Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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)

О т ч е т

по лабораторной работе № 2

Название: Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина: Архитектура ЭВМ

Студент гр. ИУ7-53Б _____ Н.В.Куликов
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ С.В. Ибрагимов
(Подпись, дата) (И.О. Фамилия)

2025 год

Цель работы

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.

Используемые средства

Для достижения поставленных целей в настоящей лабораторной работе используется синтезируемое описание микропроцессорного ядра Taiga, реализующего систему команд RV32I семейства RISC-V. Данное описание выполнено на языке описания аппаратуры SystemVerilog.

В ходе лабораторной работы используется средство моделирования MentorGraphics Modelsim для моделирования работы исследуемого микропроцессора в процессе выполнения программы и наблюдения формы внутренних сигналов.

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.

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. В том случае, если операндами являются регистры, мы будем их называть исходными регистрами (rs, source register), регистр, в который сохраняется результат — целевым регистром (rd, destination register).

Исследуемая программа

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

_start:
        addi x20, x0, len/enroll
        la x1, _x
     add x31, x0, x0
lp:
        lw x2, 0(x1)
        add x31, x31, x2
        lw x3, 4(x1)
        add x31, x31, x3 #!
        addi x1, x1, elem_sz*enroll
        addi x20, x20, -1
        bne x20, x0, 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
```

Листинг программы по индивидуальному варианту

```
Disassembly of section .text:

80000000 <_start>:
80000000:  00400a13              addi  x20,x0,4
80000004:  00000097              auipc x1,0x0
80000008:  03008093              addi  x1,x1,48 # 80000034 <_x>
8000000c:  00000fb3              add   x31,x0,x0

80000010 <lp>:
80000010:  0000a103              lw    x2,0(x1)
80000014:  002f8fb3              add   x31,x31,x2
80000018:  0040a183              lw    x3,4(x1)
8000001c:  003f8fb3              add   x31,x31,x3
80000020:  00808093              addi  x1,x1,8
80000024:  fffa0a13              addi  x20,x20,-1
80000028:  fe0a14e3              bne   x20,x0,80000010 <lp>
8000002c:  001f8f93              addi  x31,x31,1

80000030 <lp2>:
80000030:  0000006f              jal   x0,80000030 <lp2>

Disassembly of section .data:

80000034 <_x>:
80000034:  0001                  .insn 2, 0x0001
80000036:  0000                  .insn 2, 0x
80000038:  0002                  .insn 2, 0x0002
8000003a:  0000                  .insn 2, 0x
8000003c:  00000003              lb    x0,0(x0) # 0 <enroll-
0x2>
80000040:  0004                  .insn 2, 0x0004
80000042:  0000                  .insn 2, 0x
80000044:  0005                  .insn 2, 0x0005
80000046:  0000                  .insn 2, 0x
80000048:  0006                  .insn 2, 0x0006
8000004a:  0000                  .insn 2, 0x
8000004c:  00000007              .insn 4, 0x0007
80000050:  0008                  .insn 2, 0x0008
```

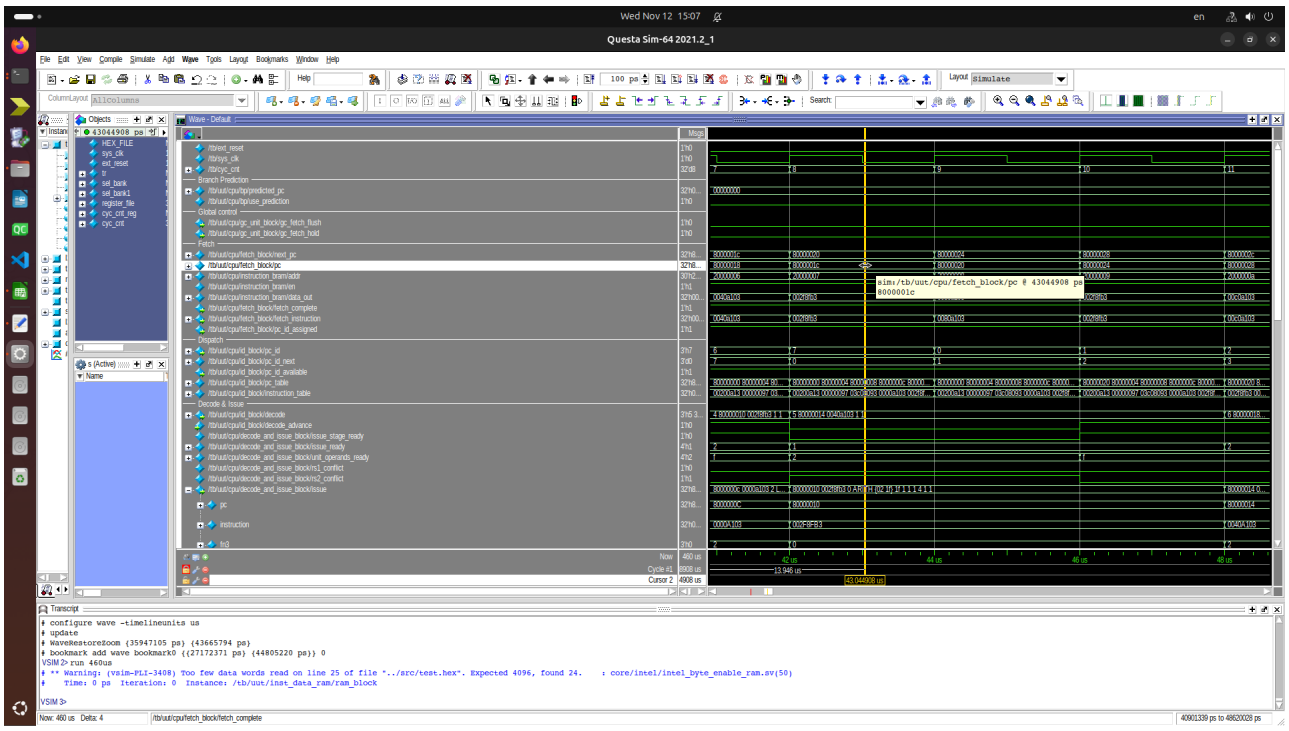
Дизассемблерный листинг программы по индивидуальному варианту

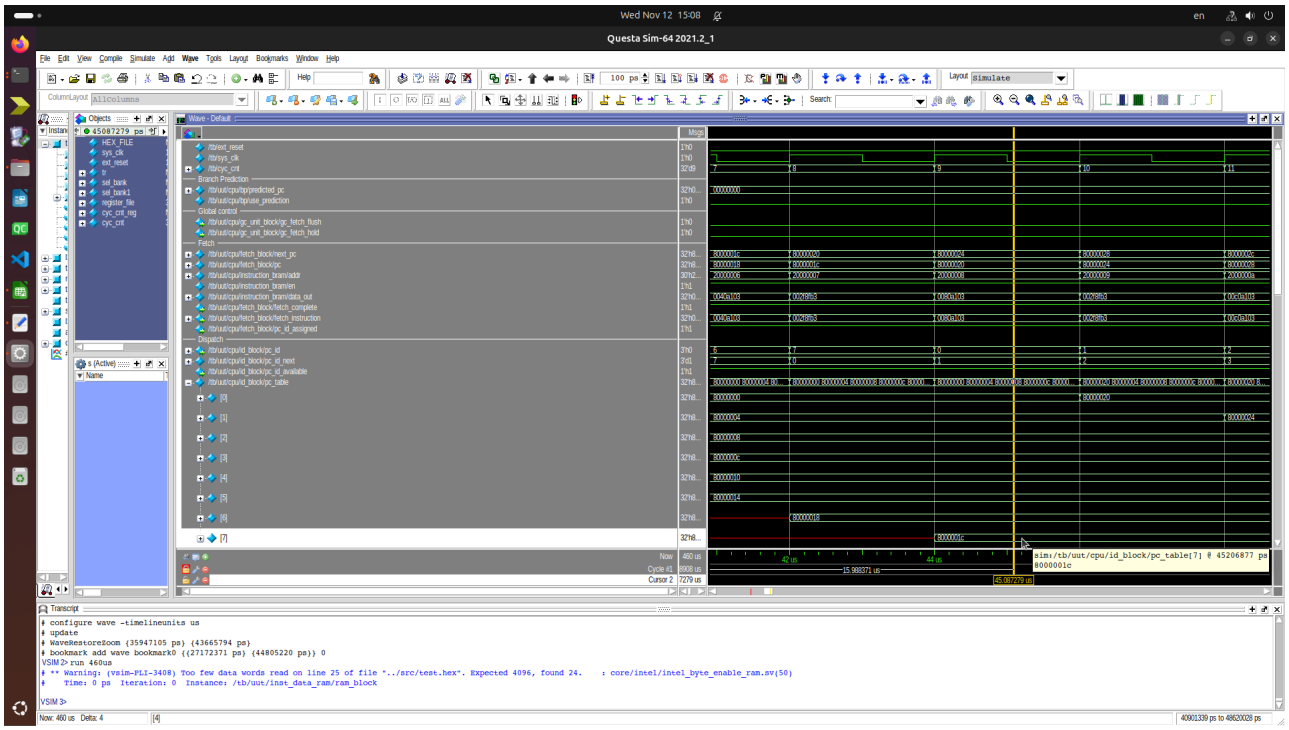
```
int main() {
    int _x[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int *x1 = _x;
    int x20 = 4;     // (len/enroll = 8/2)
    int x31 = 0;

