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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 2

Название: Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина: Архитектура ЭВМ

Студент	ИУ7-52Б		И.С. Климов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			А.Ю. Попов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

Содержание

Введение

Цель работы – ознакомление с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров, а также знакомство с принципами проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС. Для достижения данной цели необходимо выполнить следующие **задачи**:

1) познакомиться с набором команд RV32I;
2) познакомиться с основными принципами работы ядра Taiga: изучить операции, выполняемые на каждой стадии обработки команд;
3) на основе полученных знаний проанализировать ход выполнения программы и при возможности оптимизировать ее.

1. Теоретическая часть

Ниже будут представлены теоретические сведения об архитектуре набора команд RV32I и микроархитектуре, используемой в лабораторной работе.

1.1. Архитектура набора команд RV32I

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.

Набор команд RV32I предполагает использование 32 регистров общего назначения x0-x31 размером в 32 бита каждый и регистр pc, хранящего адрес следующей команды. Все регистры общего назначения равноправны, в любой команде могут использоваться любые из регистров. Регистр pc не может использоваться в командах.

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. Микроархитектура

В лабораторной работе рассматривается система, состоящая из вычислительного ядра Taiga и локальной памяти, реализованной с помощью блочной памяти ПЛИС. Команды и данные находятся в едином адресном пространстве. Дешифратор адресов настроен таким образом, что блок памяти ПЛИС отображается в адресное пространство RISC-V с адреса 0x80000000. Память ПЛИС имеет фиксированную задержку доступа в 1 такт, в связи с чем отпадает необходимость в кеш-памяти.

Taiga является конвейерным микропроцессором с элементами суперскалярности. При конвейерной организации микропроцессора различные команды одновременно проходят различные стадии своей обработки. Конвейер Taiga насчитывает 4 стадии:

1. Выборка (F) — стадия, на которой команда извлекается из ПК. Выполняется в блоке выборки.
2. Диспетчеризация (ID) — стадия, на которой происходит запись команды в очередь команд для декодирования. Выполняется в блоке управления метаданными.
3. Декодирование и планирование на выполнение (D) — стадия на которой происходит определение типа и полей команды и определение вычислительного блока, способного ее исполнить. Выполняется в блоке декодирования и планирования на выполнение.
4. Выполнение (AL, M1..M3, в зависимости от исполнительного блока) — стадия, на которой команда передается в блок выполнения.

"Ширина" конвейера Taiga равна 1 для всех стадий, кроме стадии выполнения. В лучшем случае, каждая стадия конвейера выполняется за один такт. В состав рассматриваемой конфигурации Taiga входит 3 блока выполнения команд:

1) арифметико-логическое устройство (АЛУ);
2) блок доступа к памяти (LSU);
3) блок ветвлений.

АЛУ и блок ветвлений выполняют команды за 1 такт, LSU — минимум за 3.

На рисунке 1.1 приведена структурная схема ядра Taiga.

