

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 си	стемы управл	ения»
КАФЕДРА		«Компьютерные сис	стемы и сети ((ИУ6)»
НАПРАВЛЕНИ	Е ПОДГОТОВКИ _	«09.03.04 Програ	ммная инжене	ерия»
O'.	гчет по л	АБОРАТОРНО	ОЙ РАБО	OTE №2
	по курс	су «Архитекту	ра ЭВМ)	»
		-		DICC V
«изучени	е принципов р	аботы микропро	оцессорног	о ядра RISC-v»
Студент:	<u>ИУ7-53Б</u>			М. Д. Маслова
	(группа)	(по	одпись, дата)	(И. О. Фамилия)
Преподавате	ель:			Е. Н. Дубровин

(подпись, дата)

(И. О. Фамилия)

Содержание

Bı	веден	ие	3
1	Осн	овные теоретические сведения	4
2	Зада	ание 1	5
	2.1	Результаты выполнения	5
	2.2	Вывод	7
3	Зада	ание 2	8
	3.1	Результаты выполнения	8
	3.2	Вывод	8
4	Зада	ание 3	9
	4.1	Результаты выполнения	9
	4.2	Вывод	9
5	Зада	ание 4	10
	5.1	Результаты выполнения	10
	5.2	Вывод	10
6	Зада	ание 5	11
	6.1	Проверка результата	11
	6.2	Временные диаграммы сигналов стадий выполнения	11
	6.3	Трасса выполнения программы	12
	6.4	Оптимизация программы	13
	6.5	Вывод	16
3 a	ключ	нение	18

Введение

Целью данной работы является 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.

1 Основные теоретические сведения

Для достижения поставленных целей в настоящей лабораторной работе используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V.

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.

Дизассемблирование программы по индивидуальному варианту.

2.1 Результаты выполнения

На листинге 2.1 приведен текст программы по индивидуальному варианту.

Листинг 2.1 – Текст программы по индивидуальному варианту

```
# ВАРИАНТ 12
 2
           .section .text
3
           .globl _start;
4
           len = 8 #Размер массива
 5
           enroll = 4 #Количество обрабатываемых элементов за одну итерацию
 6
           elem_sz = 4 #Размер одного элемента массива
 7
8
   _start:
9
           la x1, _x
10
           addi x20, x1, elem_sz*len #Адрес последнего элемента
11 lp:
12
           lw x2, 0(x1)
13
           lw x3, 4(x1)
           add x31, x31, x2 #!
14
15
           add x31, x31, x3
           lw x4, 8(x1)
16
17
           1w x5, 12(x1)
           add x31, x31, x4
18
19
           add x31, x31, x5
20
           addi x1, x1, elem_sz*enroll
21
           bne x1, x20, lp
           addi x31, x31, 1
22
23 lp2: j lp2
24
25
           .section .data
           .4byte 0x1
26
  _x:
27
           .4byte 0x2
28
           .4byte 0x3
29
           .4byte 0x4
30
           .4byte 0x5
31
           .4byte 0x6
32
           .4byte 0x7
33
           .4byte 0x8
```

На листиге 2.2 приведен псевдокод на языке С, соответствующий про-

грамме варианта.

Листинг 2.2 – Псевдокод на языке С программы варианта

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
5 \inf x[] = \{1, 2, 3, 4, 5, 6, 7, 8\};
6
7 void _start()
8 {
9
       int *x1 = _x;
10
       int *x20 = x1 + len;
11
       int x31 = 0;
12
13
      do
14
       {
15
           int x2 = x1[0];
16
           int x3 = x1[1];
17
           x31 += x2;
18
           x31 += x3;
19
           int x4 = x1[2];
20
           int x5 = x1[3];
21
           x31 += x4;
22
           x31 += x5;
23
           x1 += enroll;
24
       } while(x1 != x20);
25
26
      x31++;
27
28
      while(1){}
29 }
```

На листиге 2.3 приведен дизассемблерный код.

Листинг 2.3 – Дизассемблерный листинг

