



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 № 1

Название Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина Архитектура электронно-вычислительных машин

Студент:

_____ Фам М. Х.
подпись, дата Фамилия, И.О.

Преподаватель:

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

Москва — 2023 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. 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 листинге 1

Листинг 1 – Код программы для всех вариантов

```
1      .section .text
2      .globl _start;
3      len = 8
4      enroll = 4
5      elem_sz = 4
6 _start:
7      addi x20, x0, len/enroll
8      la x1, _x
9 loop:
10     lw x2, 0(x1)
11     add x31, x31, x2
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
19     addi x20, x20, -1
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 byte 0x3
28     .4 byte 0x4
29     .4 byte 0x5
30     .4 byte 0x6
31     .4 byte 0x7
32     .4 byte 0x8
```

Дизассемблерный код представлен на листинге 2.

Листинг 2 – Дизассемблированный код общей 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 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() {
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 }
```

Результаты исследования программы

Задания выполнялись по варианту 20.

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

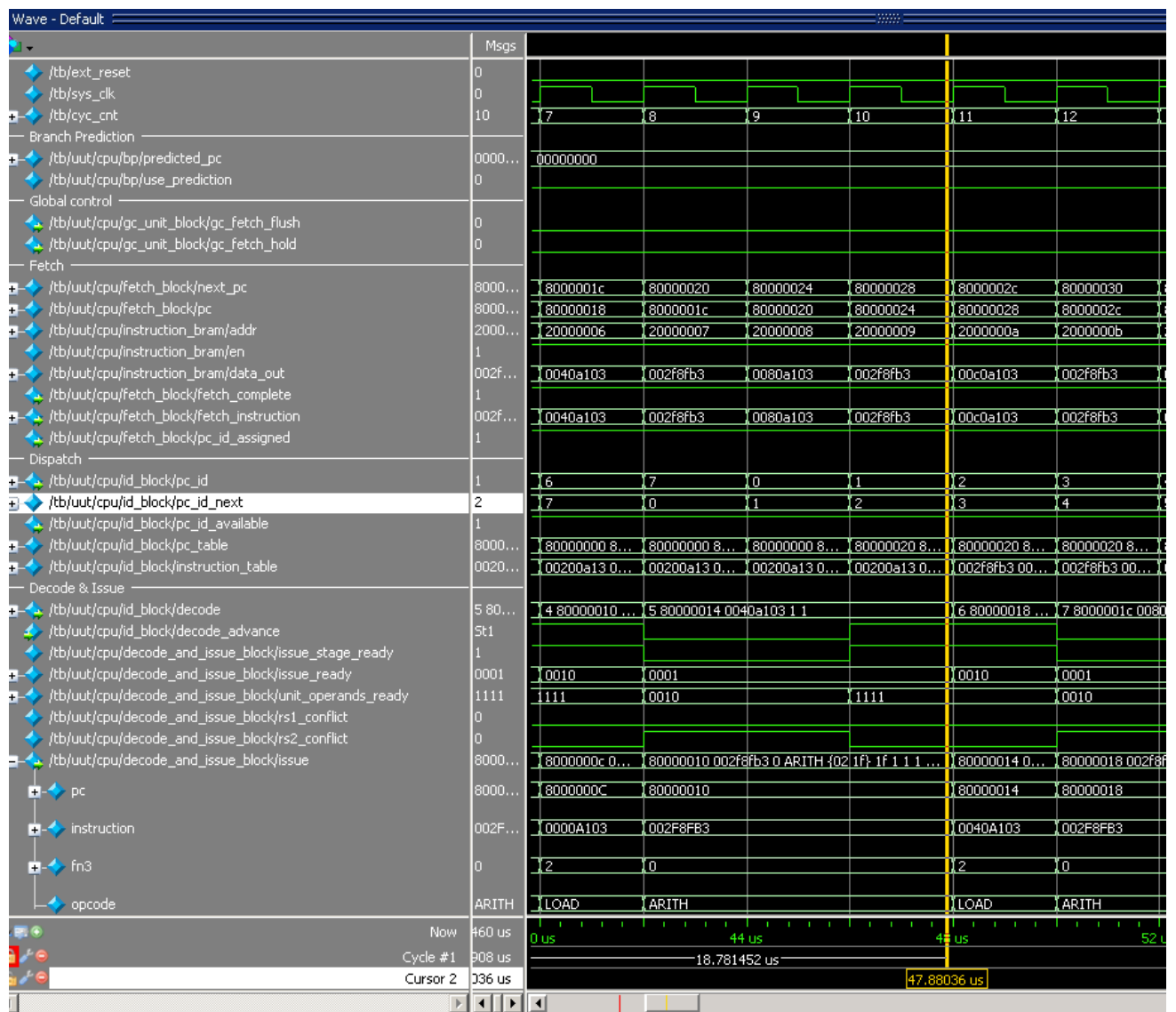


Рисунок 1 – Задание №2

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

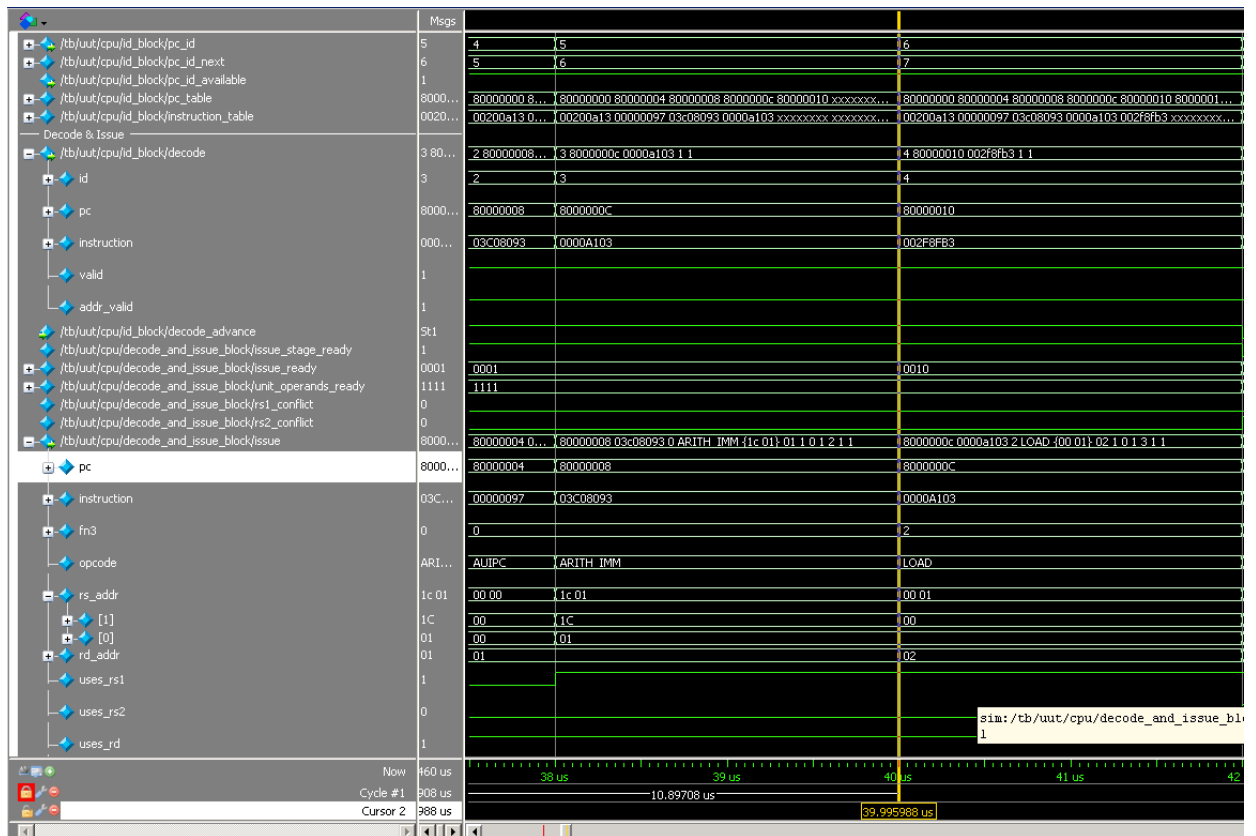


Рисунок 2 – Задание №3

Задание №4

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

Результат представлен на рисунке 3

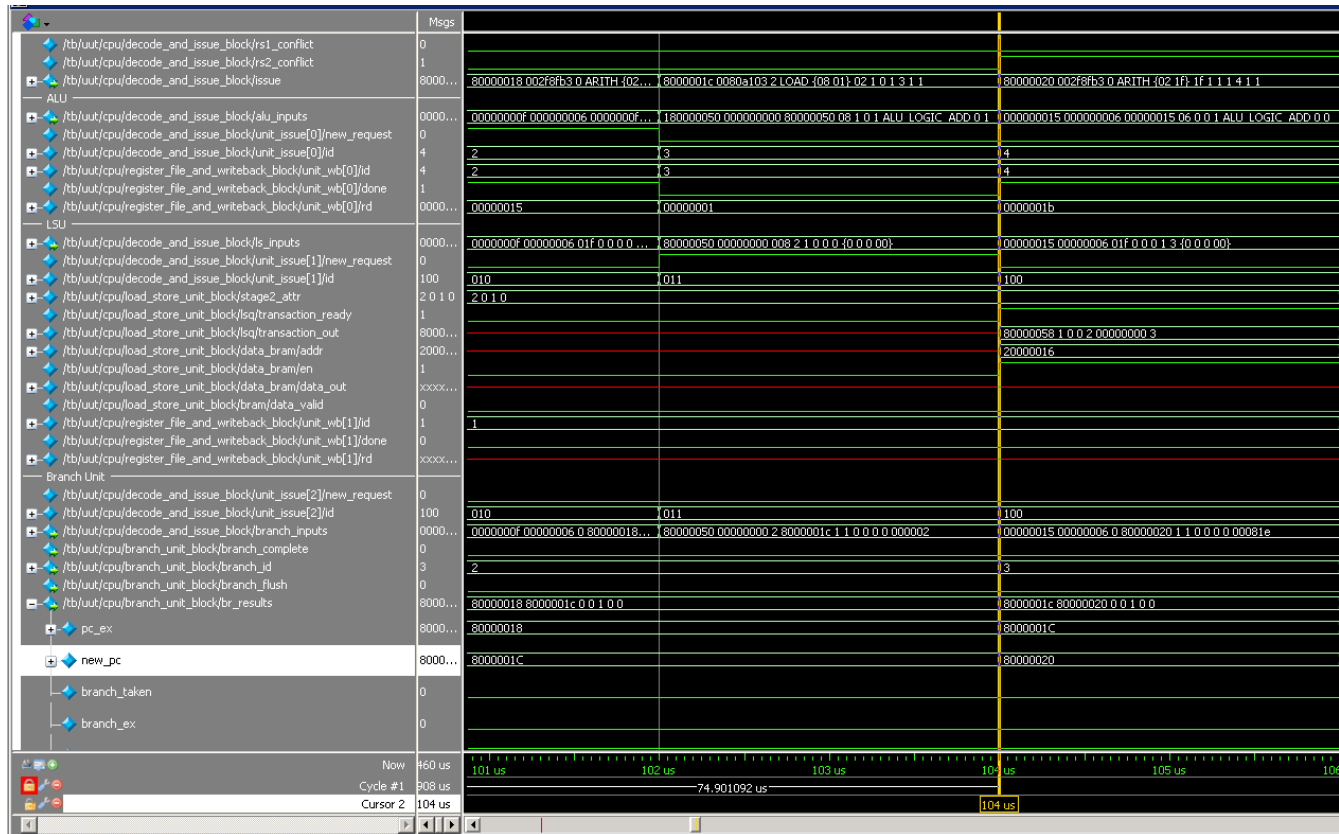


Рисунок 3 – Задание №4

Программа по варианту

Исследуемая программа

Код программы представлен в листинге 4

Листинг 4 – Код программы 20 варианта