    while (x20 != 0) {
        int x2 = *x1;
        x31 += x2;
        int x3 = *(x1 + 1);
        x31 += x3;

        x1 += 2;
        x20--;
    }

    x31 += 1;

    while (1) {}
    return 0;
}
```

Псевдокод на языке Си, поясняющий работу программы

Задание №2

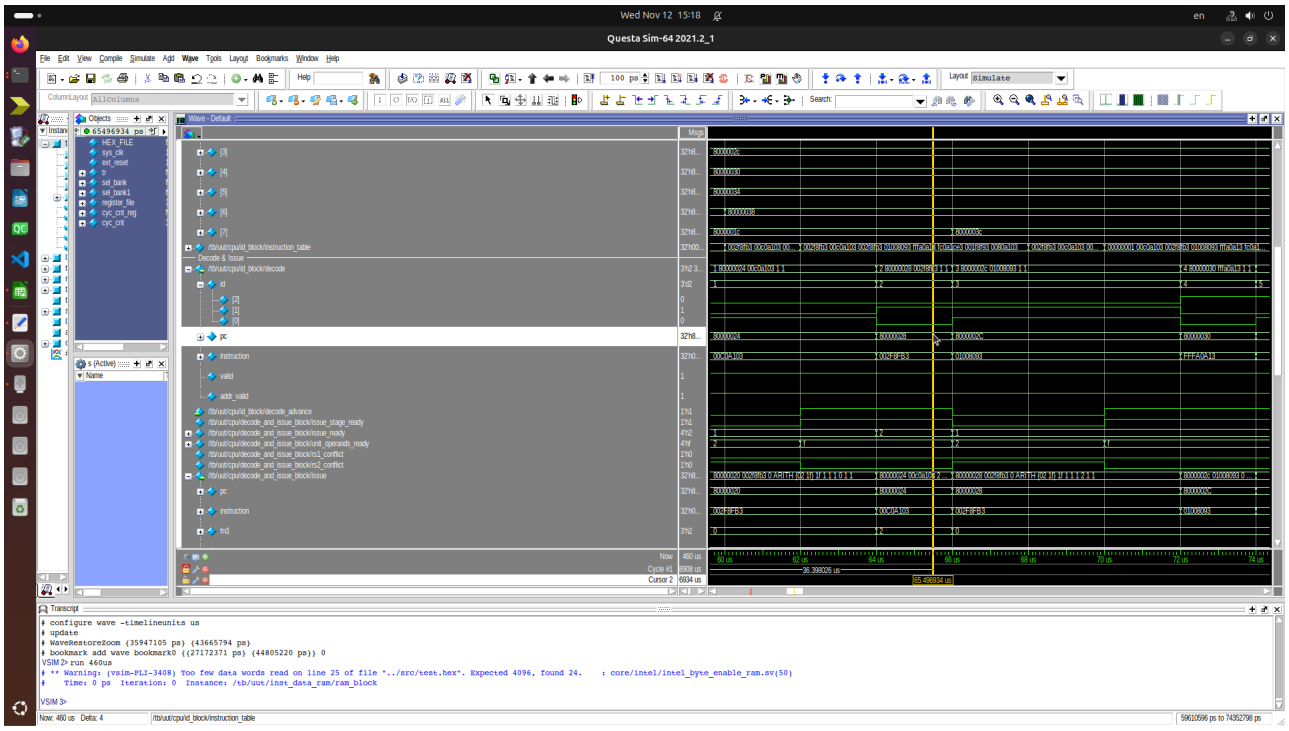