```
1 80000000 < start>:
2 80000000:
                   00000097
                                                     x1,0x0
                                             auipc
3 80000004:
                   03c08093
                                             addi
                                                     x1,x1,60 # 8000003c <_x>
4 80000008:
                                                      x20, x1, 32
                   02008a13
                                             addi
6 8000000c <lp>:
7 8000000c:
                   0000a103
                                                     x2,0(x1)
                                             lw
8 80000010:
                   0040a183
                                                      x3,4(x1)
                                             lw
9 80000014:
                                                      x31,x31,x2
                   002f8fb3
                                             add
10 80000018:
                   003f8fb3
                                                     x31,x31,x3
                                             add
11 8000001c:
                   0080a203
                                             lw
                                                     x4,8(x1)
12 80000020:
                   00c0a283
                                                     x5,12(x1)
                                             lw
13 80000024:
                                                     x31,x31,x4
                   004f8fb3
                                             add
14 80000028:
                   005f8fb3
                                                     x31,x31,x5
                                             add
15 8000002c:
                   01008093
                                             addi
                                                     x1, x1, 16
16 80000030:
                   fd409ee3
                                                     x1,x20,8000000c <1p>
                                             bne
17 80000034:
                   001f8f93
                                             addi
                                                     x31,x31,1
18
19 80000038 <1p2>:
20 80000038:
                   0000006f
                                             jal
                                                      x0,80000038 <1p2>
```

2.2 Вывод

Проанализировав исходный текст программы можно сделать вывод, что в регистре x31 в конце выполнения программы должна содержаться сумма элементов массива +1 (37).

Получение снимка экрана, содержащего временную диаграмму выполнения стадий выборки и диспетчеризации команды с адресом 8000000 на 2-ой итерации.

3.1 Результаты выполнения

На рисунке 3.1 представлены временные диаграммы выполнения стадий выборки и диспетчеризации необходимой команды.

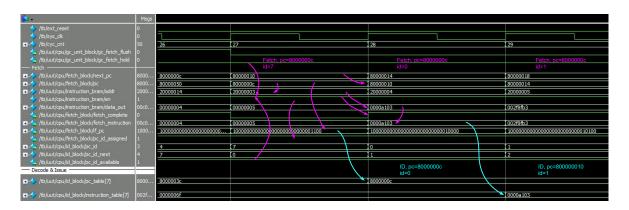


Рисунок 3.1 – Выборка и диспетчеризация команды с адресом 8000000 на второй итерации

3.2 Вывод

Выборка и диспетчеризация данной команды происходят в 27 и 28 тактах соответственно.

Получение снимка экрана, содержащего временную диаграмму выполнения стадий декодирования и планирования на выполнение команды с адресом 8000018 на 2-ой итерации.

4.1 Результаты выполнения

На рисунке 4.1 представлены временные диаграммы выполнения стадий декодирования и планирования на выполнение необходимой команды.

