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 RISC-V

Дисциплина Архитектура элекронно-вычислительных машин

Студент: _____ Ковель А.Д.
подпись, дата Фамилия, И.О.

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

Москва — 2022 г.

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.

Основные теоретические сведения

RISC-V является открытым современным набором команд, который может использоваться для построения как микроконтроллеров, так и высокопроизводительных микропроцессоров.

В данной работе исследуется набор команд RV32I, который включает в себя основные команды 32-битной целочисленной арифметики кроме умножения и деления.

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.

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 (кроме команды auipc) 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.

Общая для всех вариантов программа

Исследуемая программа

Исходный текст исследуемой программы представлен на листинге 1.

Листинг 1 – Исходный текст общей программы

```
1  .section .text (1)
2  .globl _start; (2)
3  len = 8 # size of array (3)
4  enroll = 4 #amount of processed elements in one iteration
5  elem_sz = 4 #size of one element in array
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
```

Дизассемблерный листинг исследуемой программы представлен на листинге 2.

Листинг 2 – Дизассемблерный листинг общей программы

```
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>
```

Можно сказать, что данная программа эквивалентна псевдокоду на языке C, представленному на листинге 3.

Листинг 3 – Псевдокод общей программы

```
#define len 8
#define enroll 4
#define elem_sz 4
int _x[]={1,2,3,4,5,6,7,8};
void _start() {
    int x20 = len/enroll;
    int *x1 = _x;

    do {
        int x2 = x1[0];
        x31 += x2;
        x2 = x1[1];
        x31 += x2;
        x2 = x1[2];
        x31 += x2;
        x2 = x1[3];
        x31 += x2;
        x1 += enroll;
        x20--;
    } while(x20 != 0);
    x31++;
    while(1){}
}
```

Результаты исследования программы

Все задания выполнялись по индивидуальному варианту (6).

Скриншот, полученный в ходе выполнения задания №2 (получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 80000020 на первой итерации) представлен на рисунке 1.