Выборка и диспетчеризация команды 8000001c.



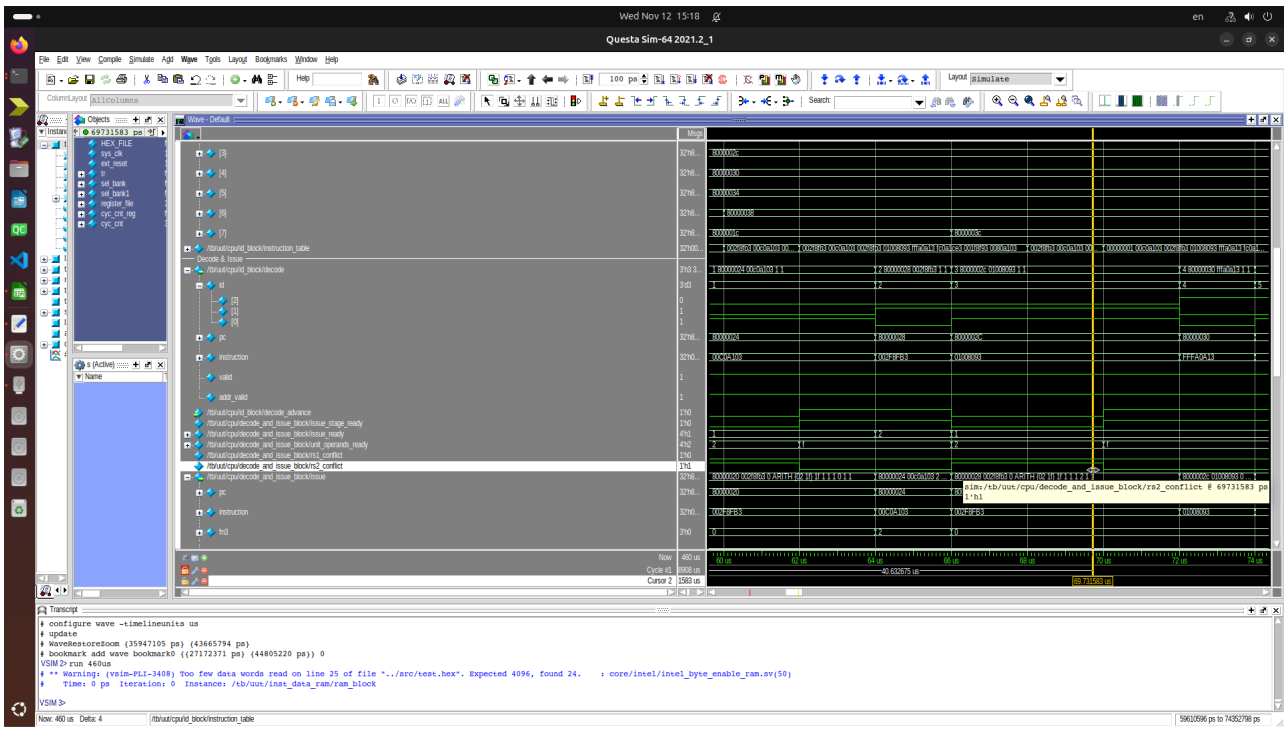


Задание №3

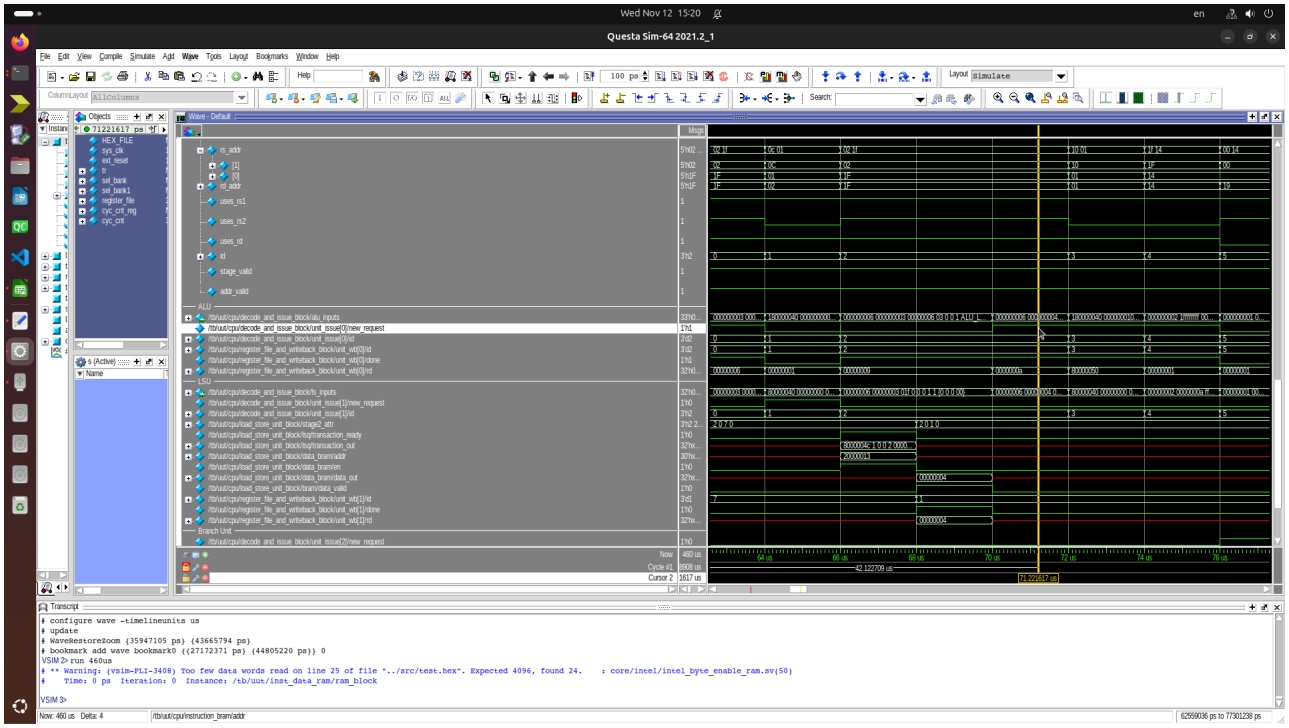
Декодирование и планирование команды 80000028.



1) Декодирование.



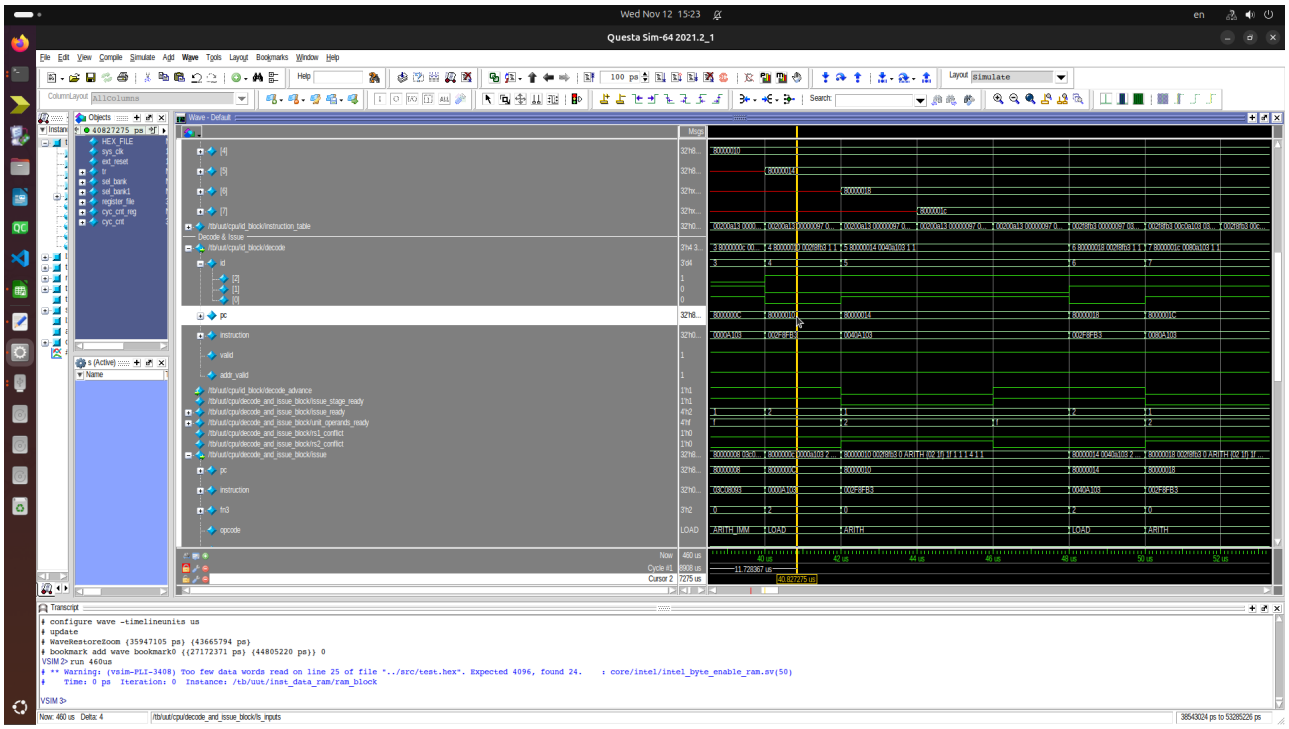
2) Конфликты (установлен флаг rs2_conflict).



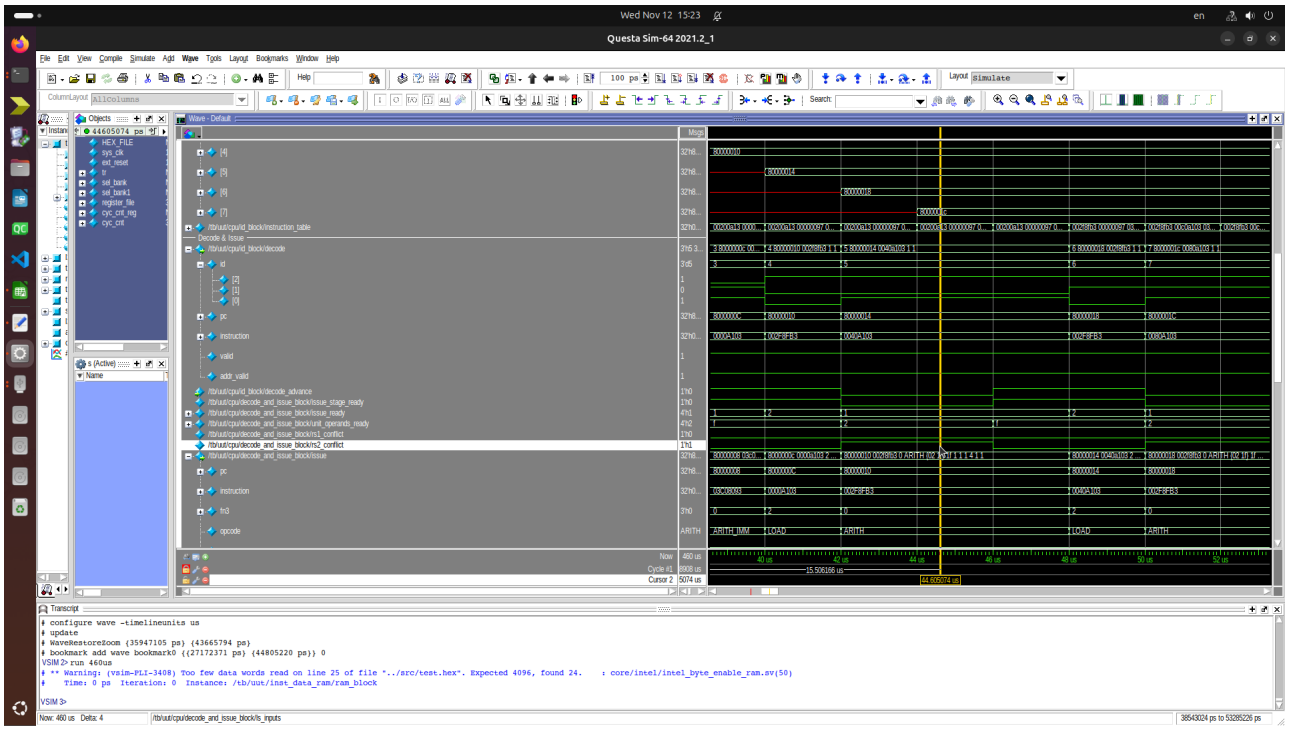