```
1      .section .text
2      .globl _start;
3      len = 9
4      enroll = 4
5      elem_sz = 4
6
7      _start:
8          la x1, _x
9          addi x20, x0, (len-1)/enroll
10         lw x31, 0(x1)
11         addi x1, x1, elem_sz*1
12 lp:
13         lw x2, 0(x1)
14         lw x3, 4(x1)
15         add x1, x1, elem_sz*enroll
16         addi x20, x20, -1 #!
17         bltu x2, x31, lt1
18         add x31, x0, x2
19 lt1:    bltu x3, x31, lt2
20         add x31, x0, x3
21 lt2:    lw x4, 8(x1)
22         lw x5, 12(x1)
23         bltu x4, x31, lt3
24         add x31, x0, x4
25 lt3:    bltu x5, x31, lt4
26         add x31, x0, x5
27 lt4:
28         bne x20, x0, lp
29 lp2: j lp2
30
31         .section .data
32         _x:      .4byte 0x1
33         .4byte 0x2
```

```

34      .4 byte 0x3
35      .4 byte 0x4
36      .4 byte 0x5
37      .4 byte 0x6
38      .4 byte 0x5
39      .4 byte 0x8
40      .4 byte 0x9

```

Дизассемблерный код представлен на листинге 5.

Листинг 5 – Дизассемблированный код 20 варианта

```

1      Disassembly of section .text:
2
3      80000000 <_start>:
4      80000000: 00000097      auipc x1,0x0
5      80000004: 05408093      addi x1,x1,84 # 80000054 <_x>
6      80000008: 00200a13      addi x20,x0,2
7      8000000c: 0000af83      lw x31,0(x1)
8      80000010: 00408093      addi x1,x1,4
9
10     80000014 <lp>:
11     80000014: 0000a103      lw x2,0(x1)
12     80000018: 0040a183      lw x3,4(x1)
13     8000001c: fffa0a13      addi x20,x20,-1
14     80000020: 01f16463      bltu x2,x31,80000028 <lt1>
15     80000024: 00200fb3      add x31,x0,x2
16
17     80000028 <lt1>:
18     80000028: 01f1e463      bltu x3,x31,80000030 <lt2>
19     8000002c: 00300fb3      add x31,x0,x3
20
21     80000030 <lt2>:
22     80000030: 0080a203      lw x4,8(x1)
23     80000034: 00c0a283      lw x5,12(x1)
24     80000038: 01f26463      bltu x4,x31,80000040 <lt3>
25     8000003c: 00400fb3      add x31,x0,x4
26
27     80000040 <lt3>:
28     80000040: 01f2e463      bltu x5,x31,80000048 <lt4>
29     80000044: 00500fb3      add x31,x0,x5
30
31     80000048 <lt4>:

```

32	80000048: 01008093	addi x1,x1,16
33	8000004c: fc0a14e3	bne x20,x0,80000014 <lp>
34		
35	80000050 <lp2>:	
36	80000050: 0000006f	jal x0,80000050 <lp2>

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

Листинг 6 – Псевдокод программы 20 варианта