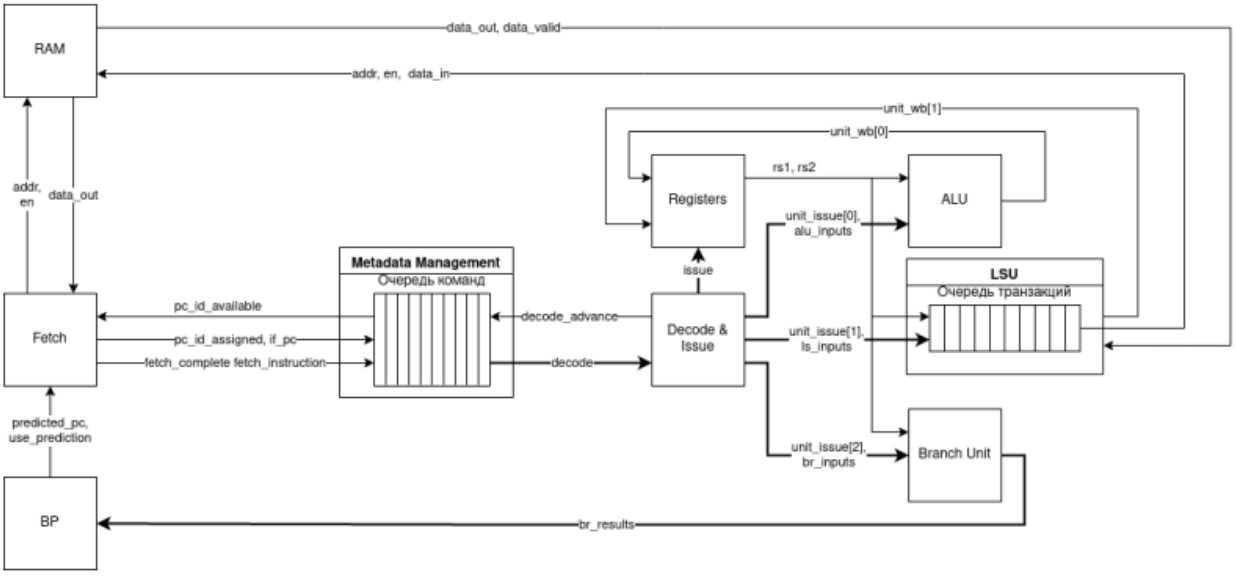

Рисунок 1.1 – обобщенная структурная схема ядра Taiga.

2. Практическая часть

Далее представлен ход выполнения лабораторной работы.

2.1. Задание 1

На листинге 2.1 представлен код исходной программы.

Листинг 2.1 – исходный код программы

```
        .section .text
        .globl _start;
        len = 8 #Размер массива
        enroll = 2 #Количество обрабатываемых элементов за одну итерацию
       elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x1, elem_sz*len #Адрес последнего элемента
lp:
        lw x2, 0(x1)
        lw x3, 4(x1)
        addi x1, x1, elem_sz*enroll
        add x31, x31, x2 #!
        add x31, x31, x3
        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
```

На листинге 2.2 представлен дизассемблированный код исходной программы.

Листинг 2.2 – дизассемблированный код исходной программы

```
Disassembly of section .text:

80000000 <_start>:
80000000:       00000097                auipc   x1,0x0
80000004:       02c08093                addi    x1,x1,44 # 8000002c <_x>
80000008:       02008a13                addi    x20,x1,32

8000000c <lp>:
8000000c:       0000a103                lw      x2,0(x1)
80000010:       0040a183                lw      x3,4(x1)
80000014:       00808093                addi    x1,x1,8
80000018:       002f8fb3                add     x31,x31,x2
8000001c:       003f8fb3                add     x31,x31,x3
80000020:       ff4096e3                bne     x1,x20,8000000c <lp>
80000024:       001f8f93                addi    x31,x31,1

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-0x2>
80000038:       0004                    c.addi4spn      x9,x2,0
8000003a:       0000                    unimp
8000003c:       0005                    c.addi  x0,1
8000003e:       0000                    unimp
80000040:       0006                    0x6
80000042:       0000                    unimp
80000044:       00000007                0x7
80000048:       0008                    c.addi4spn      x10,x2,0
```

На листинге 2.3 представлен псевдокод на языке C эквивалентной программы.

Листинг 2.3 – псевдокод на языке C

```
#define len 8
#define enroll 2
#define elem_sz 4


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


void _start() {
        int *x1 = _x;
        int *x20 = x1 + len;
        int x2, x3, x31 ;

        do {
                x2 = x1[0];
                x3 = x1[1];

                x1 += enroll;
                x31 += x2;
                x31 += x3;
        } while (x1 != x20);
        x31++;

        while (1) {}
}
```

В результате выполнения в регистре x31 будет находиться инкрементированная сумма элементов массива, равная 37.

2.2. Задание 2

В соответствии вариантом 8 получим временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 80000028, 1-я итерация, которая представлена на рисунке 2.1.

