

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 N 2

Название	Изучение принципов работы микропр	роцессорного ядра	a RISC-V	
Дисципли	на Архитектура элекронно-вычислите	ельных машин		
Студент:			Козлова И. В.	
	ATTO TIL	подпись, дата	Фамилия, И.О. Попов А. Ю.	
Преподава	vielis.	подпись, дата	Фамилия, И. О.	

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. 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.

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.

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.

Общая программа для всех вариантов

Исследуемая программа

Код программы представлен в листинте 1

Листинг 1 – Код программы для всех вариантов

```
.section .text
 1
 2
        . globl _start;
 3
        len = 8
        enroll = 4
 4
 5
        elem sz = 4
 6
   _start:
 7
        addi x20, x0, len/enroll
 8
        la x1, _x
9 loop:
10
        lw x2, 0(x1)
        add x31, x31, x2
11
        lw \times 2, 4(\times 1)
12
        add x31, x31, x2
13
14
        Iw x^2, 8(x^1)
15
        add x31, x31, x2
        lw x2, 12(x1)
16
        add x31, x31, x2
17
        addi x1, x1, elem sz*enroll
18
        addi \times 20, \times 20, -1
19
20
        bne x20, x0, loop
21
        addi x31, x31, 1
22 forever: j forever
23
24
        .section .data
25
   x: .4 byte 0x1
26
        .4 byte 0x2
27
        .4 \, \text{byte} \, 0 \, \text{x}3
28
        .4 byte 0x4
29
        .4 \, \text{byte} \, 0 \times 5
30
        .4 byte 0x6
        .4 byte 0x7
31
32
        .4 byte 0x8
```

Дизассемблерный код представлен на листинге 2.

Листинг 2 – Дизассемблированный код общей программы

1	Disassembly of s			под оощен программы
2				
3	80000000 <_start	:>:		
4	80000000:	00200a13	addi	x20,x0,2
5	80000004:	00000097	auipc	×1 , 0 ×0
6	80000008:	03c08093	addi	x1,x1,60 # 80000040 <_x>
7				
8	8000000c <lp>:</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	×4,8(×1)
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></lp>
20	80000038:	001f8f93	addi	×31,×31,1
21				
22	8000003c < lp2 >:			
23	8000003c:	0000006 f	jal	x0,8000003c < lp2>

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

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

Результаты исследования программы

Задания выполнялись по вараинту 9.