3) Выполнение на ALU.

Задание №4

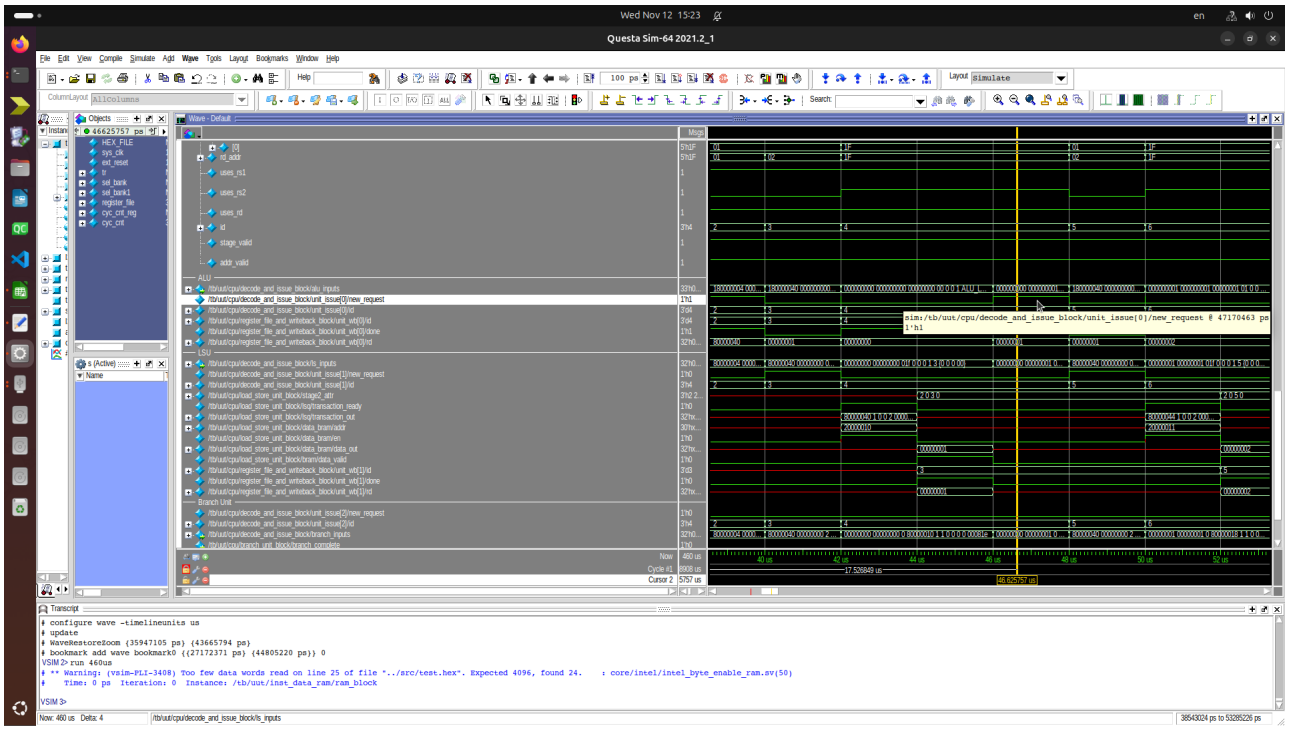
Выполнение команды 80000010.



1) Декодирование.

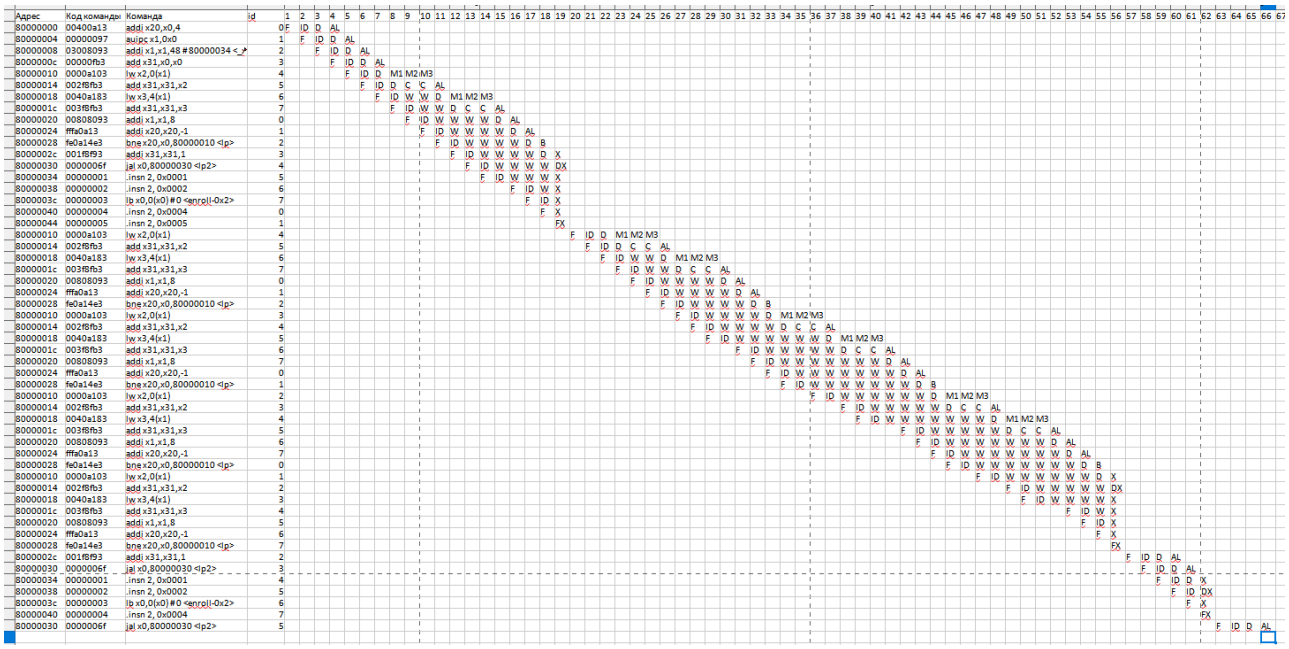


2) Конфликты (установлен флаг rs2_conflict).

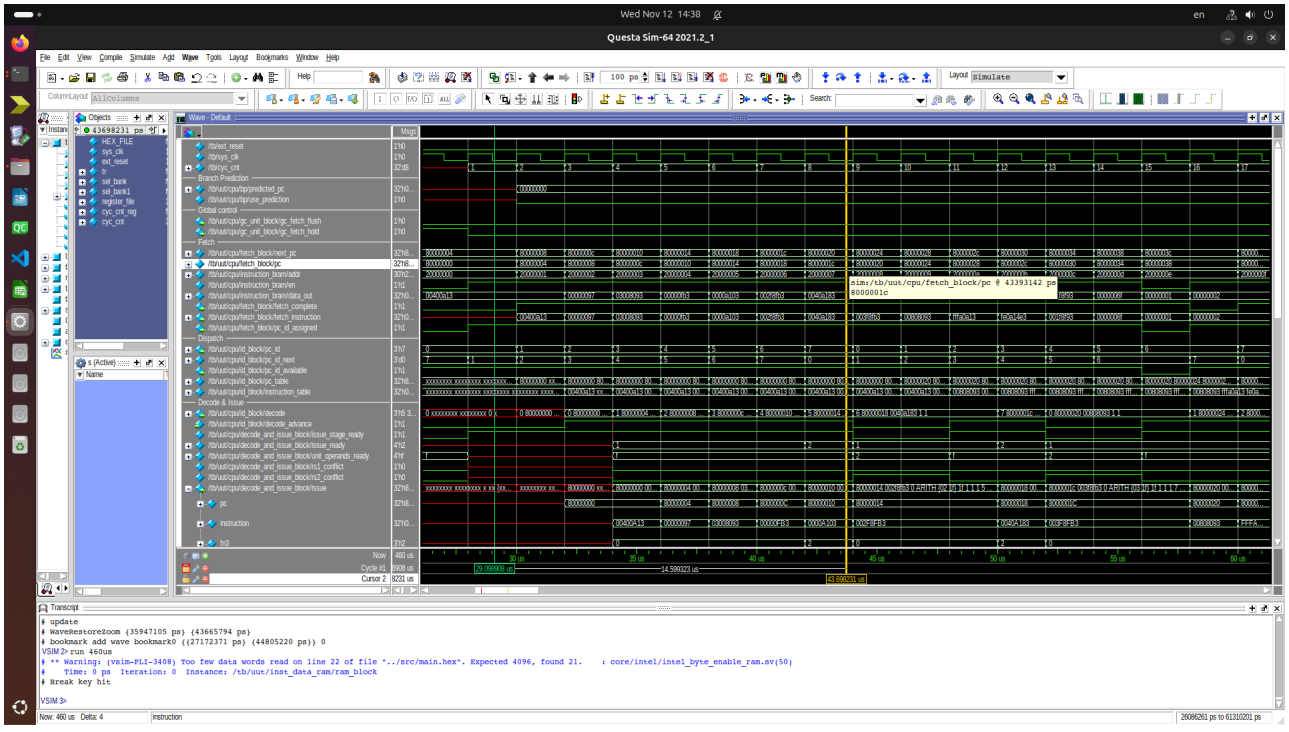


3) Выполнение на ALU.

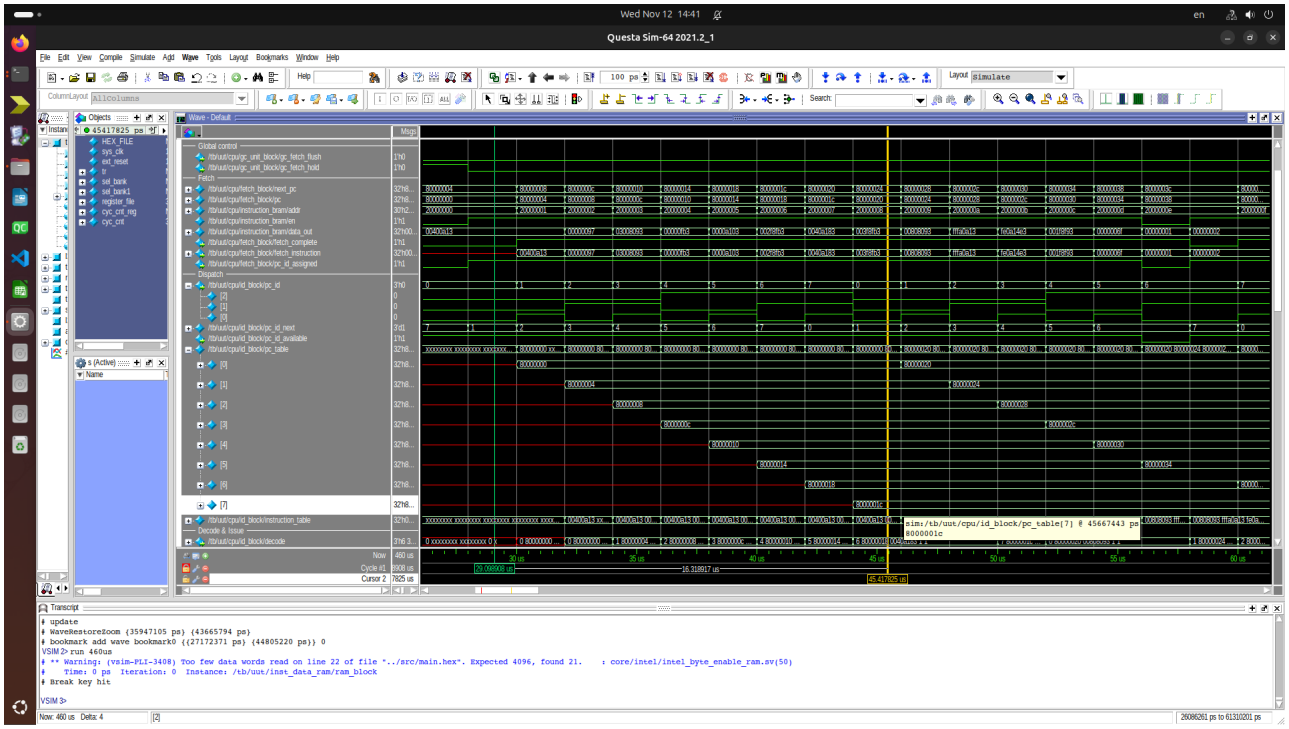
Трасса работы программы