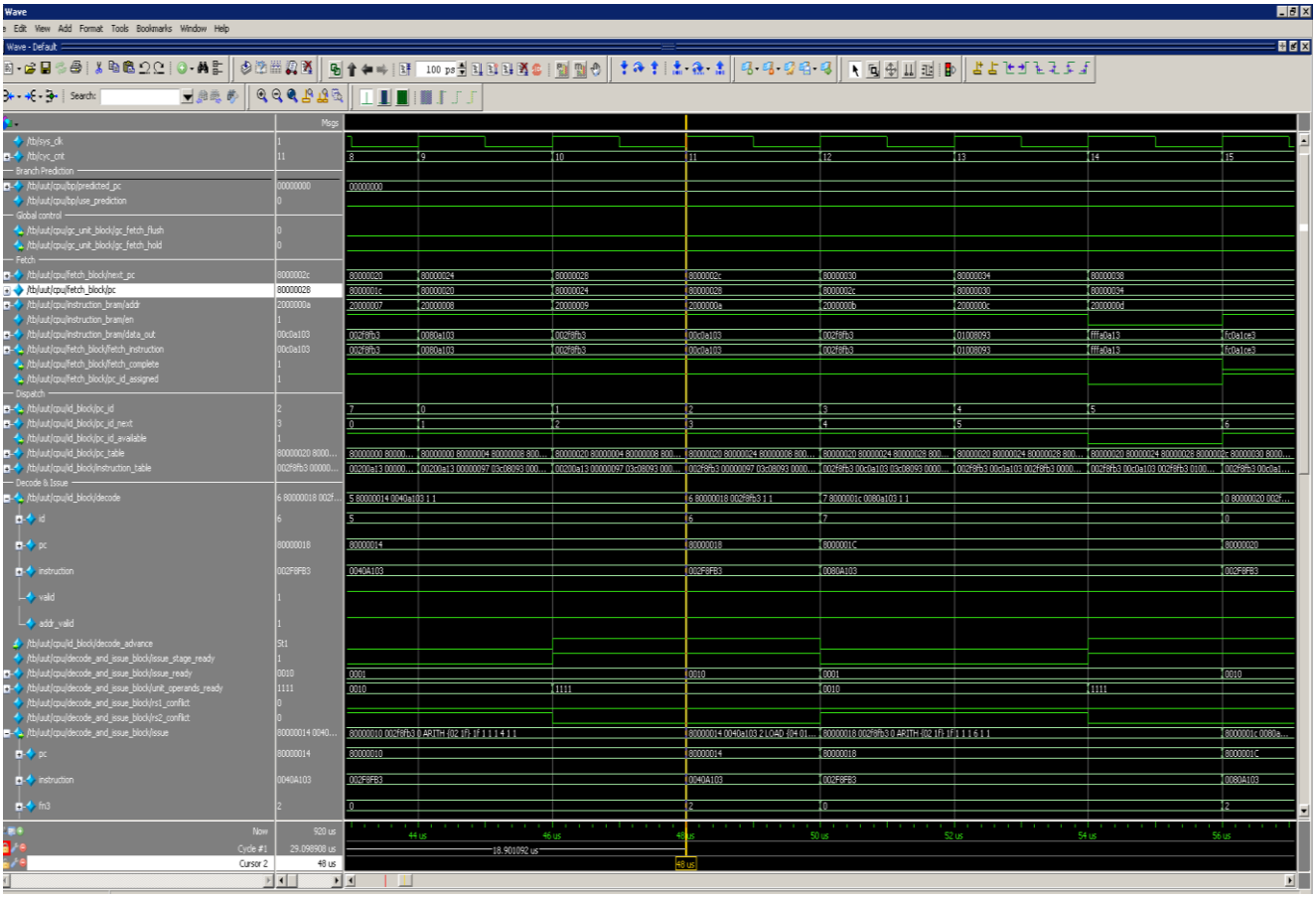

Рисунок 2.1 – временная диаграмма выполнения стадий выборки и диспетчеризации команды

2.3. Задание 3

В соответствии с вариантом 8 получим временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 80000034, 1-я итерация, которая представлена на рисунке 2.2.

