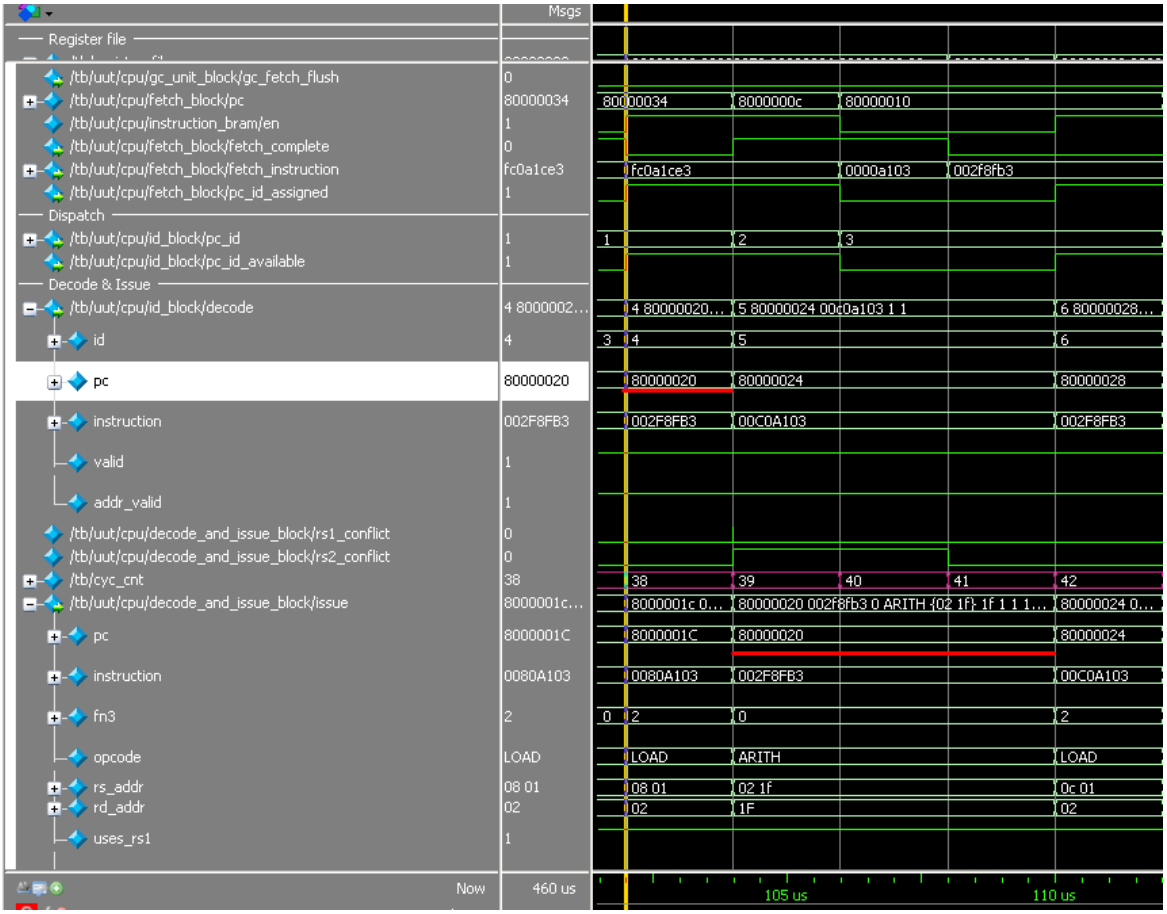

Рисунок 6. Декодирование и планирование

Задание 4

Адрес команды: 8000000c

Номер итерации: 2

Выполнение.