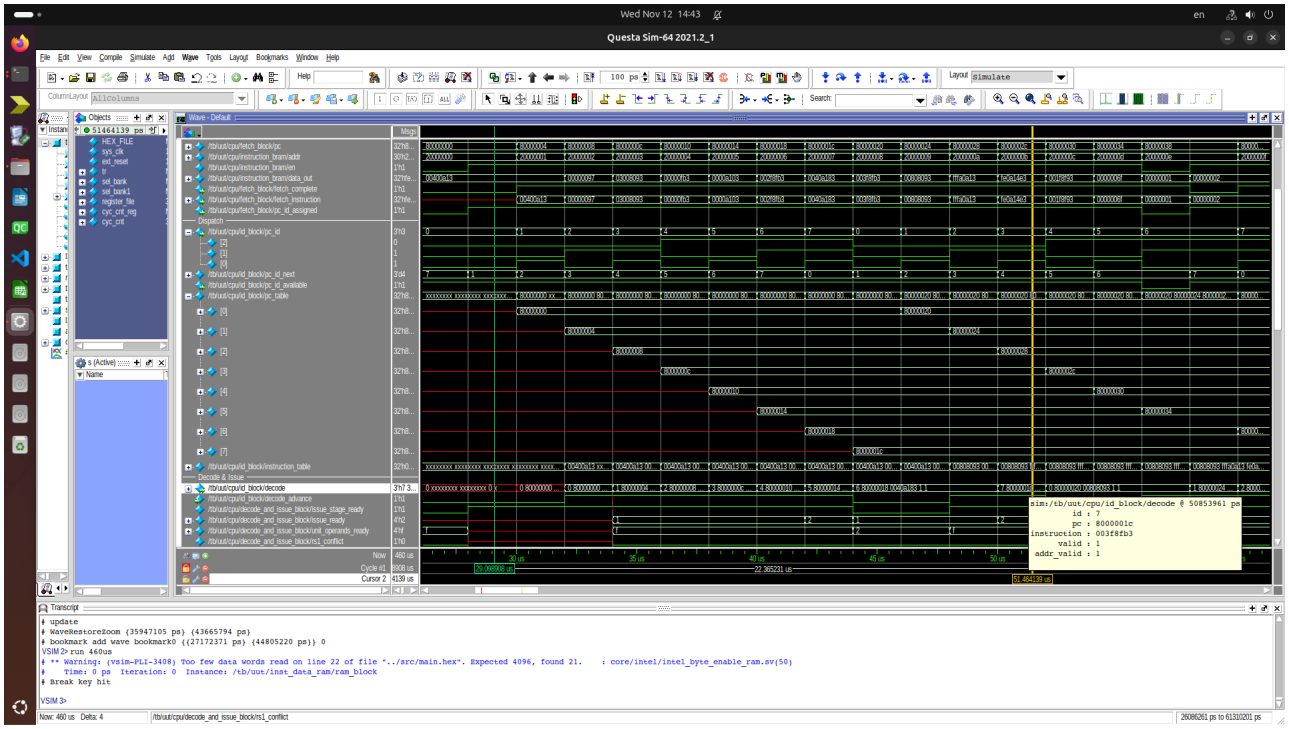
Временные диаграммы команды add x31, x31, x3



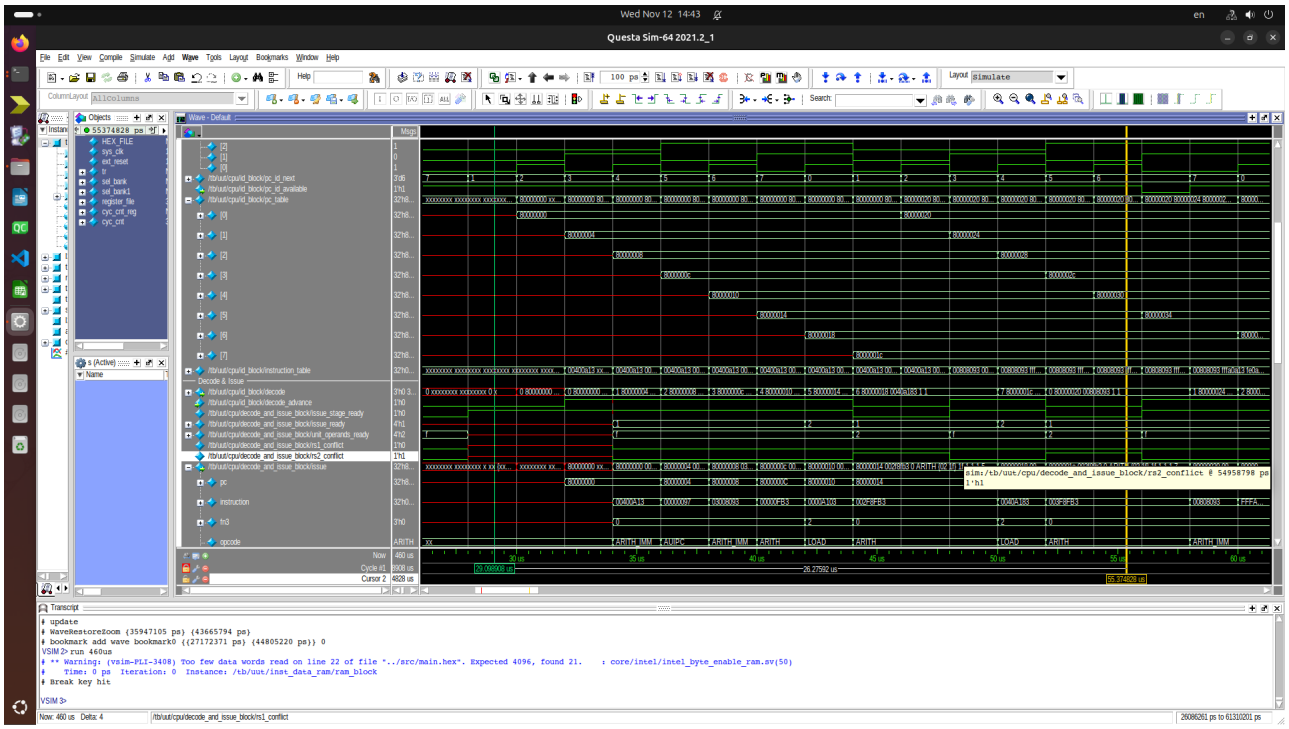
1) Выборка на 8 такте.



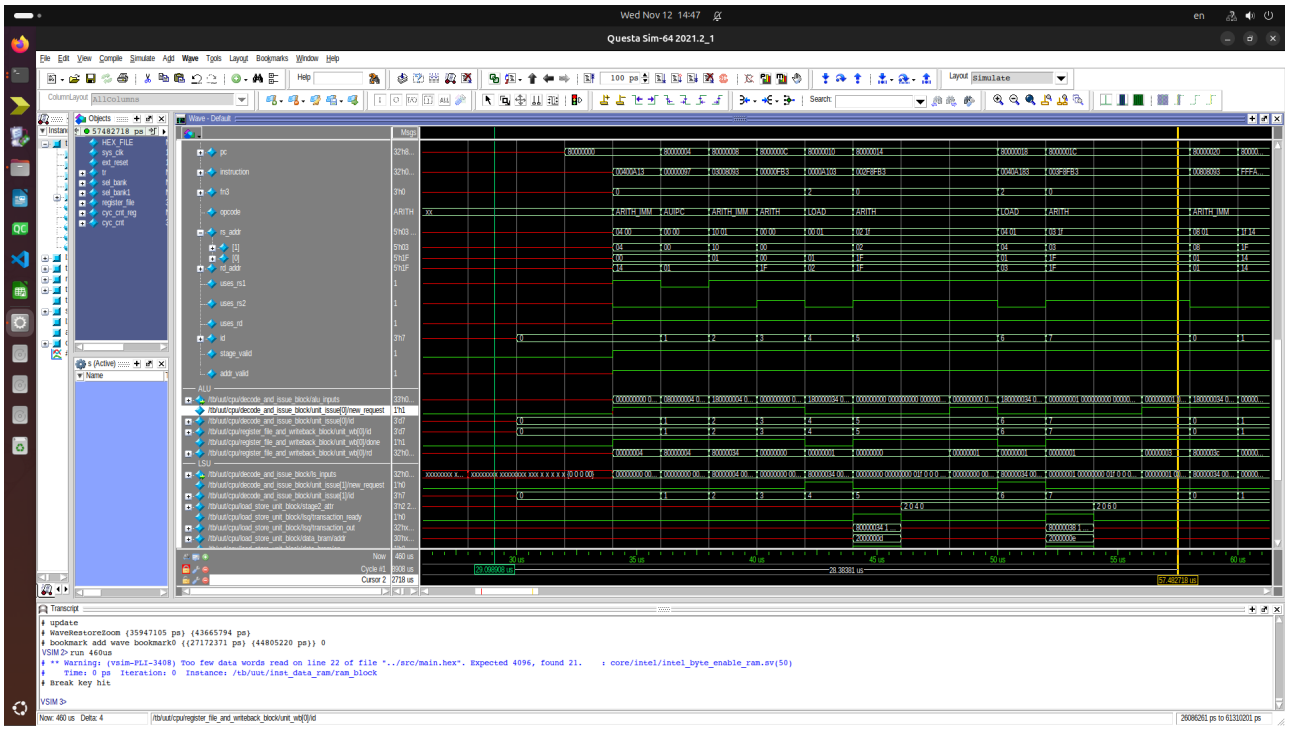
2) Диспетчеризация на 9 такте.



3) Загрузка блока декодирования (W) на 10 и 11 тактах. Декодирование на 12 такте.



4) Конфликты на 13 и 14 тактах (установлен флаг rs2_conflict).



5) Выполнение AL (установлен new_request) на 15 такте.

Вывод об эффективности работы программы

Проблема в исходном коде: процессор не может использовать результат инструкции lw сразу же в следующей инструкции add, так как загрузка из памяти занимает несколько тактов.