Задание №2

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

Результат представлен на рисунке 1

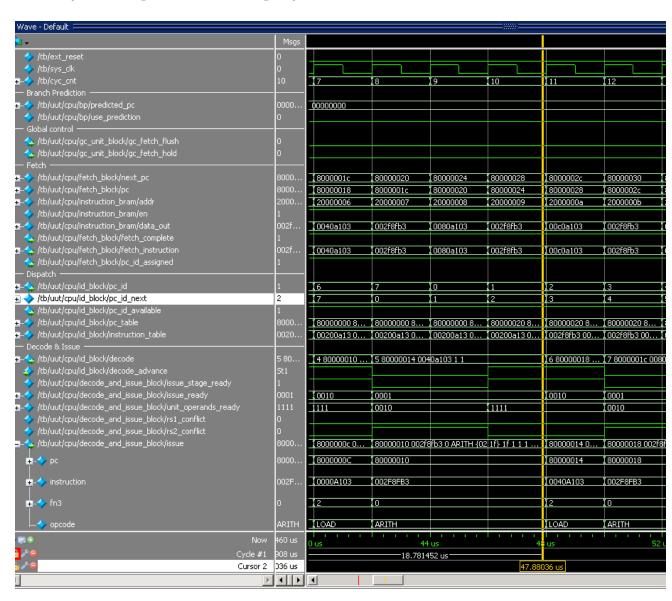


Рисунок 1 – Задание №2

Задание №3

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

Результат представлен на рисунке 2

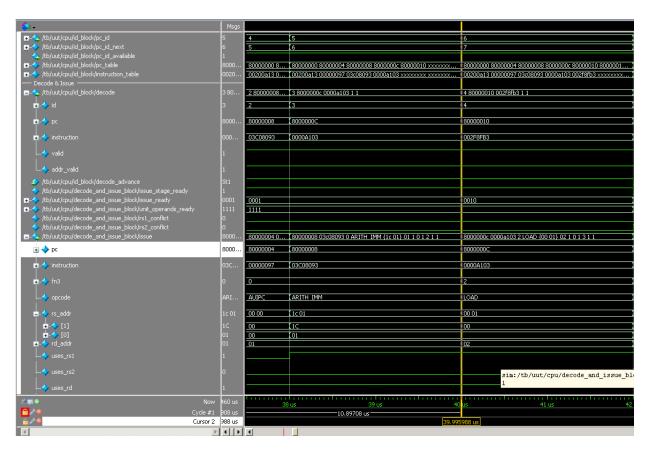


Рисунок 2 – Задание №3

Задание №4

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

Результат представлен на рисунке 3

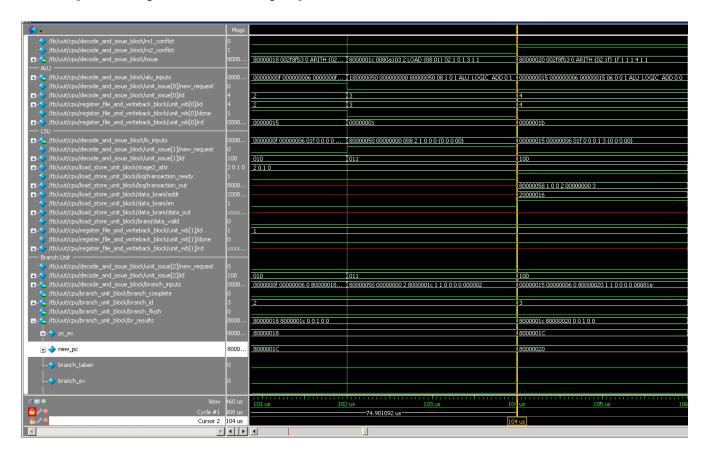


Рисунок 3 – Задание №4

Программа по варианту

Исследуемная программа

Код программы представлен в листинте 4

Листинг 4 – Код программы 9 варианта

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

Дизассемблерный код представлен на листинге 5.

Листинг 5 — Дизассемблированный код 9 варианта

_ [у дизассемолир		
1	Disassembly of	section .text:		
2				
3	80000000 <_sta	rt >:		
4	8000000:	00200a13	addi	x20,x0,2
5	8000004:	00000097	auipc	×1 , 0 ×0
6	80000008:	03c08093	addi	×1,×1,60 # 8000040
	<_x>			
7				
8	8000000c <lp>:</lp>			
9	8000000c:	0000a103	lw	x2,0(x1)
10	80000010:	0040a183	add	x31 , x31 , x2
11	80000014:	0080a203	lw	×4,8(×1)
12	80000018:	00c0a283	add	x31 , x31 , x3
13	8000001c:	002f8fb3	lw	x3,4(x1)
14	80000020:	003f8fb3	add	x31 , x31 , x4
15	80000024:	004 f8 fb 3	lw	x5,12(x1)
16	80000028:	005 f8fb3	add	x31 , x31 , x5
17	8000002c:	01008093	addi	×1 , ×1 , 16
18	80000030:	fffa0a13	addi	×20 , ×20 , —1
19	80000034:	fc0a1ce3	bne	x20,x0,8000000c <lp></lp>
20	80000038:	001f8f93	addi	x31,x31,1
21				
22	8000003c < lp2 >	:		
23	8000003c:	0000006 f	jal	x0,8000003c <lp2></lp2>

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

Листинг 6 – Псевдокод программы 9 варианта

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

Трасса работы программы

Трасса работы представлена на рисунке 7.