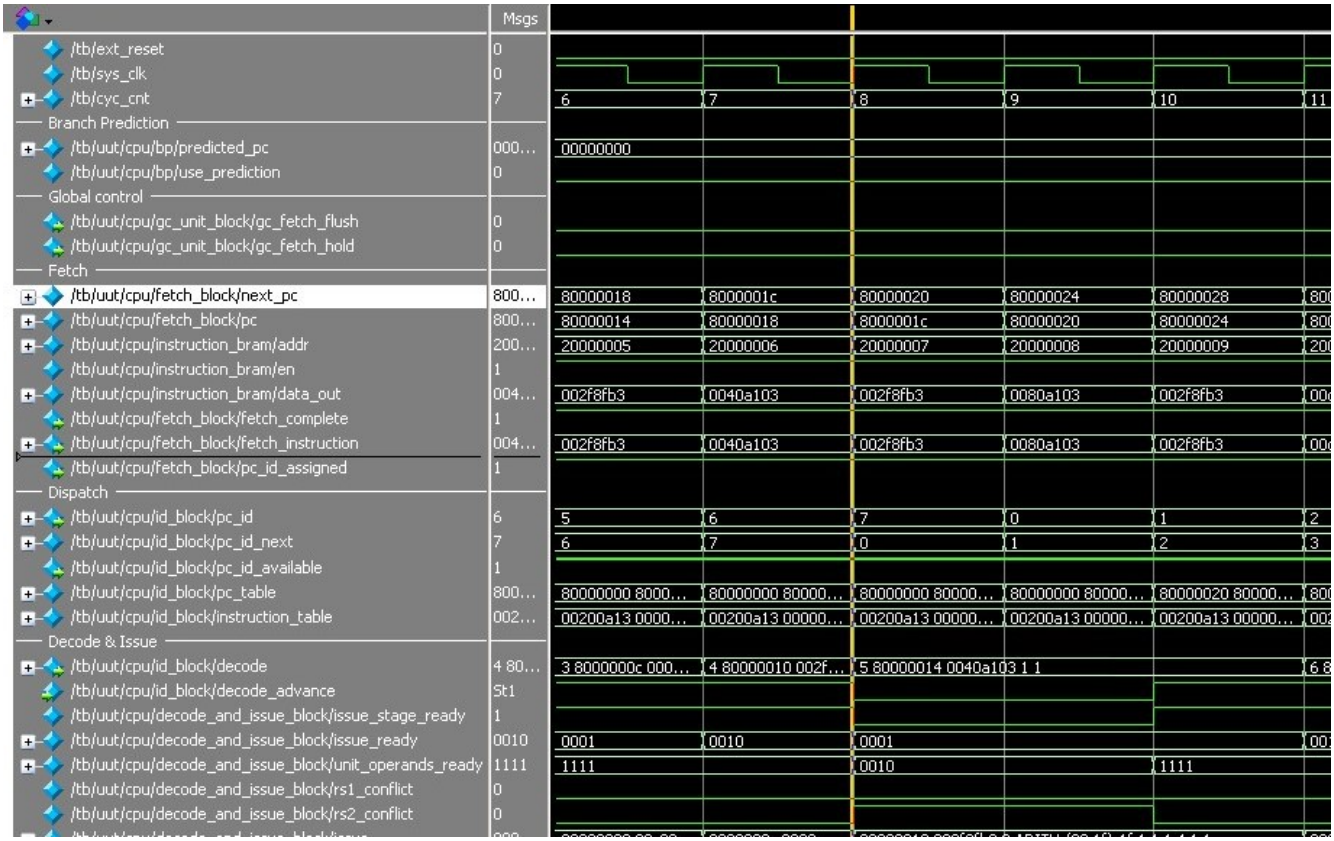

Рисунок 1 – Временная диаграмма выполнения стадий выборки и диспетчеризации

Скриншот, полученный в ходе выполнения задания №3 (получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования на выполнение команды с адресом 8000002c на первой итерации) представлен на рисунке 2.