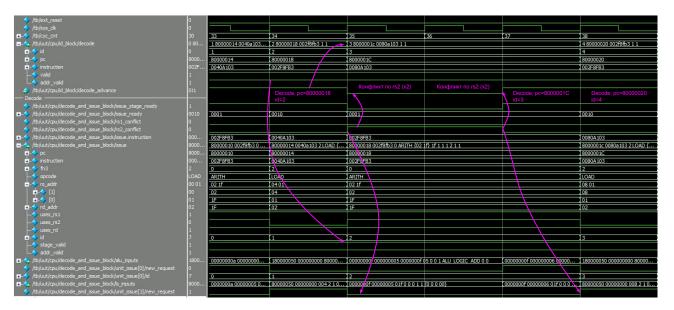


Рисунок 4.1 – Декодирование и планирование на выполнение команды с адресом 80000018 на второй итерации

4.2 Вывод

Декодирование данной команды происходит на 34 такте, однако так как представленная команда является командой загрузки из памяти, а следующая команда использует загруженное значение, возникает конфликт по rs2 (x2) и декодирование следующей команды происходит только на 37 такте.

Получение снимка экрана, содержащего временную диаграмму выполнения стадии выполнения команды с адресом 8000002с на 1-ой итерации.

5.1 Результаты выполнения

На рисунке 5.1 представлены временные диаграммы выполнения стадии выполнения необходимой команды.

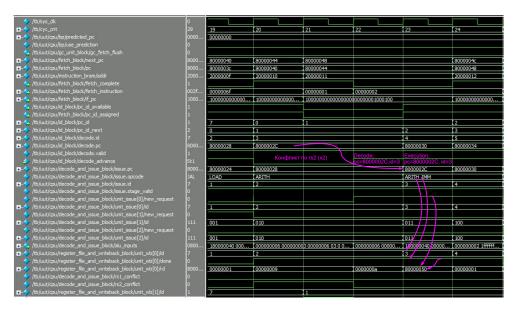


Рисунок 5.1 – Выполнение команды с адресом 8000002с на первой итерации

5.2 Вывод

Представленная команда является командой обработки данных и выполеняется на АЛУ за 1 такт, а именно на 23-ем такте, при этом, так как предыдущая комнда является командой загрузки из памяти, а результат загрузки используется в данной комаде, 2 такта происходит ожидание завершения выполнения команды загрузки блоком обращения к памяти.

6.1 Проверка результата

Значение регистра x31 представлено на рисунке 6.1. Значение равно 25 в шестнадцатиричной системе, что соответствует значению 37 в десятиричной системе счисления, приведенном в задании 1.



Рисунок 6.1 – Проверка значения регистра х31

6.2 Временные диаграммы сигналов стадий выполнения

В данном подразделе представлены временные диаграммы сигналов стадий выполнения сигналов команды add x31, x31, x2, находящейся по адресу 0x8000014 (на первой итерации).

На рисунке 6.2 представлены временные диаграммы сигналов стадий выборки и диспетчеризации. На рисунке 6.3 представлены временные диаграммы сигналов стадий декодирования и планирования на выполнение. На рисунке 6.4 представлены временные диаграммы сигналов стадии выполнения.

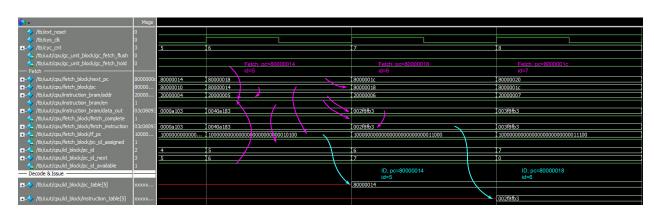


Рисунок 6.2 – Выборка и диспетчеризация команды #!

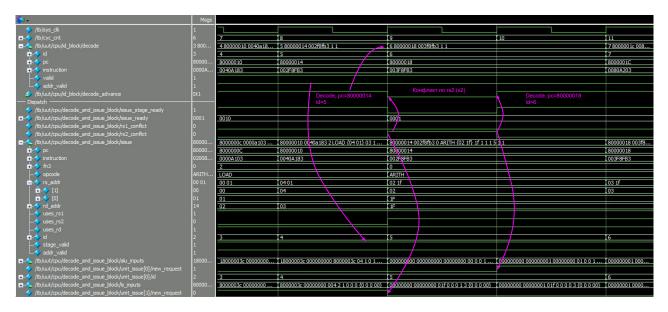


Рисунок 6.3 – Декодирование и планирование на выполнение команды #!

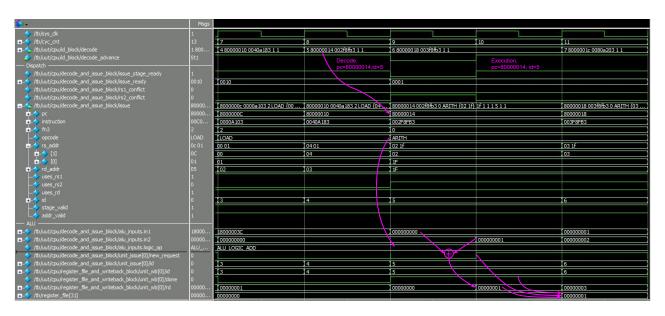


Рисунок 6.4 – Выполнение команды #!

6.3 Трасса выполнения программы

На рисунке 6.5 представлена трасса выполнения программы.

Проанализировав трассу выполнения программы, можно сделать вывод о том, что конфликты возникают тогда, когда блок обращения к памяти не успевает загрузить необходимое для выполнения арифметической операции значение. Это происходит потому, что блок обращения к памяти выполняет загрузку за три такта, а арифметическая команда следует за загрузкой нужной переменной через одну команду, таким образом блок обращения к памяти завершает только 2 такта обработки.

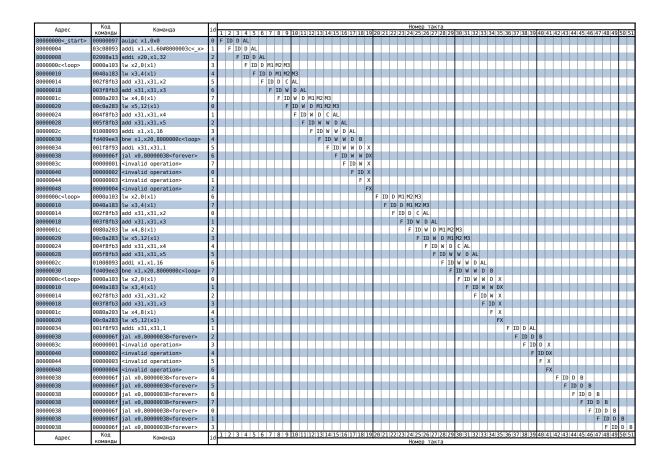


Рисунок 6.5 – Трасса выполнения программы

6.4 Оптимизация программы

Хотя блок обращения к памяти выполняет команду за 3 такта, он может принимать команду на вход каждый такт, таким образом, при последовательном расположении команд загрузки мы сможем увеличить количество команд между загрузкой и арифметической операцией, исключив возникающие конфликты и недопустив новые.

На листинге 6.1 приведен текст оптимизированной программы по индивидуальному варианту.

Листинг 6.1 – Текст оптимизированной программы по индивидуальному варианту