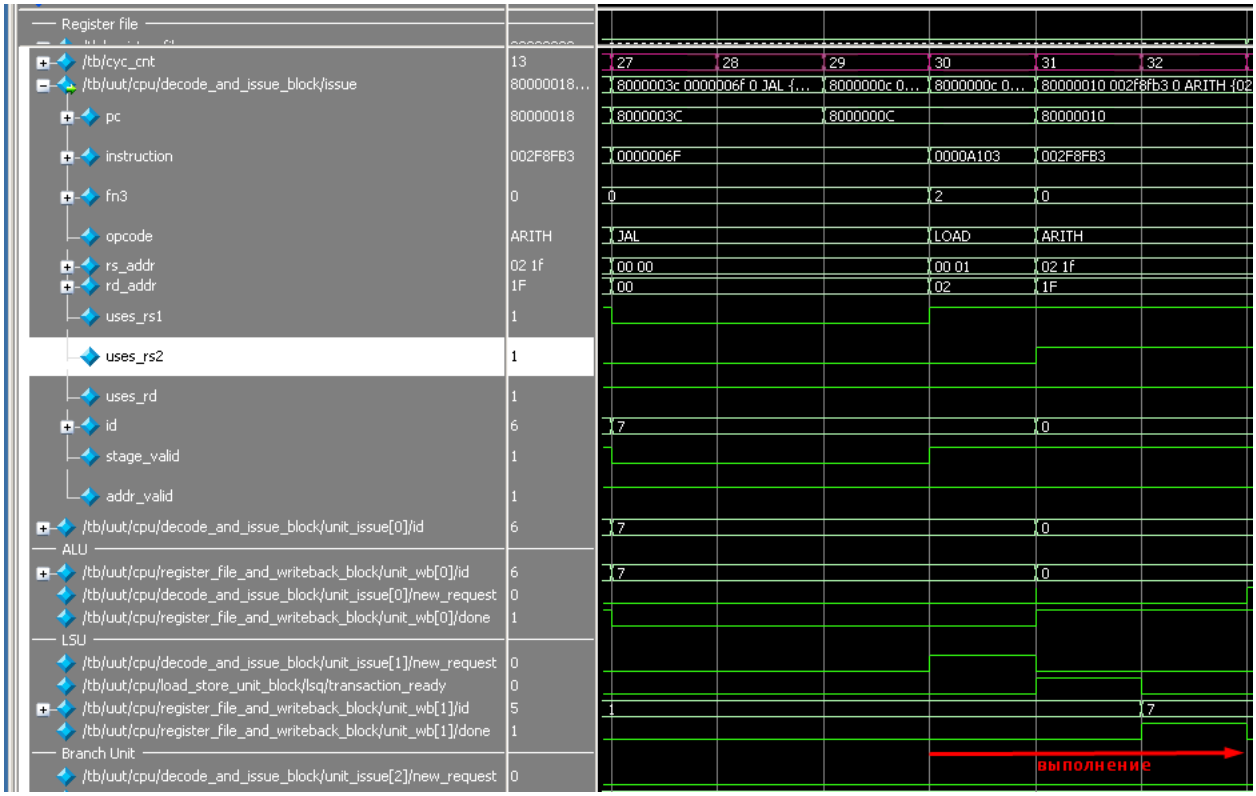

Рисунок 7. Выполнение

Задание 5

Адрес команды с #!: 80000001c

Выборка, диспетчеризация, декодирование, выполнение:

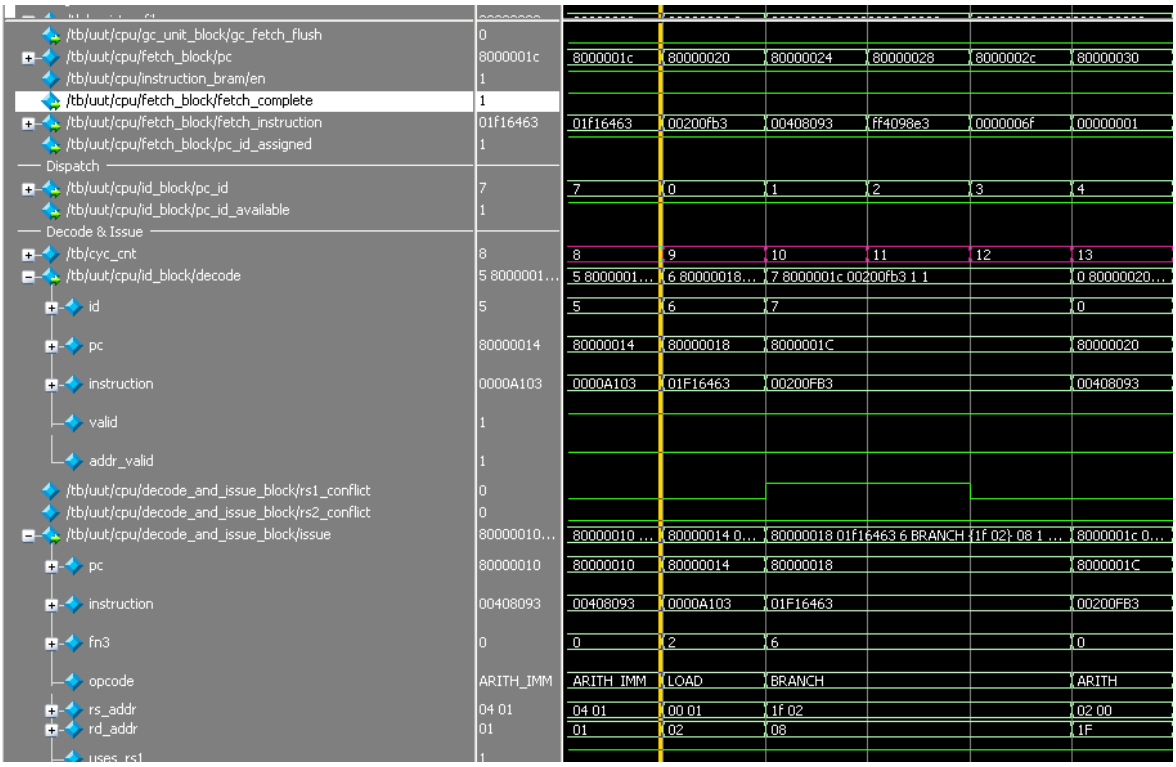

Рисунок 8. Выборка, диспетчеризация, декодирование

Выполнение:

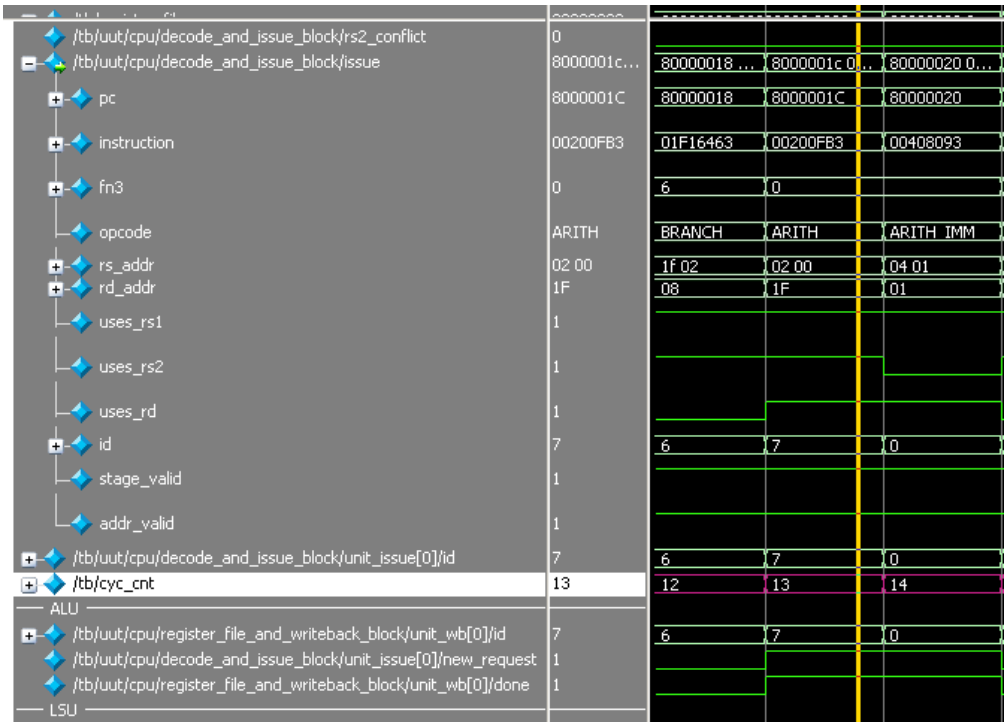

Рисунок 9. Выполнение

Трасса программы:

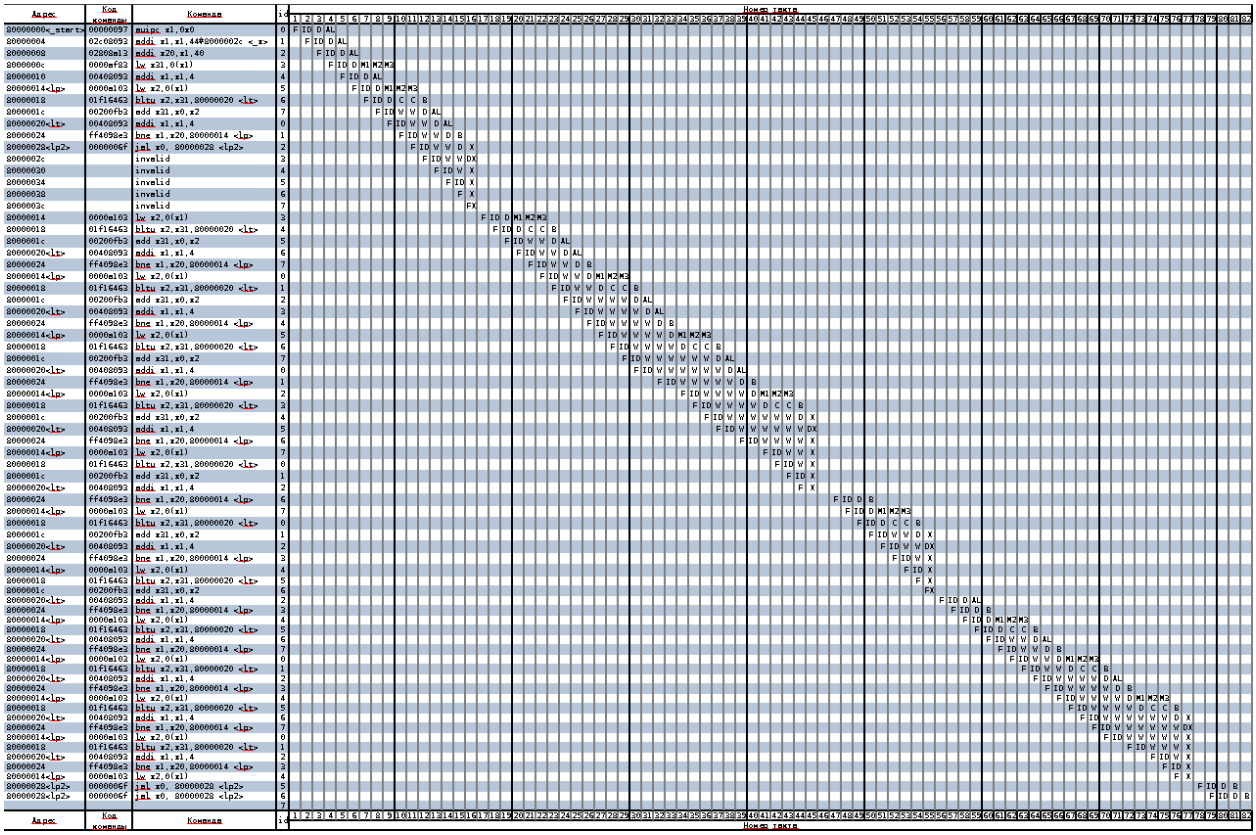

Рисунок 10. Трасса изначальной программы

Из трассы видно возникновение конфликтов, которые замедляют работу программы. Для оптимизации можно перенести команду addi x1,x1,4 в место между конфликтующими командами.

Ниже приведены исходный и оптимизированный коды программы моего варианта.

```
Disassembly of section .text:

80000000 <_start>:
80000000:       00000097                auipc   x1,0x0
80000004:       02c08093                addi    x1,x1,44 # 8000002c <_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:       01f16463                bltu    x2,x31,80000020 <lt>
8000001c:       00200fb3                add     x31,x0,x2

80000020 <lt>:
80000020:       00408093                addi    x1,x1,4
80000024:       ff4098e3                bne     x1,x20,80000014 <lp>

80000028 <lp2>:
80000028:       0000006f                jal     x0,80000028 <lp2>

Disassembly of section .data:

8000002c <_x>:
8000002c:       0001                    c.addi  x0,0
8000002e:       0000                    unimp
80000030:       0002                    0x2
80000032:       0000                    unimp
80000034:       00000003                lb      x0,0(x0) # 0 <enroll-0x1>
80000038:       0004                    c.addi4spn      x9,x2,0
8000003a:       0000                    unimp
8000003c:       0008                    c.addi4spn      x10,x2,0
8000003e:       0000                    unimp
80000040:       0006                    0x6
80000042:       0000                    unimp
80000044:       00000007                0x7
80000048:       0005                    c.addi  x0,1
8000004a:       0000                    unimp
8000004c:       0004                    c.addi4spn      x9,x2,0
```

Рисунок 11. Код изначальной программы

```
Disassembly of section .text:

80000000 <_start>:
80000000:       00000097                auipc   x1,0x0
80000004:       02c08093                addi    x1,x1,44 # 8000002c <_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:       00408093                addi    x1,x1,4
8000001c:       01f16463                bltu    x2,x31,80000024 <lt>
80000020:       00200fb3                add     x31,x0,x2

80000024 <lt>:
80000024:       ff4098e3                bne     x1,x20,80000014 <lp>

80000028 <lp2>:
80000028:       0000006f                jal     x0,80000028 <lp2>

Disassembly of section .data:

8000002c <_x>:
8000002c:       0001                    c.addi  x0,0
8000002e:       0000                    unimp
80000030:       0002                    0x2
80000032:       0000                    unimp
80000034:       00000003                lb      x0,0(x0) # 0 <enroll-0x1>
80000038:       0004                    c.addi4spn      x9,x2,0
8000003a:       0000                    unimp
8000003c:       0008                    c.addi4spn      x10,x2,0
8000003e:       0000                    unimp
80000040:       0006                    0x6
80000042:       0000                    unimp
80000044:       00000007                0x7
80000048:       0005                    c.addi  x0,1
8000004a:       0000                    unimp
8000004c:       0004                    c.addi4spn      x9,x2,0
          ...
```