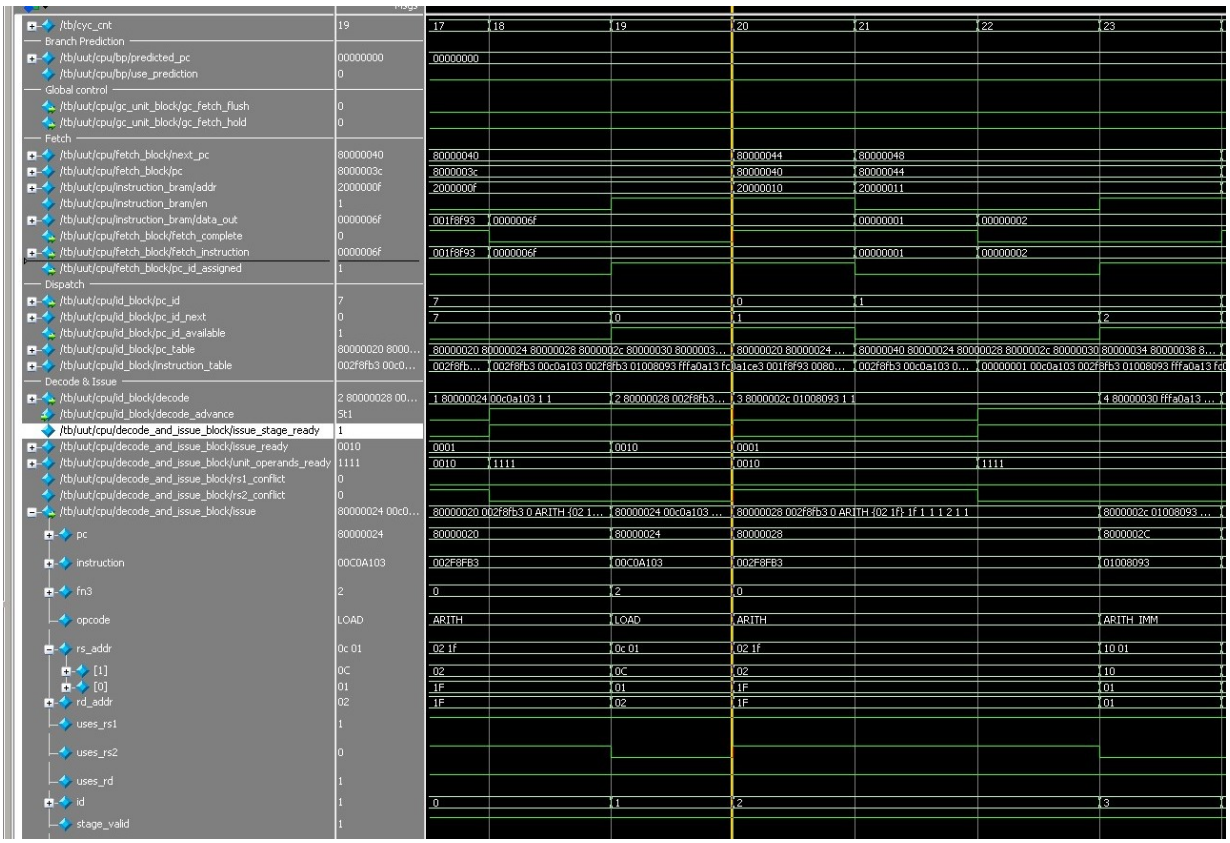

Рисунок 2 – Временная диаграмма выполнения стадии декодирования и планирования на выполнение

Скриншот, полученный в ходе выполнения задания №4 (получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды с адресом 80000014 на первой итерации) представлен на рисунке 3.