```
# ВАРИАНТ 12 (оптимизация)
2
           .section .text
3
           .globl _start;
4
           len = 8 #Размер массива
 5
           enroll = 4 #Количество обрабатываемых элементов за одну итерацию
6
           elem_sz = 4 #Размер одного элемента массива
7
8
  _start:
9
           la x1, _x
10
           addi x20, x1, elem_sz*len #Адрес последнего элемента
11 lp:
12
           lw x2, 0(x1)
13
           1w x3, 4(x1)
14
           lw x4, 8(x1)
           1w x5, 12(x1)
15
16
           add x31, x31, x2 #!
17
           add x31, x31, x3
18
           add x31, x31, x4
           add x31, x31, x5
19
20
           addi x1, x1, elem_sz*enroll
           bne x1, x20, lp
21
           addi x31, x31, 1
22
23 lp2: j lp2
24
25
           .section .data
26 _x:
           .4byte 0x1
27
           .4byte 0x2
           .4byte 0x3
28
29
           .4byte 0x4
30
           .4byte 0x5
31
           .4byte 0x6
32
           .4byte 0x7
33
           .4byte 0x8
```

На листиге 6.2 приведен псевдокод на языке С, соответствующий оптимизированной программе варианта.

Листинг 6.2 – Псевдокод на языке С оптимизированной программы варианта