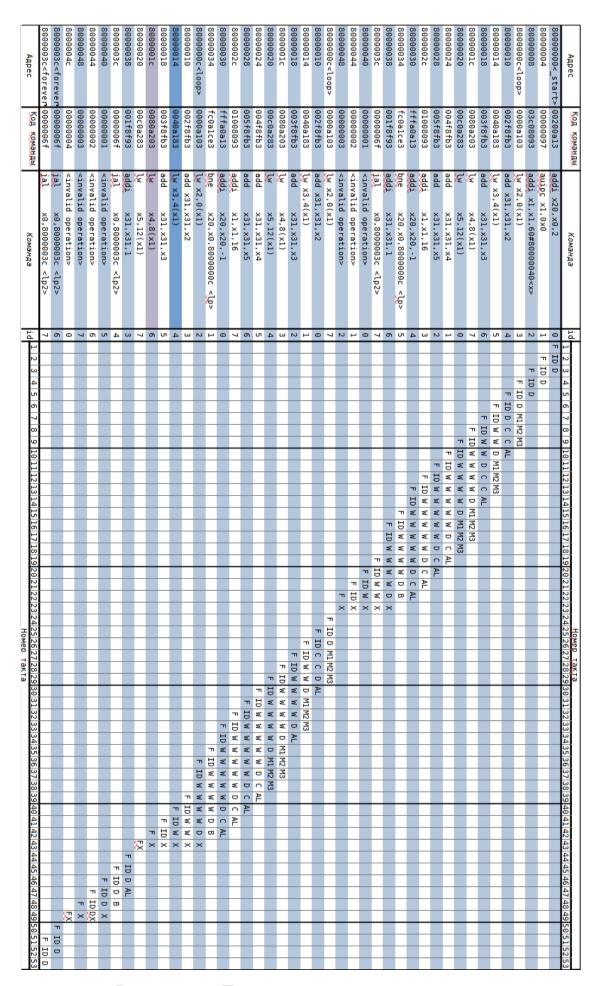


Рисунок 4 – Трасса выполнения программы

Временные диаграммы

Временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом #! (add x31, x31, x2) представлены на рисунке 5.

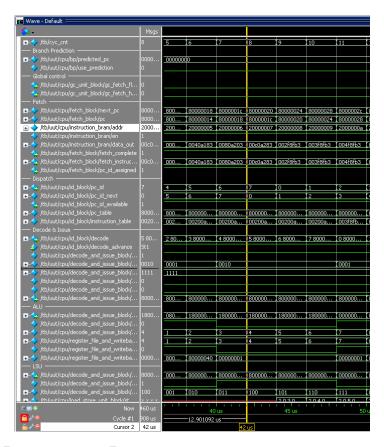


Рисунок 5 – Временные диаграммы сигналов

Вывод и предложение по оптимизации

Как видно на трассе работы программы, представленой на рисунке 7, конфликты возникают из-за того, что данные загружаются в память тогда, когда уже готова выполниться операция сложения тех данных, которые загружаются. Из-за этого и возникают конфликты, так как нечего складывать, так как в памяти пока нет ничего.

Можно заметить, что трижды возникает ситуация ошибочной выборки, которая негативно сказывается на производительности, так как приводит к очистки конвейера.

Оптимизировать программы можно тем, что сначала загрузить все данные в память, а потом их складывать. Тем самым у нас не будет конфликтов, не будет ожидания конца загрузки данных в память.

В итоге, можно будет уменьшить программу на 2 такта 4 раза в программе, на 1 такт 3 раза в программе, то есть на 11/53=20% программа будет работать быстрее.

Оптимизированная программа

Код программы представлен в листинте 7

Листинг 7 – Код программы 9 варианта (оптимизированный)

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

Дизассемблерный код представлен на листинге 8.

Листинг 8 — Дизассемблированный код 9 варинта (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:	005 f8 fb 3	add	x31 , x31 , x5
17	8000002c:	01008093	addi	×1,×1,16
18	80000030:	fffa0a13	addi	×20,×20,—1
19	80000034:	fc0a1ce3	bne	x20,x0,8000000c <1p>
20	80000038:	001f8f93	addi	x31,x31,1
21				
22	8000003c < lp2 >:			
23	8000003c:	0000006 f	jal	x0,8000003c <lp2></lp2>

Можно сказать, что данная программа эквивалентна следующему псевдокоду на языке С, представленному на листинге 9.

Листинг 9 – Псевдокод программы 9 варинта (оптимизированный)

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

Трасса работы программы

Трасса работы представлена на рисунке 8.