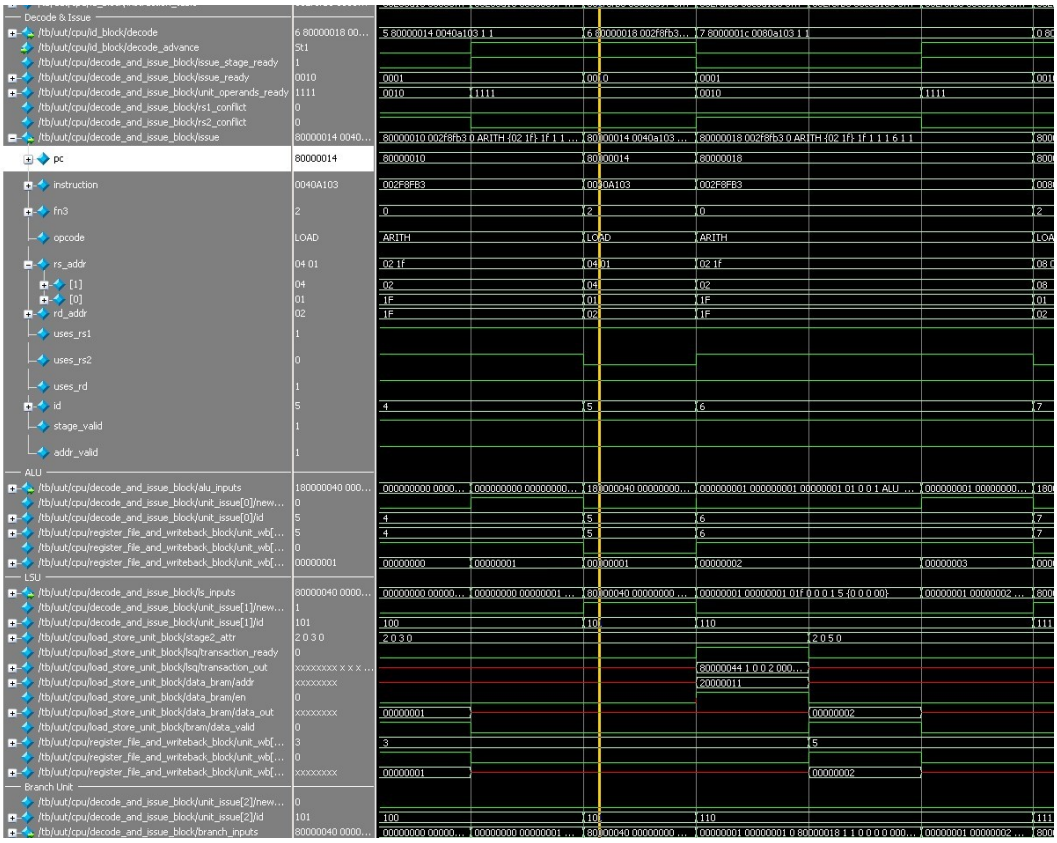

Рисунок 3 – Временная диаграмма выполнения стадии выполнения

Программа по варианту

Все задания выполнялись по индивидуальному варианту (6).

Исследуемая программа

Исходный текст исследуемой программы представлен на листинге 4.

Листинг 4 – Исходный текст исследуемой программы

```
1     .section .text
2     .globl _start;
3     len = 8 # size of array
4     enroll = 2 #amount of processed elements in one iteration
5     elem_sz = 4 #size of one element in array
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    addi x1, x1, elem_sz*enroll
14    addi x20, x20, -1
15    add x31, x31, x2
16    add x31, x31, x3
17    bne x20, x0, lp
18    addi x31, x31, 1
19    lp2: j lp2
20
21    .section .data
22    _x:     .4byte 0x1
23    .4byte 0x2
24    .4byte 0x3
25    .4byte 0x4
26    .4byte 0x5
27    .4byte 0x6
28    .4byte 0x7
29    .4byte 0x8
```

Дизассемблерный листинг исследуемой программы представлен на листинге 5.

Листинг 5 – Дизассемблерный листинг исследуемой программы

```
1     80000000 <_start>:
2     80000000:       00400a13                addi    x20,x0,4
3     80000004:       00000097                auipc   x1,0x0
4     80000008:       02c08093                addi    x1,x1,44 # 80000030
        <_x>
5
6     8000000c <lp>:
7     8000000c:       0000a103                lw      x2,0(x1)
8     80000010:       0040a183                lw      x3,4(x1)
9     80000014:       00808093                addi    x1,x1,8
10    80000018:       fffa0a13                addi    x20,x20,-1
11    8000001c:       002f8fb3                add     x31,x31,x2
12    80000020:       003f8fb3                add     x31,x31,x3
13    80000024:       fe0a14e3                bne     x20,x0,8000000c <lp>
14    80000028:       001f8f93                addi    x31,x31,1
15
16    8000002c <lp2>:
17    8000002c:       0000006f                jal     x0,8000002c <lp2>
```

Можно сказать, что данная программа эквивалентна псевдокоду на языке C, представленному на листинге 6.

Листинг 6 – Псевдокод исследуемой программы

```
1   #define len 8
2   #define enroll 2
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          x1 += enroll;
13          x20--;
14          x31 += x2;
15          x31 += x3;
16      } while(x20 != 0);
17      x31++;
18      while(1){}
19  }
```

Трасса работы программы

Трасса работы программы представлена на рисунке 4.