```
1 #define len 8
2 #define enroll 4
3 #define elem_sz 4
5 \inf _x[] = \{1, 2, 3, 4, 5, 6, 7, 8\};
7 void _start()
8 {
9
      int *x1 = _x;
10
       int *x20 = x1 + len;
      int x31 = 0;
11
12
13
      do
14
       {
15
           int x2 = x1[0];
16
           int x3 = x1[1];
17
           int x4 = x1[2];
18
           int x5 = x1[3];
19
           x31 += x2;
20
           x31 += x3;
21
           x31 += x4;
22
           x31 += x5;
23
           x1 += enroll;
24
       } while(x1 != x20);
25
26
      x31++;
27
28
      while(1){}
29 }
```

На листиге 6.3 приведен дизассемблерный код оптимизированной программы.

Листинг 6.3 – Дизассемблерный листинг

```
1 80000000 < start>:
2 80000000:
                   00000097
                                             auipc
                                                      x1,0x0
3 80000004:
                   03c08093
                                             addi
                                                      x1,x1,60 # 8000003c <_x>
4 80000008:
                   02008a13
                                                      x20, x1, 32
                                             addi
6 8000000c <lp>:
7 8000000c:
                   0000a103
                                                      x2,0(x1)
                                             lw
8 80000010:
                                                      x3,4(x1)
                   0040a183
                                             lw
9 80000014:
                   0080a203
                                                      x4,8(x1)
                                             lw
10 80000018:
                   00c0a283
                                                      x5,12(x1)
11 8000001c:
                   002f8fb3
                                             add
                                                      x31, x31, x2
12 80000020:
                   003f8fb3
                                             add
                                                      x31,x31,x3
13 80000024:
                                                      x31,x31,x4
                   004f8fb3
                                             add
14 80000028:
                                                      x31,x31,x5
                   005f8fb3
                                             add
15 8000002c:
                   01008093
                                                      x1, x1, 16
                                             addi
16 80000030:
                                                      x1,x20,8000000c <lp>
                   fd409ee3
                                             bne
17 80000034:
                   001f8f93
                                             addi
                                                      x31, x31, 1
18
19 80000038 <1p2>:
20 80000038:
                   0000006f
                                             jal
                                                      x0,80000038 <1p2>
```

На рисунке 6.6 представлена трасса выполнения оптимизированной программы.

6.5 Вывод

При выполнении неоптимизированной программы вычисления завершаются за 41 такт, при выполнении оптимизированной — за 37 (без учета бесконечного цикла). Таким образом, в результате оптимизации удалось сократить выполнение на 4 такта, то есть получилось ускорить программу на $\frac{4}{41} \approx 0.1(10\%)$

	Код	V												_			_					Ном	ер	так	та																				
Адрес	команды	Команда	id				5	6 7	7 8	9	10	11	12 1	3 1	4 15	16	17	18	19 2	0 2	1 22	23	24	25 2	6 2	7 28	29	30	31 3	2 3	3 34	35	36	37	38 3	9 40	41	42	43	14 4	5 4	6 4	7 4		
80000000<_start>	00000097	auipc x1,0x0	0	F II				T						T						T					T					T										T	T	T			
80000004	03c08093	addi x1,x1,60#8000003c<_x>	1	F		D																																							
80000008	02008a13	addi x20,x1,32	2		F	ID	D A	۱L																																					
8000000c <loop></loop>	0000a103	lw x2,0(x1)	3			F	ID				П																																		
80000010	0040a183	lw x3,4(x1)	4				F 3	D E	M	1 M2	МЗ																																		
80000014	0080a203	lw x4,8(x1)	5		П			FΙ	D D	M1	M2	МЗ							Т	Т	П									Т						Т						Т	Т		