```
1  #define len 8
2  #define enroll 4
3  #define elem_sz 4
4  int _x[]={1,2,3,4,5,6,5,8,9};
5  void _start() {
6      int *x1 = _x;
7      int x20 = (len - 1) / enroll;
8      int x31 = x1[0];
9      x1 += 1;
10
11     do {
12         int x2 = x1[0];
13         int x3 = x1[1];
14         x1 += enroll;
15         x20--;
16         if (x2 > x31)
17             x31 = x2;
18         if (x3 > x31)
19             x31 = x3;
20         int x4 = x1[2];
21         int x5 = x1[3];
22         if (x4 > x31)
23             x31 = x4;
24         if (x5 > x31)
25             x31 = x5;
26     } while(x20 != 0);
27     while(1){}
28 }
```

Трасса работы программы

Трасса работы представлена на рисунке 7.

Адрес	Код команды	Команда	Ид	Номер такта
80000000<start>	00000097	ajpc x1,0,0	0	1
80000004	05400093	add x1,x1,04 # 80000004 <_x>	1	2
80000008	02000113	add x20,x2,2	2	3
8000000c	000a083	lw x31,0(x1)	3	4
80000010	00400093	add x1,x1,4	4	5
80000014<ip>	000a103	lw x2,0(x1)	5	6
80000018	00401183	lw x3,4(x1)	6	7
8000001c	#f0a13	add x20,x20,-1	7	8
80000020	0116483	blu x2,x31,80000002c<lit>	8	9
80000024	00200b3	add x31,x0,x2	9	10
80000028	0116483	blu x3,x31,800000034<i2>	10	11
8000002c<lit>	00300b3	add x31,x0,x3	11	12
80000030	0080a203	lw x4,8(x1)	12	13
80000034<i2>	00ca283	lw x5,12(x1)	13	14
80000038	0128483	blu x4,x31,800000044<i3>	14	15
8000003c	00400b3	add x31,x0,x4	15	16
80000040	0128483	blu x5,x31,80000004c<i4>	16	17
80000044<i3>	00500b3	add x31,x0,x5	17	18
80000048	01000093	add x1,x1,16	18	19
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	19	20
80000050<ip2>	0000009f	jal x0,800000000<ip2>	20	21
80000054	00000001	<invalid operation>	21	22
80000058	00000002	<invalid operation>	22	23
8000005c	00000003	<invalid operation>	23	24
80000060	00000004	<invalid operation>	24	25
80000014<ip>	000a103	lw x2,0(x1)	25	26
80000018	00401183	lw x3,4(x1)	26	27
8000001c	#f0a13	add x20,x20,-1	27	28
80000020	0116483	blu x2,x31,80000002c<lit>	28	29
80000024	00200b3	add x31,x0,x2	29	30
8000002c<lit>	00300b3	add x31,x0,x3	30	31
80000030	0080a203	lw x4,8(x1)	31	32
80000034<i2>	00ca283	lw x5,12(x1)	32	33
80000038	0128483	blu x4,x31,800000044<i3>	33	34
8000003c	00500b3	add x31,x0,x5	34	35
80000044<i3>	00500b3	add x31,x0,x5	35	36
80000048	01000093	add x1,x1,16	36	37
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	37	38
80000014<ip>	000a103	lw x2,0(x1)	38	39
80000018	00401183	lw x3,4(x1)	39	40
8000001c	#f0a13	add x20,x20,-1	40	41
80000020	0116483	blu x2,x31,80000002c<lit>	41	42
80000024	00200b3	add x31,x0,x2	42	43
8000002c<lit>	00300b3	add x31,x0,x3	43	44
80000030	0080a203	lw x4,8(x1)	44	45
80000034<i2>	00ca283	lw x5,12(x1)	45	46
80000038	0128483	blu x4,x31,800000044<i3>	46	47
8000003c	00500b3	add x31,x0,x5	47	48
80000044<i3>	00500b3	add x31,x0,x5	48	49
80000048	01000093	add x1,x1,16	49	50
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	50	51
80000014<ip>	000a103	lw x2,0(x1)	51	52
80000018	00401183	lw x3,4(x1)	52	53
8000001c	#f0a13	add x20,x20,-1	53	54
80000020	0116483	blu x2,x31,80000002c<lit>	54	55
80000024	00200b3	add x31,x0,x2	55	56
8000002c<lit>	00300b3	add x31,x0,x3	56	57
80000030	0080a203	lw x4,8(x1)	57	58
80000034<i2>	00ca283	lw x5,12(x1)	58	59
80000038	0128483	blu x4,x31,800000044<i3>	59	60
8000003c	00500b3	add x31,x0,x5	60	61
80000044<i3>	00500b3	add x31,x0,x5	61	62
80000048	01000093	add x1,x1,16	62	63
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	63	64
80000014<ip>	000a103	lw x2,0(x1)	64	65
80000018	00401183	lw x3,4(x1)	65	66
8000001c	#f0a13	add x20,x20,-1	66	67
80000020	0116483	blu x2,x31,80000002c<lit>	67	68
80000024	00200b3	add x31,x0,x2	68	69
8000002c<lit>	00300b3	add x31,x0,x3	69	70
80000030	0080a203	lw x4,8(x1)	70	71
80000034<i2>	00ca283	lw x5,12(x1)	71	72
80000038	0128483	blu x4,x31,800000044<i3>	72	73
8000003c	00500b3	add x31,x0,x5	73	74
80000044<i3>	00500b3	add x31,x0,x5	74	75
80000048	01000093	add x1,x1,16	75	76
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	76	77
80000014<ip>	000a103	lw x2,0(x1)	77	78
80000018	00401183	lw x3,4(x1)	78	79
8000001c	#f0a13	add x20,x20,-1	79	80
80000020	0116483	blu x2,x31,80000002c<lit>	80	81
80000024	00200b3	add x31,x0,x2	81	82
8000002c<lit>	00300b3	add x31,x0,x3	82	83
80000030	0080a203	lw x4,8(x1)	83	84
80000034<i2>	00ca283	lw x5,12(x1)	84	85
80000038	0128483	blu x4,x31,800000044<i3>	85	86
8000003c	00500b3	add x31,x0,x5	86	87
80000044<i3>	00500b3	add x31,x0,x5	87	88
80000048	01000093	add x1,x1,16	88	89
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	89	90
80000014<ip>	000a103	lw x2,0(x1)	90	91
80000018	00401183	lw x3,4(x1)	91	92
8000001c	#f0a13	add x20,x20,-1	92	93
80000020	0116483	blu x2,x31,80000002c<lit>	93	94
80000024	00200b3	add x31,x0,x2	94	95
8000002c<lit>	00300b3	add x31,x0,x3	95	96
80000030	0080a203	lw x4,8(x1)	96	97
80000034<i2>	00ca283	lw x5,12(x1)	97	98
80000038	0128483	blu x4,x31,800000044<i3>	98	99