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	46	47	48	49
80000000<_start>	00400a13	addi x20,x0,4	0	F	ID	D	AL																																													
80000004	00000097	auipc x1,0x0	1		F	ID	D	AL																																											
80000008	02c08093	addi x1,x1,44 # 80000030 <_x>	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	808093	addi x1,x1,8	5						F	ID	D	AL																																						
80000018	fffa0a13	addi x20,x20,-1	6							F	ID	D	AL																																					
8000001c	002f8fb3	add x31,x31,x2	7								F	ID	D	AL																																				
80000020	003f8fb3	add x31,x31,x3	0									F	ID	D	AL																																			
80000024	fe0a14e3	bne x20,x0,8000000c <lp>	1										F	ID	D	B																																		
80000028	001f8f93	addi x31,x31,1	2											F	ID	D	X																																	
8000002c<lp2>	0000006f	jal x0,8000002c <lp2>	3												F	ID	DX																																	
80000030	1		4													F	X																																	
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	808093	addi x1,x1,8	5																	F	ID	D	AL																													
80000018	fffa0a13	addi x20,x20,-1	6																		F	ID	D	AL																												
8000001c	002f8fb3	add x31,x31,x2	7																			F	ID	D	AL																											
80000020	003f8fb3	add x31,x31,x3	0																				F	ID	D	AL																										
80000024	fe0a14e3	bne x20,x0,8000000c <lp>	1																					F	ID	D	B																									
8000000c<lp>	0000a103	lw x2,0(x1)	2																						F	ID	D	M1	M2	M3																						
80000010	0040a183	lw x3,4(x1)	3																							F	ID	D	M1	M2	M3																					
80000014	808093	addi x1,x1,8	4																								F	ID	D	AL																						
80000018	fffa0a13	addi x20,x20,-1	5																									F	ID	D	AL																					
8000001c	002f8fb3	add x31,x31,x2	6																										F	ID	D	AL																				
80000020	003f8fb3	add x31,x31,x3	7																											F	ID	D	AL																			
80000024	fe0a14e3	bne x20,x0,8000000c <lp>	0																												F	ID	D	B																		
8000000c<lp>	0000a103	lw x2,0(x1)	1																													F	ID	D	M1	M2	M3															
80000010	0040a183	lw x3,4(x1)	2																														F	ID	D	M1	M2	M3														
80000014	808093	addi x1,x1,8	3																															F	ID	D	AL															
80000018	fffa0a13	addi x20,x20,-1	4																																F	ID	D	AL														
8000001c	002f8fb3	add x31,x31,x2	5																																	F	ID	D	AL													
80000020	003f8fb3	add x31,x31,x3	6																																		F	ID	D	AL												
80000024	fe0a14e3	bne x20,x0,8000000c <lp>	7																																			F	ID	D	B											
8000000c<lp>	0000a103	lw x2,0(x1)	0																																				F	ID	D	DX										
80000010	0040a183	lw x3,4(x1)	1																																					F	ID	X										
80000014	808093	addi x1,x1,8	2																																						F	X										
80000028	001f8f93	addi x31,x31,1	1																																								F	ID	D	AL						
8000002c<lp2>	0000006f	jal x0,8000002c <lp2>	2																																									F	ID	D	B					
80000030	1		3																																										F	ID	D	X				
80000034	2		4																																											F	ID	DX				
80000038	3		5																																												F	X				
8000002c<lp2>	0000006f	jal x0,8000002c <lp2>	4																																														F	ID	D	AL
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	43	44	45	46	47	48	49

Рисунок 4 – Трасса работы программы

Временные диаграммы

Временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом #! (80000010: 0040a183 lw x3,4(x1)), представлены на рисунке 5.