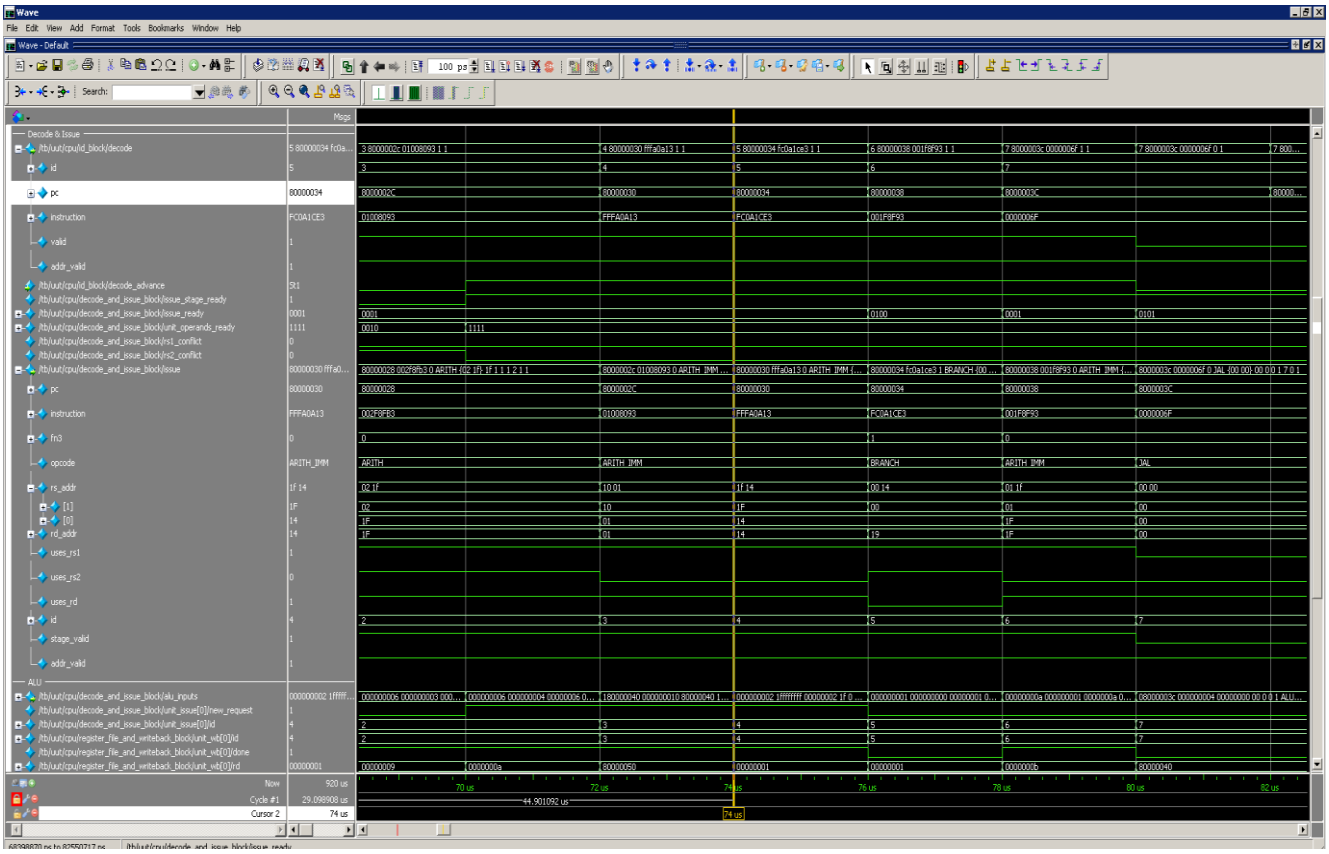

Рисунок 2.2 – временная диаграмма выполнения стадии декодирования и планирования на выполнение команды

2.4. Задание 4

В соответствии с вариантом 8 получим временную диаграмму выполнения стадии выполнения команды с адресом 8000001c, 1-я итерация, которая представлена на рисунке 2.3.