80000018	00c0a283	lw x5,12(x1)	6					F	I	D	M1	M2	МЗ																																
8000001c	002f8fb3	add x31,x31,x2	7		П			Т	F	ID	D	AL							Т	Т	П									Т						Т						Т	Т		
80000020	003f8fb3	add x31,x31,x3	Θ							F	ID	D	AL																																
80000024	004f8fb3	add x31,x31,x4	1		Т			Т	Т	П	F	ID	D A	L	Т				Т	Т	Т			Т	Т	Т				Т	Т					Т				Т	Т	Т	Т		
80000028	005f8fb3	add x31,x31,x5	2									F	ID I	D A	L																														
8000002c	01008093	addi x1,x1,16	3		Т			Т	Т	П			F I	D 0	AL				Т	Т	Т			Т	Т	Т				Т	Т					Т				Т	Т	Т	Т		
80000030	fd409ee3	bne x1,x20,8000000c <loop></loop>	4											FI	D D	В																													
80000034	001f8f93	addi x31,x31,1	5		Т				Т					F	ID	D	Х			Т	Т				Т							Т				Т			T	Т		Т	Т		
80000038	0000006f	jal x0,80000038 <forever></forever>	6												F	ID	DX																												
8000003c	00000001	<invalid operation=""></invalid>	7					Т								F			Т	T																Г			1	T	Т	Т	Т		
80000040	00000002	<invalid operation=""></invalid>	0														FX																												
8000000c <loop></loop>		lw x2,0(x1)	6															F I	ID I	М	1 M2	МЗ			Т											Т				T			Т		
80000010	0040a183	lw x3.4(x1)	7																FI	D D	M1	M2	МЗ																						
80000014	0080a203	lw x4,8(x1)	Θ		т			т		П					т								M2 I	43	т	Т				т		П				П			П	т	т	Т	т		
80000018		lw x5.12(x1)	1																	F	ID	D	M1 I	12 M	13																				
8000001c	002f8fb3	add x31,x31,x2	2		т			т		П					т				т	Т	F	ID	D A	AL		Т				т		П				П			П	т	т	Т	т		
80000020	003f8fb3	add x31.x31.x3	3																			F	ID	D A	L																				
80000024	004f8fb3	add x31,x31,x4	4		т			т		П					т				т	Т	т		F :	ID I) AI	L				т		П				П			П	т	т	Т	т		
80000028	005f8fb3	add x31.x31.x5	5																					FΙ	D D	AL																			
8000002c	01008093	addi x1,x1,16	6		т			т		П					т				т	Т	т				- II	D D	AL			т		П				П			П	т	т	Т	т		
80000030	fd409ee3	bne x1,x20,8000000c <loop></loop>	7																						F	ID	D	В																	
8000000c <loop></loop>		lw x2,0(x1)	Θ		т			т		П					т				т	Т	т			т	т	F	ID	D	х	т		П				П			П	т	т	Т	т		
80000010		lw x3,4(x1)	1																									ID																	
80000014		lw x4.8(x1)	2		т			т		П					т				т	Т	т			т	т	Т		F	х	т		П				П			П	т	т	Т	т		
80000018		lw x5,12(x1)	3																										FX																
80000034	001f8f93	addi x31,x31,1	1																											FΙ	D D	AL									T	T	Т		
		jal x0,80000038 <forever></forever>	2																												IC		В												
8000003c	00000001	<pre><invalid operation=""></invalid></pre>	3																												F	ID	D	х								T	Т		
	00000002	<pre><invalid operation=""></invalid></pre>	4																														ID												
80000044	00000003	<invalid operation=""></invalid>	5																														F	х						Т		T	Т		
80000048	00000004	<invalid operation=""></invalid>	6																															FX											
80000038	0000006f	jal x0,80000038 <forever></forever>	4																																F I	D	В			Т		T	Т		
80000038	0000006f	jal x0,80000038 <forever></forever>	5																																	ID		В							
80000038	0000006f	jal x0,80000038 <forever></forever>	6																																			D	В			T	Т		
80000038	0000006f	jal x0,80000038 <forever></forever>	7																																			ID		В					
80000038	0000006f	ial x0.80000038 <forever></forever>	Θ																	Т					Т													F	ID	D	В		Т		
80000038	0000006f	jal x0,80000038 <forever></forever>	1																																				F :			3			
80000038	0000006f	jal x0,80000038 <forever></forever>	2																																					_	_	_) E		
	Код		Ħ	1 2	3	4	5	6 7	7 8	9	10	11	12 1	3 1	4 15	16	17	18	19 2	0 2	1 22	23	24	25 2	6 2	7 28	29	30	31 3	2 3	3 34	35	36	37	38 3	9 40	41	42	43						
Адрес	команды	Команда	10								_	_					_						ер											- 1		, -	_	_			_		_		

Рисунок 6.6 – Трасса выполнения оптимизированной программы

Заключение

В ходе лабораторной работы были изучены принципы функционирования и построения, а также особенности архитектуры суперскалярных конвейерных микропроцессоров на примере микропроцессорного ядра Taiga, реализующего систему команд семейства RISC-V. Таким образом, цель данной работы была достигнута.