	18 19 29 21 22 23 24 25 26 27 28 29 38 31 32 33 34 35 36 37 38 39 49 41 42	2 13 14 15 16 17	7 8 9 10 11 12	2 3 4 5 6 7	1 1	Коменов	Код команды	Δπρες
F ID D					6	jal x0,8000003c <lp2></lp2>	0000006f	8000003c
					0	<invalid operation=""></invalid>	00000004	8000004c
FX					7	<invalid operation=""></invalid>	00000003	80000048
F ID DX					6	<invalid operation=""></invalid>	00000002	80000044
F ID D X					u	<invalid operation=""></invalid>	10000000	80000040
F ID D B					4	jąl x0,8000003c <lp2></lp2>	0000006f	8000003c
F ID D AL					ω	gddi x31,x31,1	001f8f93	80000038
æ					5	J _W x5,12(x1)	00c0a283 ·	80000018
T X					4	Jw x4,8(x1)	0080a203	80000014
					ω	Jw x3,4(x1)	0040a183	80000010
F ID D X					2	J _W ×2,0(×1)	0000a103	<pre>8000000c<loop></loop></pre>
F ID D B					_	<u>ხე</u> ę x20,x0,8000000c < <u>ე</u> p>	fc0alce3	80000034
F ID D AL					0	addi x20,x20,-1	fffa0al3	80000030
F ID D AL					7	addi x1,x1,16	01008093	8000002c
F ID D AL					6	add x31,x31,x5	005f8fb3	80000028
ID D AL					5	add x31,x31,x4	004f8fb3	80000024
D D AL	F ID				4	add x31,x31,x3	003f8fb3	80000020
AL	F ID D AL				ω	add x31,x31,x2	002f8fb3	8000001c
1 M2 M3	F ID D M1 M2 M3				2	<u></u>	00c0a283	80000018
2 M3	F ID D M1 M2 M3				_	<u></u> w x4,8(x1)	0080a203	80000014
	F ID D M1 M2 M3				0	<u></u>	0040a183	80000010
	F ID D M1 M2 M3				7	J _W x2,0(x1)	0000a103	8000000c <loop></loop>
	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\				1	<invalid operation=""></invalid>	00000002	80000044
	×	T 1			0	<invalid operation=""></invalid>	10000000	80000040
	××	F ID DX			7	jal x0,8000003c <lp2></lp2>	0000006f	8000003c
	×	F ID D			6	addi x31,x31,1	001f8f93	80000038
		F ID D B			5			80000034
		F ID D AL			4			80000030
		ID D AL	- -		ω	addi x1,x1,16	01008093	8000002c
		D AL	F ID		2	add x31,x31,x5	005f8fb3	80000028
		AL	F ID D		_	add x31,x31,x4	004f8fb3	80000024
			F ID D AL		0		Ţ	80000020
			F ID D AL		7	d x31,x31,x2	002f8fb3	21000008
			F ID D M1 M2 M3		6	<u></u>	00c0a283	8100008
			ID D M1 M2 M3	F I	5	x4,8(x1)	0080a203	80000014
			D M1 M2 M3	F ID C	4	W x3,4(x1)	0040a183	80000010
			M1 M2 M3	F ID D M	ω	J _W x2,0(x1)	0000a103	<qool>>36666668</qool>
				F ID D AL	2	addi x1,x1,60#80000040 <x></x>	03c08093	80000008
				F ID D AL	_	guipc x1,0x0	00000097	80000004
				ID D AL	e F	2,0x,02x jppe	00200a13	<pre><tart></tart></pre>
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42	0 1/ 18 19 28 21 22 23 24 25		I CT 15T CT 7T TT NT 6 0 /	2 3 4 3 0 7	۲			

Рисунок 6 – Трасса выполнения 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.

Поставленная цель достигнута.