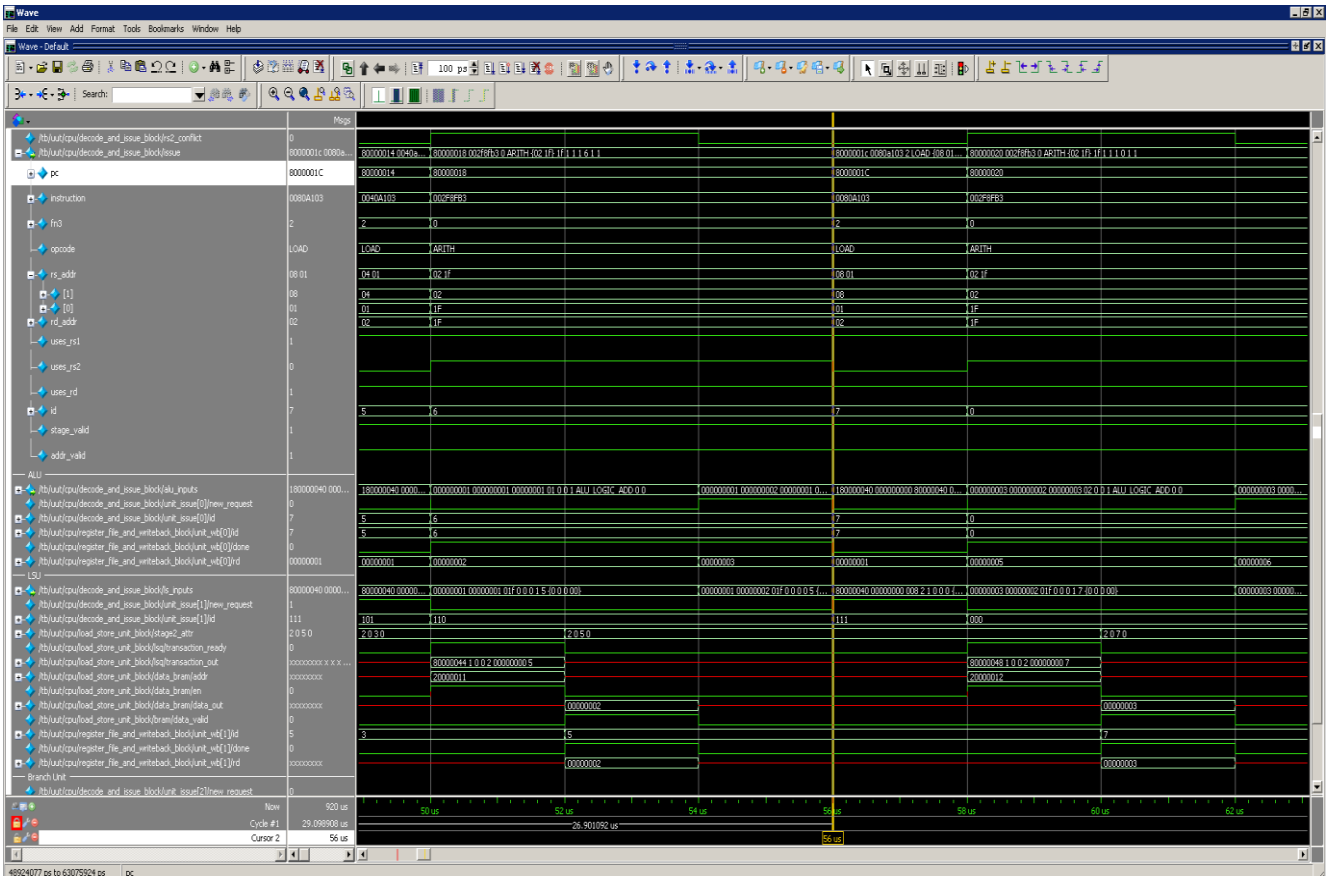

Рисунок 2.3 - диаграмма, соответствующая этапу выполнения

2.5. Задание 5

На рисунках 2.4 – 2.6 показаны временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом #!.