8000003c	00500b3	add x31,x0,x5	99	100
80000044<i3>	00500b3	add x31,x0,x5	100	101
80000048	01000093	add x1,x1,16	101	102
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	102	103
80000014<ip>	000a103	lw x2,0(x1)	103	104
80000018	00401183	lw x3,4(x1)	104	105
8000001c	#f0a13	add x20,x20,-1	105	106
80000020	0116483	blu x2,x31,80000002c<lit>	106	107
80000024	00200b3	add x31,x0,x2	107	108
8000002c<lit>	00300b3	add x31,x0,x3	108	109
80000030	0080a203	lw x4,8(x1)	109	110
80000034<i2>	00ca283	lw x5,12(x1)	110	111
80000038	0128483	blu x4,x31,800000044<i3>	111	112
8000003c	00500b3	add x31,x0,x5	112	113
80000044<i3>	00500b3	add x31,x0,x5	113	114
80000048	01000093	add x1,x1,16	114	115
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	115	116
80000014<ip>	000a103	lw x2,0(x1)	116	117
80000018	00401183	lw x3,4(x1)	117	118
8000001c	#f0a13	add x20,x20,-1	118	119
80000020	0116483	blu x2,x31,80000002c<lit>	119	120
80000024	00200b3	add x31,x0,x2	120	121
8000002c<lit>	00300b3	add x31,x0,x3	121	122
80000030	0080a203	lw x4,8(x1)	122	123
80000034<i2>	00ca283	lw x5,12(x1)	123	124
80000038	0128483	blu x4,x31,800000044<i3>	124	125
8000003c	00500b3	add x31,x0,x5	125	126
80000044<i3>	00500b3	add x31,x0,x5	126	127
80000048	01000093	add x1,x1,16	127	128
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	128	129
80000014<ip>	000a103	lw x2,0(x1)	129	130
80000018	00401183	lw x3,4(x1)	130	131
8000001c	#f0a13	add x20,x20,-1	131	132
80000020	0116483	blu x2,x31,80000002c<lit>	132	133
80000024	00200b3	add x31,x0,x2	133	134
8000002c<lit>	00300b3	add x31,x0,x3	134	135
80000030	0080a203	lw x4,8(x1)	135	136
80000034<i2>	00ca283	lw x5,12(x1)	136	137
80000038	0128483	blu x4,x31,800000044<i3>	137	138
8000003c	00500b3	add x31,x0,x5	138	139
80000044<i3>	00500b3	add x31,x0,x5	139	140
80000048	01000093	add x1,x1,16	140	141
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	141	142
80000014<ip>	000a103	lw x2,0(x1)	142	143
80000018	00401183	lw x3,4(x1)	143	144
8000001c	#f0a13	add x20,x20,-1	144	145
80000020	0116483	blu x2,x31,80000002c<lit>	145	146
80000024	00200b3	add x31,x0,x2	146	147
8000002c<lit>	00300b3	add x31,x0,x3	147	148
80000030	0080a203	lw x4,8(x1)	148	149
80000034<i2>	00ca283	lw x5,12(x1)	149	150
80000038	0128483	blu x4,x31,800000044<i3>	150	151
8000003c	00500b3	add x31,x0,x5	151	152
80000044<i3>	00500b3	add x31,x0,x5	152	153
80000048	01000093	add x1,x1,16	153	154
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	154	155
80000014<ip>	000a103	lw x2,0(x1)	155	156
80000018	00401183	lw x3,4(x1)	156	157
8000001c	#f0a13	add x20,x20,-1	157	158
80000020	0116483	blu x2,x31,80000002c<lit>	158	159
80000024	00200b3	add x31,x0,x2	159	160
8000002c<lit>	00300b3	add x31,x0,x3	160	161
80000030	0080a203	lw x4,8(x1)	161	162
80000034<i2>	00ca283	lw x5,12(x1)	162	163
80000038	0128483	blu x4,x31,800000044<i3>	163	164
8000003c	00500b3	add x31,x0,x5	164	165
80000044<i3>	00500b3	add x31,x0,x5	165	166
80000048	01000093	add x1,x1,16	166	167
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	167	168
80000014<ip>	000a103	lw x2,0(x1)	168	169
80000018	00401183	lw x3,4(x1)	169	170
8000001c	#f0a13	add x20,x20,-1	170	171
80000020	0116483	blu x2,x31,80000002c<lit>	171	172
80000024	00200b3	add x31,x0,x2	172	173
8000002c<lit>	00300b3	add x31,x0,x3	173	174
80000030	0080a203	lw x4,8(x1)	174	175
80000034<i2>	00ca283	lw x5,12(x1)	175	176
80000038	0128483	blu x4,x31,800000044<i3>	176	177
8000003c	00500b3	add x31,x0,x5	177	178
80000044<i3>	00500b3	add x31,x0,x5	178	179
80000048	01000093	add x1,x1,16	179	180
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	180	181
80000014<ip>	000a103	lw x2,0(x1)	181	182
80000018	00401183	lw x3,4(x1)	182	183
8000001c	#f0a13	add x20,x20,-1	183	184
80000020	0116483	blu x2,x31,80000002c<lit>	184	185
80000024	00200b3	add x31,x0,x2	185	186
8000002c<lit>	00300b3	add x31,x0,x3	186	187
80000030	0080a203	lw x4,8(x1)	187	188
80000034<i2>	00ca283	lw x5,12(x1)	188	189
80000038	0128483	blu x4,x31,800000044<i3>	189	190
8000003c	00500b3	add x31,x0,x5	190	191
80000044<i3>	00500b3	add x31,x0,x5	191	192
80000048	01000093	add x1,x1,16	192	193
8000004c<i4b>	fc0a1463	bne x20,x0,800000014<ip>	193	194
80000014<ip>	000a103			

Временные диаграммы

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

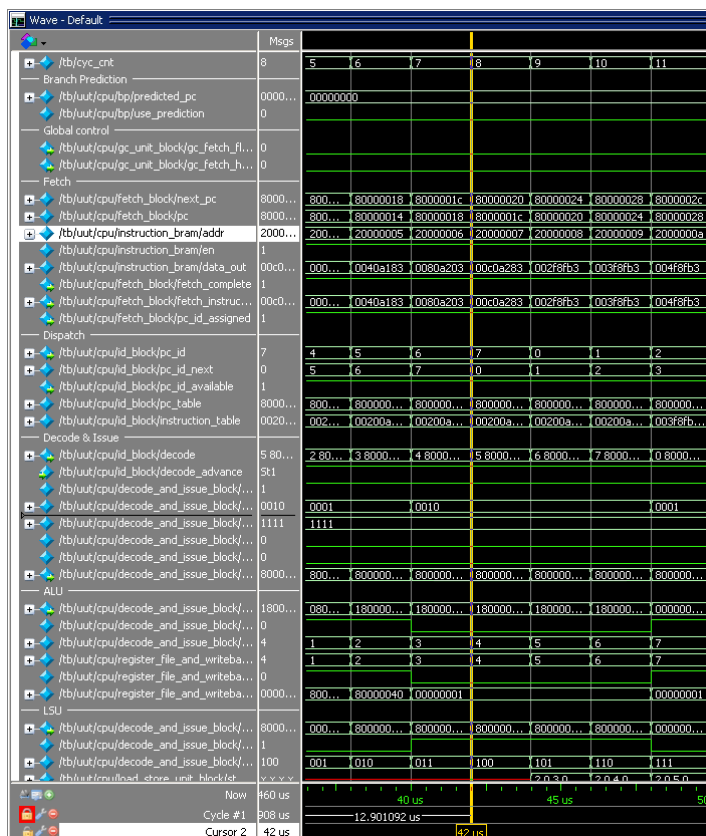


Рисунок 5 – Временные диаграммы сигналов

Вывод и предложение по оптимизации

Конфликт возникает из-за того, что команда загрузки памяти не завершил выполнение, а команда ветвления использовал `x4` в качестве параметра.

Оптимизировать программы можно тем, что вставить одну команду между командой загрузки памяти и командой ветвления.

В итоге, можно будет уменьшить программу на 2 такта в программе, то есть на $2/48 = 4\%$ программа будет работать быстрее.

Оптимизированная программа

Код программы представлен в листинге 7

Листинг 7 – Код программы 20 варианта(оптимизированный)