```
...
lw x2, 0(x1)
add x31, x31, x2
lw x3, 4(x1)
add x31, x31, x3
...
```

Решение: использование значения, загруженного на предыдущей итерации и загрузка значения для следующей итерации.

Оптимизированная программа

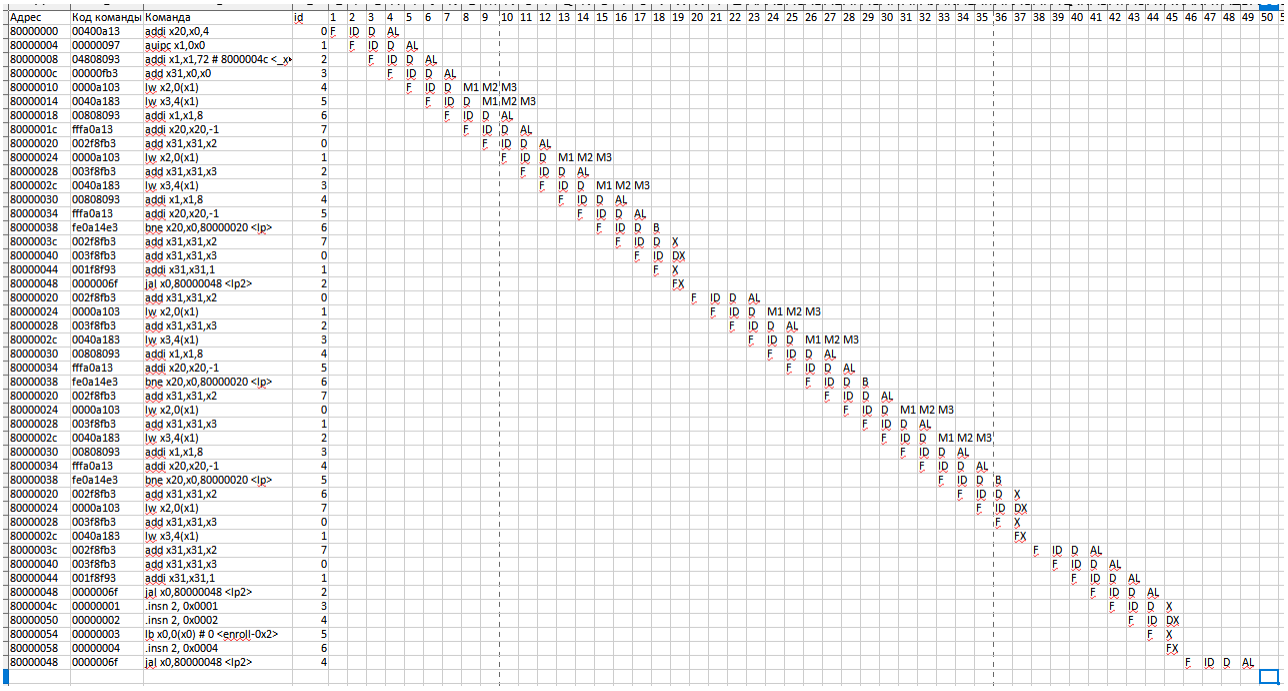
```
.section .text
    .globl _start
    len = 8
    enroll = 2
    elem_sz = 4