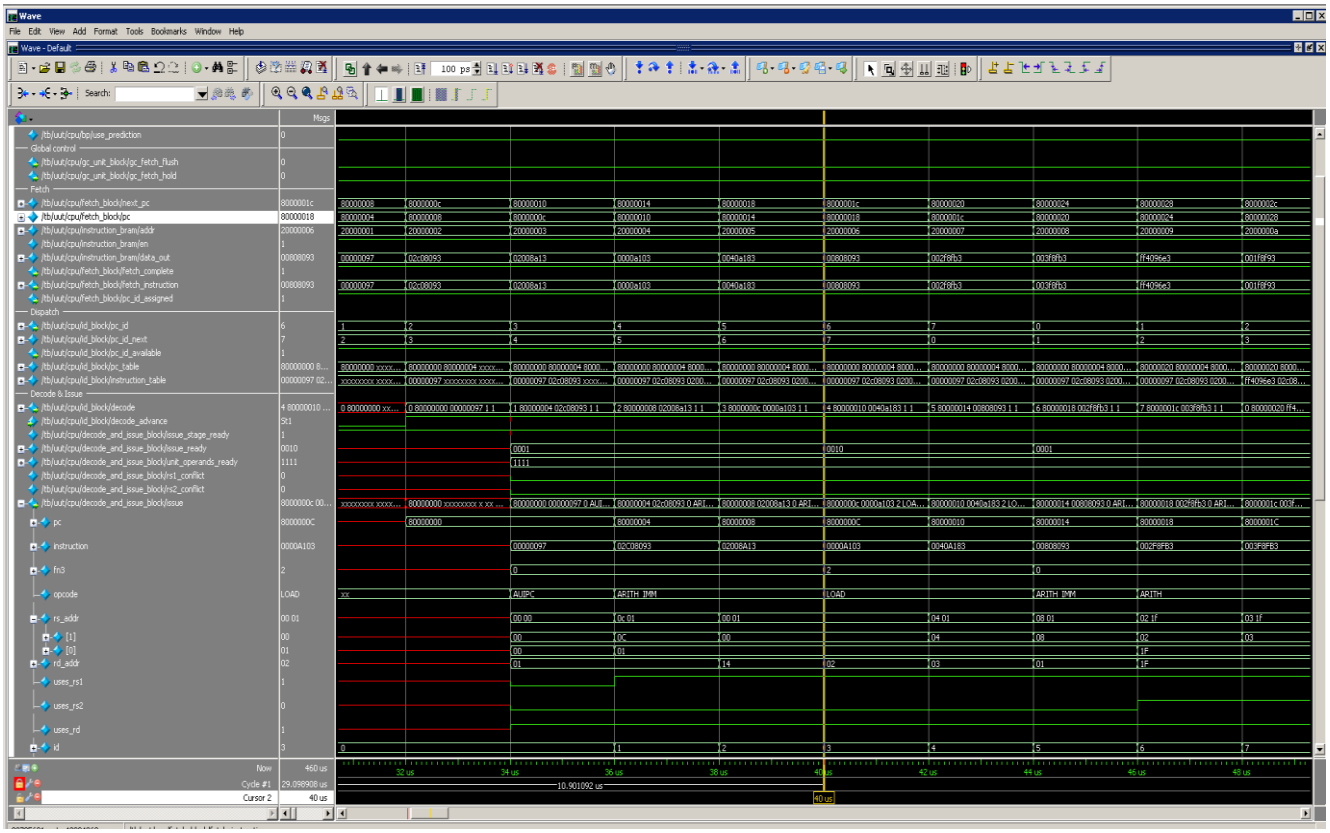

Рисунок 2.4 – диаграмма, соответствующая этапам выборки и диспетчеризации команды, обозначенной в тексте программы символом #!