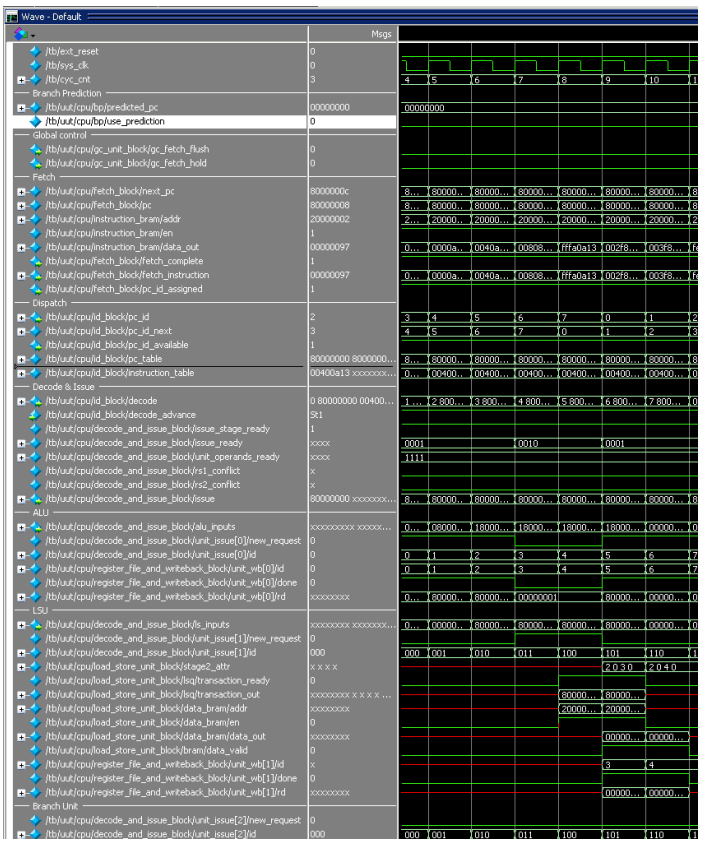

Рисунок 5 – Временные диаграммы сигналов

Вывод об эффективности работы программы

Как видно на трассе работы программы, представленой на рисунке 4, конфликта по регистрам не возникает ни разу, и все команды принимаются на исполнение сразу же по готовности. Это обеспечивается продуманным чередованием команд обращения к памяти и арифметических команд.

Можно также заметить, что трижды возникает ситуация ошибочной выборки, которая крайне негативно сказывается на производительности, так как приводит к необходимости очистки конвеера, и ожидания прохождения новой комаы по всем стадиям работы. Однако, учитывая большое количество ветвлений в данной программе (только сам цикл повторяется 4 раза), можно сказать, что предсказаель ветвлений ошибается не слишком часто.

Оптимизировать же программу можно, уменьшив количество циклов в ней. В исходном коде цикл повторяется 4 раза, и некоторые команды, не относящиеся напрямую к вычислению суммы элементов массива (во-первых, команда условного перехода bne x20, x0, lp; во-вторых, декремент счетчика цикла addi x20, x20,

-1; в-третьих, смещение указателя на начало массива: addi x1, x1, elem_sz*enroll) повторяются 4 раза.

Если же уменьшить количество циклов до двух (за счет увеличения количества обрабатываемых элементов массива за одну итерацию до 4), то эти команды выполнятся всего 2 раза. При правильной и продуманной последовательности операций обращения к памяти и выполнения арифметических вычисслений, все еще удастся избежать конфликтов по регистрам.

В итоге можно обеспечить выигрыш в 3*2=6 тактов, и оптимизированная программа будет работать на 6/49=13% быстрее.

Оптимизированная программа

Исходный текст оптимизированной программы представлен на листинге 7.

Листинг 7 – Исходный текст оптимизированной программы

```
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    addi x20, x20, -1
14    add x31, x31, x2
15    add x31, x31, x3
16
17    lw x2, 8(x1)
18    lw x3, 12(x1)
19    addi x1, x1, elem_sz*enroll
20    add x31, x31, x2
21    add x31, x31, x3
22
23    bne x20, x0, lp
```

```
24      addi x31, x31, 1
25      lp2: j lp2
26
27      .section .data
28      _x:     .4byte 0x1
29      .4byte 0x2
30      .4byte 0x3
31      .4byte 0x4
32      .4byte 0x5
33      .4byte 0x6
34      .4byte 0x7
35      .4byte 0x8
```

Дизассемблерный листинг оптимизированной программы представлен на листинге 8.