```
1      .section .text
2      .globl _start;
3      len = 9
4      enroll = 4
5      elem_sz = 4
6
7      _start:
8      la x1, _x
9      addi x20, x0, (len-1)/enroll
10     lw x31, 0(x1)
11     addi x1, x1, elem_sz*1
12     lp:
13     lw x2, 0(x1)
14     lw x3, 4(x1)
15     add x1, x1, elem_sz*enroll
16     addi x20, x20, -1 #!
17     bltu x2, x31, lt1
18     add x31, x0, x2
19     lt1:    bltu x3, x31, lt2
20     add x31, x0, x3
21     lt2:    lw x4, 8(x1)
22     lw x5, 12(x1)
23     bltu x4, x31, lt3
24     add x31, x0, x4
25     lt3:    bltu x5, x31, lt4
26     add x31, x0, x5
27     lt4:
28     bne x20, x0, lp
29     lp2:    j lp2
30
31     .section .data
32     _x:      .4 byte 0x1
33     .4 byte 0x2
34     .4 byte 0x3
35     .4 byte 0x4
36     .4 byte 0x5
```


37	.4 byte	0x6
38	.4 byte	0x5
39	.4 byte	0x8
40	.4 byte	0x9

Дизассемблерный код представлен на листинге 8.

Листинг 8 – Дизассемблированный код 20 варианта (оптимизированный)