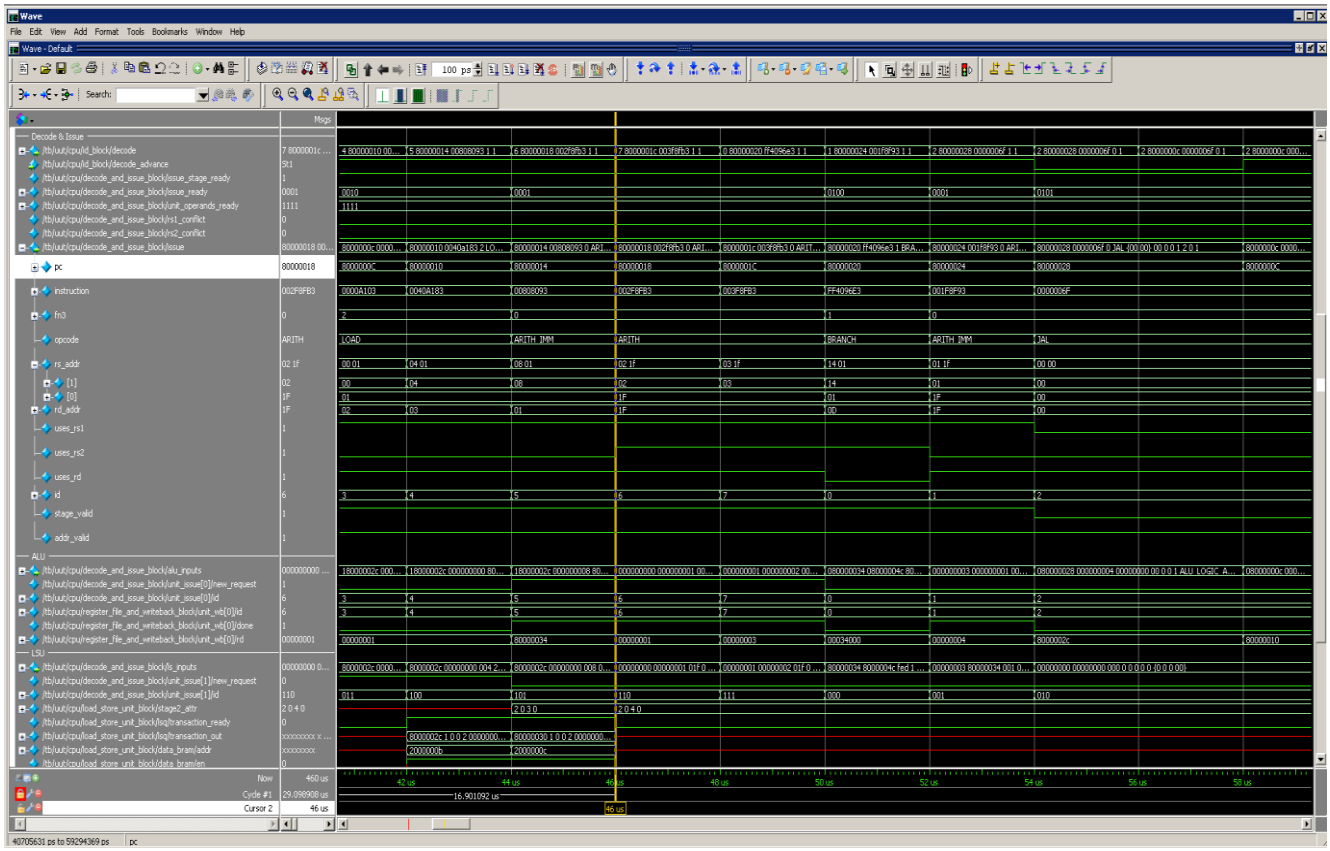

Рисунок 2.5 – диаграмма, соответствующая этапам декодирования и планирования команды, обозначенной в тексте программы символом #!

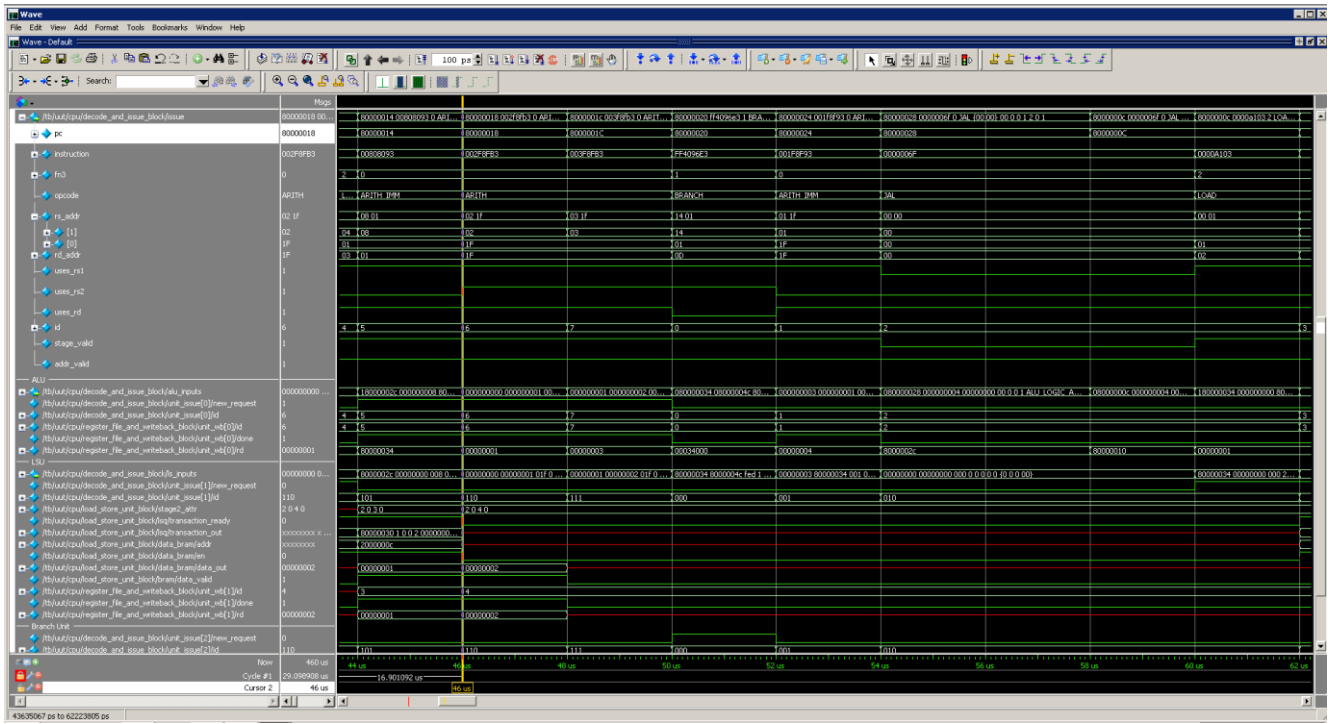

Рисунок 2.6 – диаграмма, соответствующая этапу выполнения команды, обозначенной в тексте программы символом #!

Результат выполнения программы сохранен в регистр x31. Как и было рассчитано теоретически, в момент окончания работы в нем хранится значение 0x25 (рисунок 2.7).