Листинг 8 – Дизассемблерный листинг оптимизированной программы

```
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:       0040a183                lw      x3,4(x1)
80000014:       fffa0a13                addi    x20,x20,-1
80000018:       002f8fb3                add     x31,x31,x2
8000001c:       003f8fb3                add     x31,x31,x3
80000020:       0080a103                lw      x2,8(x1)
80000024:       00c0a183                lw      x3,12(x1)
80000028:       01008093                addi    x1,x1,16
8000002c:       002f8fb3                add     x31,x31,x2
80000030:       003f8fb3                add     x31,x31,x3
80000034:       fc0a1ce3                bne     x20,x0,8000000c <lp>
80000038:       001f8f93                addi    x31,x31,1

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

Трасса работы программы

Трасса работы оптимизированной программы представлена на рисунке 6.

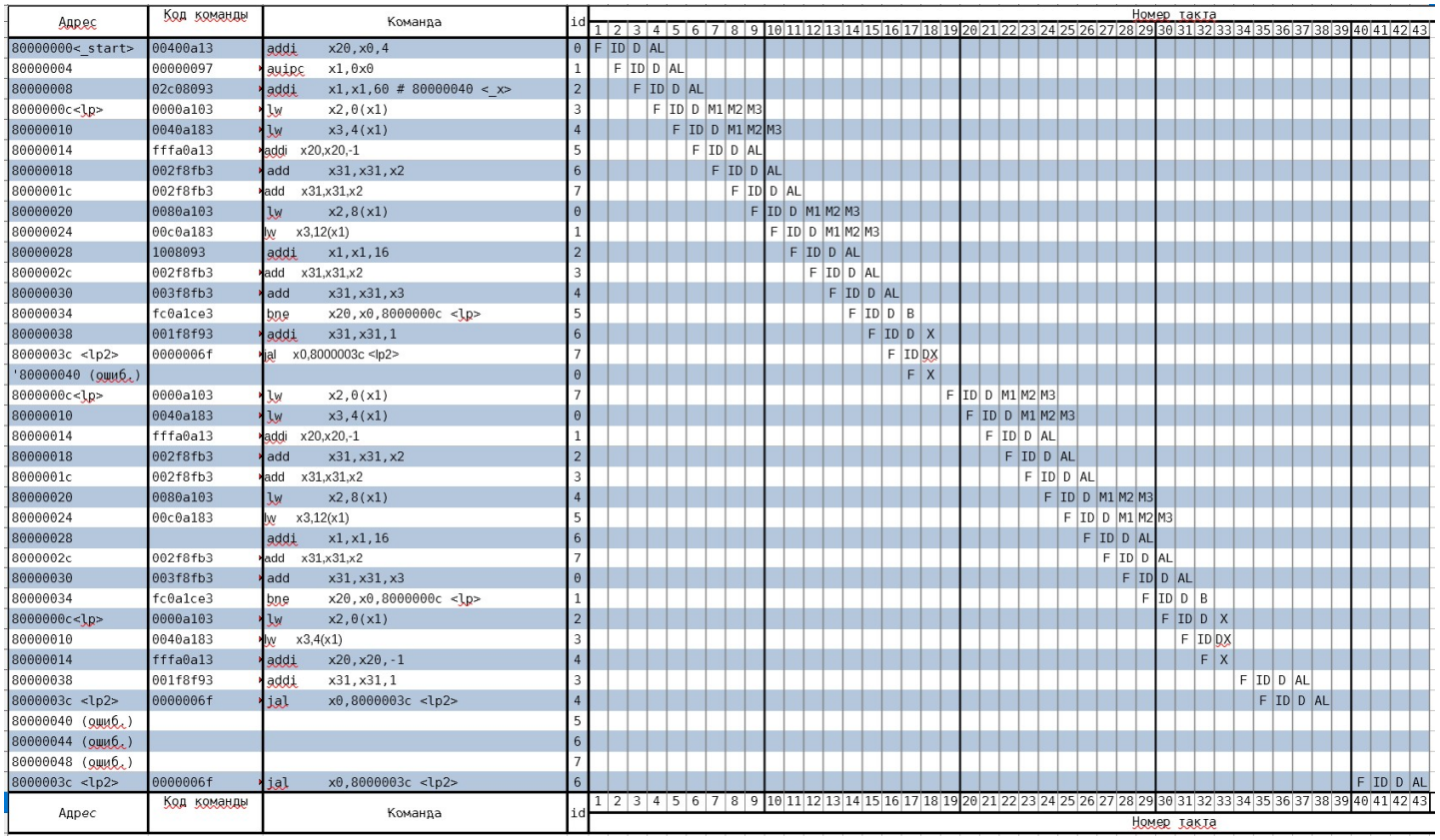

Рисунок 6 – Трасса работы оптимизированной программы

Вывод

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.