Приложение

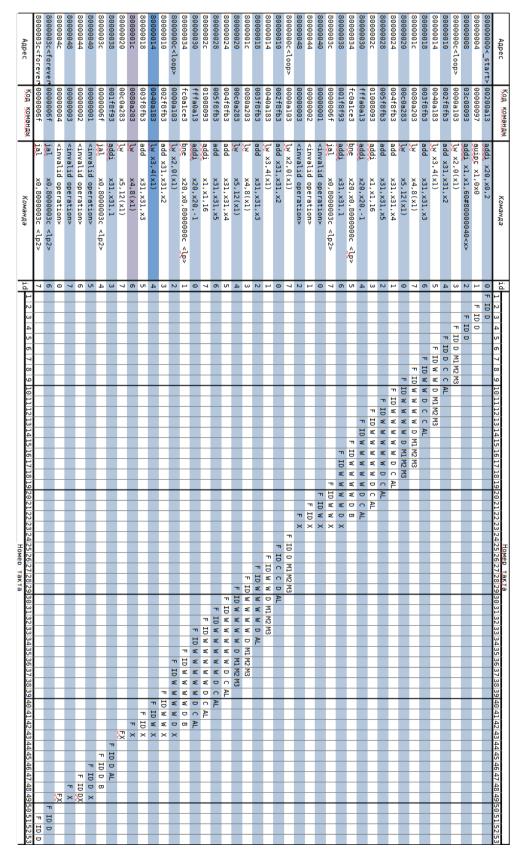


Рисунок 7 – Трасса выполнения программы

	18 19 29 21 22 23 24 25 26 27 28 29 38 31 32 33 34 35 36 37 38 39 49 41 42	2 13 14 15 16 17	7 8 9 10 11 12	2 3 4 5 6 7	1 1	Коменов	Код команды	Δπρες
F ID D					6	jal x0,8000003c <lp2></lp2>	0000006f	8000003c
					0	<invalid operation=""></invalid>	00000004	8000004c
FX					7	<invalid operation=""></invalid>	00000003	80000048
F ID DX					6	<invalid operation=""></invalid>	00000002	80000044
F ID D X					u	<invalid operation=""></invalid>	10000000	80000040
F ID D B					4	jąl x0,8000003c <lp2></lp2>	0000006f	8000003c
F ID D AL					ω	gddi x31,x31,1	001f8f93	80000038
æ					5	J _W x5,12(x1)	00c0a283 ·	80000018
T X					4	Jw x4,8(x1)	0080a203	80000014
					ω	Jw x3,4(x1)	0040a183	80000010
F ID D X					2	J _W ×2,0(×1)	0000a103	<pre>8000000c<loop></loop></pre>
F ID D B					_	<u>ხე</u> ę x20,x0,8000000c < <u>ე</u> p>	fc0alce3	80000034
F ID D AL					0	addi x20,x20,-1	fffa0al3	80000030
F ID D AL					7	addi x1,x1,16	01008093	8000002c
F ID D AL					6	add x31,x31,x5	005f8fb3	80000028
ID D AL					5	add x31,x31,x4	004f8fb3	80000024
D D AL	F ID				4	add x31,x31,x3	003f8fb3	80000020
AL	F ID D AL				ω	add x31,x31,x2	002f8fb3	8000001c
1 M2 M3	F ID D M1 M2 M3				2	<u></u>	00c0a283	80000018
2 M3	F ID D M1 M2 M3				_	<u></u> w x4,8(x1)	0080a203	80000014
	F ID D M1 M2 M3				0	<u></u>	0040a183	80000010
	F ID D M1 M2 M3				7	J _W x2,0(x1)	0000a103	8000000c <loop></loop>
	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\				1	<invalid operation=""></invalid>	00000002	80000044
	×	T 1			0	<invalid operation=""></invalid>	10000000	80000040
	××	F ID DX			7	jal x0,8000003c <lp2></lp2>	0000006f	8000003c
	×	F ID D			6	addi x31,x31,1	001f8f93	80000038
		F ID D B			5			80000034
		F ID D AL			4			80000030
		ID D AL	- -		ω	addi x1,x1,16	01008093	8000002c
		D AL	F ID		2	add x31,x31,x5	005f8fb3	80000028
		AL	F ID D		_	add x31,x31,x4	004f8fb3	80000024
			F ID D AL		0		Ţ	80000020
			F ID D AL		7	d x31,x31,x2	002f8fb3	21000008
			F ID D M1 M2 M3		6	<u></u>	00c0a283	8100008
			ID D M1 M2 M3	F I	5	x4,8(x1)	0080a203	80000014
			D M1 M2 M3	F ID C	4	W x3,4(x1)	0040a183	80000010
			M1 M2 M3	F ID D M	ω	J _W x2,0(x1)	0000a103	<qool>>36666668</qool>
				F ID D AL	2	addi x1,x1,60#80000040 <x></x>	03c08093	80000008
				F ID D AL	_	guipc x1,0x0	00000097	80000004
				ID D AL	e F	2,0x,02x jppe	00200a13	<pre><tart></tart></pre>
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42	0 1/ 18 19 28 21 22 23 24 25		I CT 15T CT 7T TT NT 6 0 /	2 3 4 3 0 7	۲			

Рисунок 8 – Трасса выполнения программы