_start:
    addi x20, x0, len/enroll
    la x1, _x
    add x31, x0, x0

    lw x2, 0(x1)
    lw x3, 4(x1)
    addi x1, x1, elem_sz*enroll
    addi x20, x20, -1

lp:
    add x31, x31, x2
    lw x2, 0(x1)
    add x31, x31, x3
    lw x3, 4(x1)

    addi x1, x1, elem_sz*enroll
    addi x20, x20, -1
    bne x20, x0, lp


    add x31, x31, x2
    add x31, x31, x3

    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
```

Листинг оптимизированной программы

```
Disassembly of section .text:

80000000 <_start>:
80000000:  00400a13              addi  x20,x0,4
80000004:  00000097              auipc x1,0x0
80000008:  04808093              addi  x1,x1,72 # 8000004c <_x>
8000000c:  00000fb3              add   x31,x0,x0
80000010:  0000a103              lw    x2,0(x1)
80000014:  0040a183              lw    x3,4(x1)
80000018:  00808093              addi  x1,x1,8
8000001c:  fffa0a13              addi  x20,x20,-1

80000020 <lp>:
80000020:  002f8fb3              add   x31,x31,x2
80000024:  0000a103              lw    x2,0(x1)
80000028:  003f8fb3              add   x31,x31,x3
8000002c:  0040a183              lw    x3,4(x1)
80000030:  00808093              addi  x1,x1,8
80000034:  fffa0a13              addi  x20,x20,-1
80000038:  fe0a14e3              bne   x20,x0,80000020 <lp>
8000003c:  002f8fb3              add   x31,x31,x2
80000040:  003f8fb3              add   x31,x31,x3
80000044:  001f8f93              addi  x31,x31,1

80000048 <lp2>:
80000048:  0000006f              jal   x0,80000048 <lp2>

Disassembly of section .data:

8000004c <_x>:
8000004c:  0001                  .insn 2, 0x0001
8000004e:  0000                  .insn 2, 0x
80000050:  0002                  .insn 2, 0x0002
80000052:  0000                  .insn 2, 0x
80000054:  00000003              lb    x0,0(x0) # 0 <enroll-
0x2>
80000058:  0004                  .insn 2, 0x0004
8000005a:  0000                  .insn 2, 0x
8000005c:  0005                  .insn 2, 0x0005
8000005e:  0000                  .insn 2, 0x
80000060:  0006                  .insn 2, 0x0006
80000062:  0000                  .insn 2, 0x
80000064:  00000007              .insn 4, 0x0007
80000068:  0008                  .insn 2, 0x0008
```

Дизассемблерный листинг оптимизированной программы

Трасса работы оптимизированной программы



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	50
80000000	00400a13	addi x20,x0,4	0	F	ID	D	AL																																														
80000004	00000097	auipc x1,0x0	1		F	ID	D	AL																																													
80000008	04808093	addi x1,x1,72 # 8000004c <_x>	2			F	ID	D	AL																																											
8000000c	00000fb3	add x31,x0,x0	3				F	ID	D	AL																																										
80000010	0000a103	lw x2,0(x1)	4					F	ID	D	M1	M2	M3																																							
80000014	0040a183	lw x3,4(x1)	5						F	ID	D	M1	M2	M3																																					
80000018	00808093	addi x1,x1,8	6							F	ID	D	AL																																					
8000001c	fffa0a13	addi x20,x20,-1	7								F	ID	D	AL																																				
80000020	002f8fb3	add x31,x31,x2	0									F	ID	D	AL																																		
80000024	0000a103	lw x2,0(x1)	1										F	ID	D	M1	M2	M3																															
80000028	003f8fb3	add x31,x31,x3	2											F	ID	D	AL																															
8000002c	0040a183	lw x3,4(x1)	3												F	ID	D	M1	M2	M3																													
80000030	00808093	addi x1,x1,8	4													F	ID	D	AL																													
80000034	fffa0a13	addi x20,x20,-1	5														F	ID	D	AL																												
80000038	fe0a14e3	bne x20,x0,80000020 <lp>	6															F	ID	D	B																												
8000003c	002f8fb3	add x31,x31,x2	7																F	ID	D	X																											
80000040	003f8fb3	add x31,x31,x3	0																	F	ID	DX																											
80000044	001f8f93	addi x31,x31,1	1																		F	X																											
80000048	0000006f	jal x0,80000048 <lp2>	2																			FX																											
80000020	002f8fb3	add x31,x31,x2	0																				F	ID	D	AL																						
80000024	0000a103	lw x2,0(x1)	1																					F	ID	D	M1	M2	M3																		
80000028	003f8fb3	add x31,x31,x3	2																						F	ID	D	AL																		
8000002c	0040a183	lw x3,4(x1)	3																							F	ID	D	M1	M2	M3															
80000030	00808093	addi x1,x1,8	4																								F	ID	D	AL															
80000034	fffa0a13	addi x20,x20,-1	5																									F	ID	D	AL													
80000038	fe0a14e3	bne x20,x0,80000020 <lp>	6																										F	ID	D	B												
80000020	002f8fb3	add x31,x31,x2	7																											F	ID	D	AL										
80000024	0000a103	lw x2,0(x1)	0																												F	ID	D	M1	M2	M3						
80000028	003f8fb3	add x31,x31,x3	1																													F	ID	D	AL						
8000002c	0040a183	lw x3,4(x1)	2																														F	ID	D	M1	M2	M3			
80000030	00808093	addi x1,x1,8	3																															F	ID	D	AL				
80000034	fffa0a13	addi x20,x20,-1	4																																F	ID	D	AL			
80000038	fe0a14e3	bne x20,x0,80000020 <lp>	5																																	F	ID	D	B		
80000020	002f8fb3	add x31,x31,x2	6																																		F	ID	D	X	
80000024	0000a103	lw x2,0(x1)	7																																			F	ID	DX	
80000028	003f8fb3	add x31,x31,x3	0																																				F	X	
8000002c	0040a183	lw x3,4(x1)	1																																					FX	
8000003c	002f8fb3	add x31,x31,x2	7																																						F	ID	D	AL
80000040	003f8fb3	add x31,x31,x3	0																																						F	ID	D	AL
80000044	001f8f93	addi x31,x31,1	1																																							F	ID	D	AL
80000048	0000006f	jal x0,80000048 <lp2>	2																																								F	ID	D	AL
8000004c	00000001	.insn 2, 0x0001	3																																									F	ID	D	X
80000050	00000002	.insn 2, 0x0002	4																																										F	ID	DX
80000054	00000003	lb x0,0(x0) # 0 <enroll-0x2>	5																																											F	X
80000058	00000004	.insn 2, 0x0004	6																																												FX
80000048	0000006f	jal x0,80000048 <lp2>	4																																											F	ID	D	AL

Выводы

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.