Рисунок 12. Код оптимизированной программы

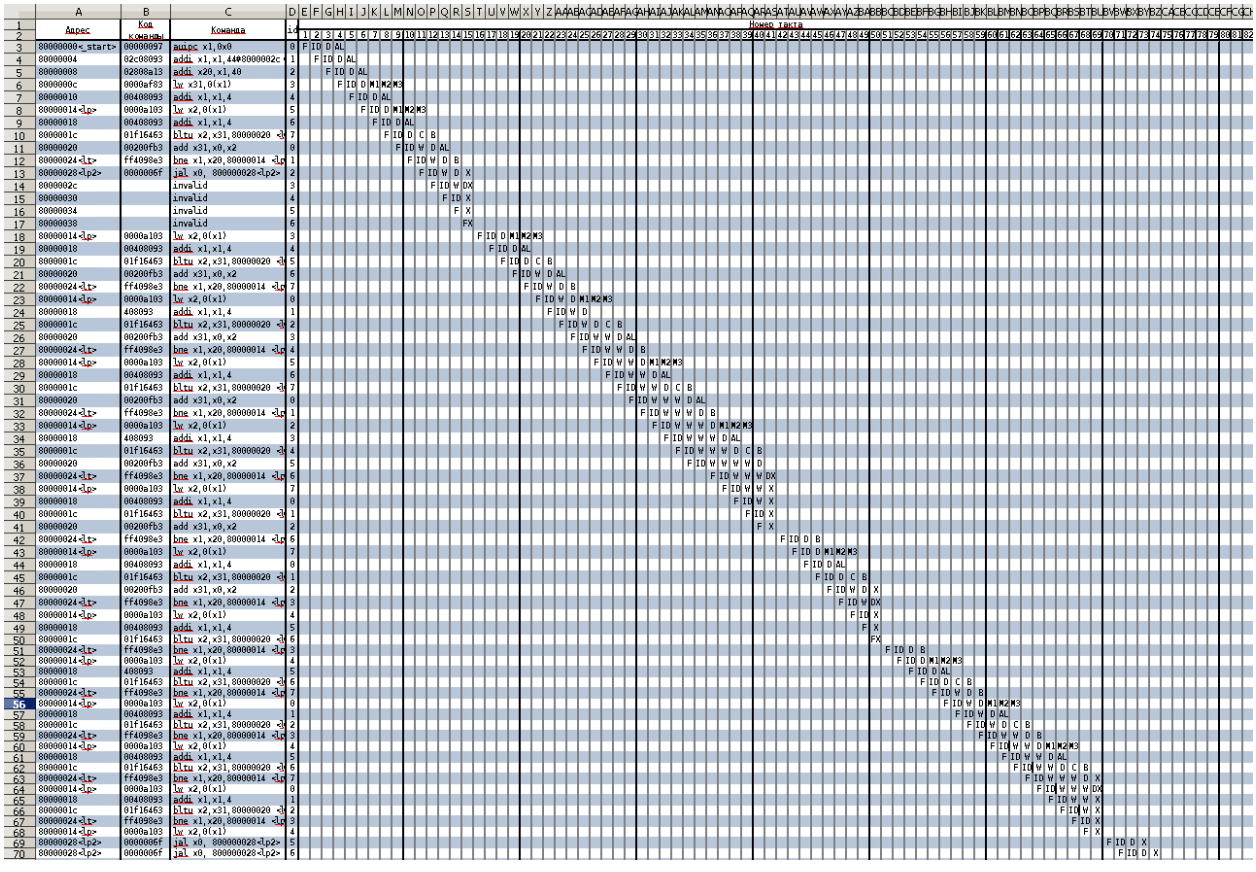

Рисунок 13. Трасса оптимизированной программы

Оптимизированная программа работает на 8 тактов быстрее.

Заключение

В ходе лабораторной работы я ознакомилась с принципами функционирования и построения суперскалярных конвейерных микропроцессоров.