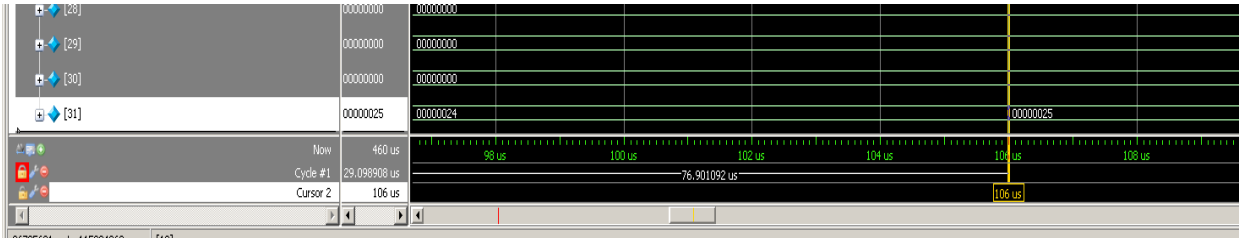

Рисунок 2.7 — Результат выполнения программы

Далее необходимо проанализировать диаграмму и заполнить трассу выполнения программы. На рисунке 2.8 приведена заполненная трасса.

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	43	44	45
80000000<_start	00000097	auipc x1,0x0	0	F	ID	D	AL																																									
80000004	02c08093	addi x1,x1,44 #8000002c<_x>	1		F	ID	D	AL																																								
80000008	02008a13	addi x20,x1,32	2			F	ID	D	AL																																						
8000000c<lp>	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	00808093	addi x1,x1,8	5						F	ID	D	AL																																		
80000018	002f8fb3	add x31,x31,x2	6							F	ID	D	AL																																
8000001c	003f8fb3	add x31,x31,x3	7								F	ID	D	AL																														
80000020	ff4096e3	bne x1,x20,8000000c<lp>	0									F	ID	D	B																												
8000000c<lp>	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	00808093	addi x1,x1,8	5												F	ID	D	AL																										
80000018	002f8fb3	add x31,x31,x2	6													F	ID	D	AL																									
8000001c	003f8fb3	add x31,x31,x3	7														F	ID	D	AL																								
80000020	ff4096e3	bne x1,x20,8000000c<lp>	0															F	ID	D	B																							
8000000c<lp>	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	00808093	addi x1,x1,8	5																		F	ID	D	AL																		
80000018	002f8fb3	add x31,x31,x2	6																			F	ID	D	AL																	
8000001c	003f8fb3	add x31,x31,x3	7																				F	ID	D	AL																
80000020	ff4096e3	bne x1,x20,8000000c<lp>	0																					F	ID	D	B															
8000000c<lp>	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	00808093	addi x1,x1,8	5																								F	ID	D	AL											
80000018	002f8fb3	add x31,x31,x2	6																									F	ID	D	AL										
8000001c	003f8fb3	add x31,x31,x3	7																										F	ID	D	AL									
80000020	ff4096e3	bne x1,x20,8000000c<lp>	0																											F	ID	D	B								
8000000c<lp>	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	00808093	addi x1,x1,8	5																													F	ID	D	AL					
80000018	002f8fb3	add x31,x31,x2	6																														F	ID	D	AL				
8000001c	003f8fb3	add x31,x31,x3	7																															F	ID	D	AL			
80000020	ff4096e3	bne x1,x20,8000000c<lp>	0																																F	ID	D	B		
8000000c<lp>	0000a103	lw x2,0(x1)	3																																	F	ID	D	X		
80000010	0040a183	lw x3,4(x1)	4																																		F	ID	X		
80000014	00808093	addi x1,x1,8	5																																			F	X		
80000018	002f8fb3	add x31,x31,x2	6																																				FX		
80000024	001f8f93	add x31,x31,1	1																																					F	ID	D	AL
80000028	0000006f	jal x0,8000003c<lp2>	2																																					F	ID	D	X
8000002c	00000001	<invalid operation>	4																																					F	ID	X	
80000030	00000002	<invalid operation>	5																																					F	X	
80000034	00000003	<invalid operation>	3																																						FX	
80000038	00000004	<invalid operation>	4																																						FX	
80000028	0000006f	jal x0,8000003c<lp2>	2																																							F	ID	D	B
80000028	0000006f	jal x0,8000003c<lp2>	2																																							F	ID	D	B
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	43	44	45

Рисунок 4.5 — Трасса работы программы

Можно заметить, что при данной реализации конфликтов не наблюдается, из чего можно сделать вывод, что программа уже является оптимизированной.

Вывод

В результате выполнения лабораторной работы была достигнута главная цель, которая заключалась в ознакомлении с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров. При этом были решены поставленные задачи.

В результате анализа программы варианта 8 не было обнаружено конфликтов. То есть программа уже является оптимизированным вариантом.