```
1 80000000 <_start>:
2 80000000: 00000097          auipc x1,0x0
3 80000004: 05408093          addi x1,x1,84 # 80000054 <_x>
4 80000008: 00200a13          addi x20,x0,2
5 8000000c: 0000af83          lw x31,0(x1)
6 80000010: 00408093          addi x1,x1,4
7
8 80000014 <lp>:
9 80000014: 0000a103          lw x2,0(x1)
10 80000018: 0040a183          lw x3,4(x1)
11 8000001c: fffa0a13          addi x20,x20,-1
12 80000020: 01f16463          bltu x2,x31,80000028 <lt1>
13 80000024: 00200fb3          add x31,x0,x2
14
15 80000028 <lt1>:
16 80000028: 01f1e463          bltu x3,x31,80000030 <lt2>
17 8000002c: 00300fb3          add x31,x0,x3
18
19 80000030 <lt2>:
20 80000030: 0080a203          lw x4,8(x1)
21 80000034: 00c0a283          lw x5,12(x1)
22 80000038: 01008093          addi x1,x1,16
23 8000003c: 01f26463          bltu x4,x31,80000044 <lt3>
24 80000040: 00400fb3          add x31,x0,x4
25
26 80000044 <lt3>:
27 80000044: 01f2e463          bltu x5,x31,8000004c <lt4>
28 80000048: 00500fb3          add x31,x0,x5
29
30 8000004c <lt4>:
31 8000004c: fc0a14e3          bne x20,x0,80000014 <lp>
32
33 80000050 <lp2>:
34 80000050: 0000006f          jal x0,80000050 <lp2>
```

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

Листинг 9 – Псевдокод программы 20 варианта (оптимизированный)

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
4 int _x[]={1,2,3,4,5,6,5,8,9};
5 void _start() {
6     int *x1 = _x;
7     int x20 = (len - 1) / enroll;
8     int x31 = x1[0];
9     x1 += 1;
10
11     do {
12         int x2 = x1[0];
13         int x3 = x1[1];
14         x1 += enroll;
15         if (x2 > x31)
16             x31 = x2;
17         if (x3 > x31)
18             x31 = x3;
19         int x4 = x1[2];
20         int x5 = x1[3];
21         x20--;
22         if (x4 > x31)
23             x31 = x4;
24         if (x5 > x31)
25             x31 = x5;
26     } while(x20 != 0);
27     while(1){}
28 }
```

Трасса работы программы

Трасса работы представлена на рисунке 8.

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

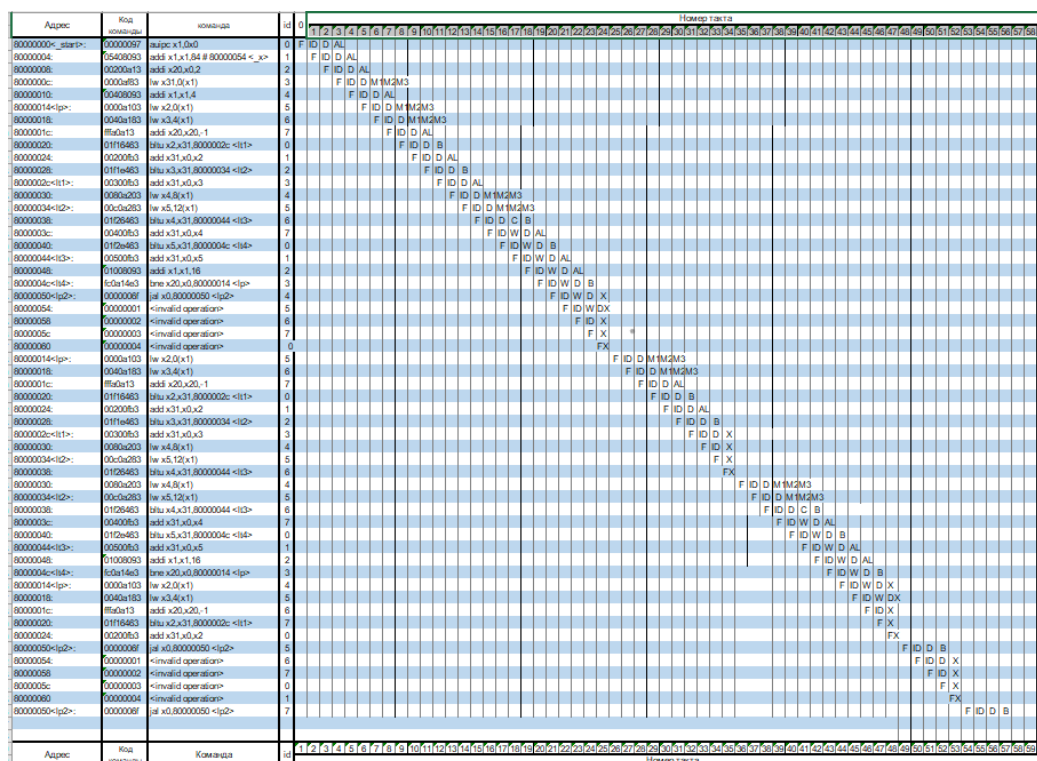


Рисунок 7 – Трасса выполнения программы

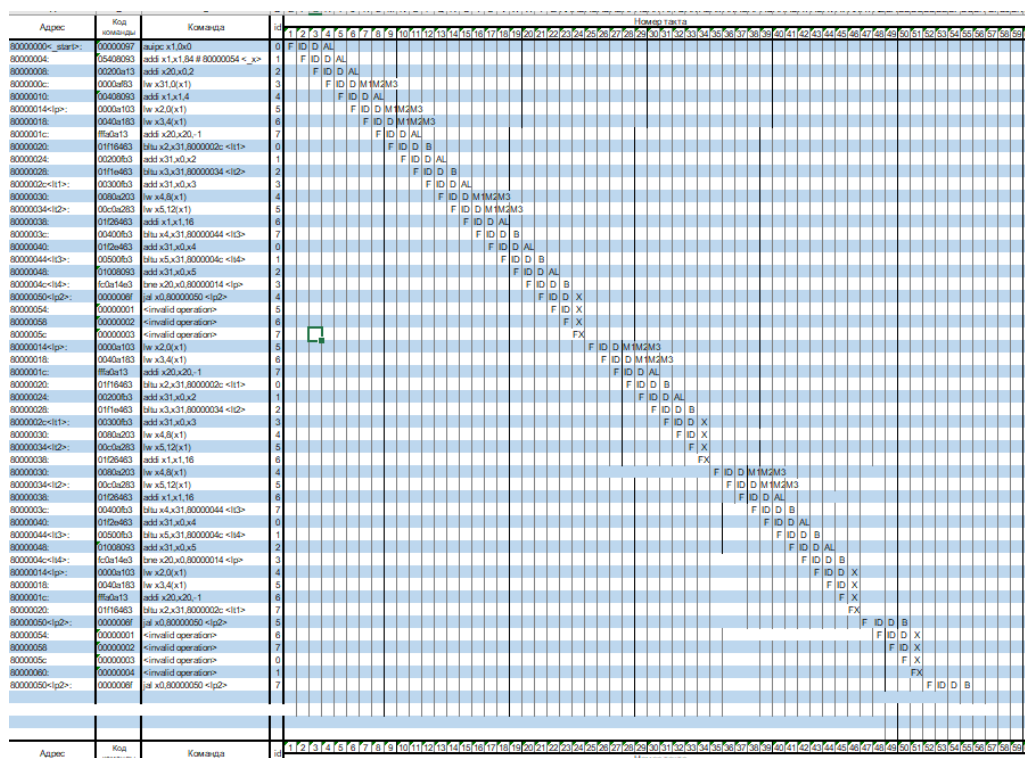


Рисунок 8 – Трасса выполнения программы