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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

Отчет
по лабораторной работе № 1

по дисциплине «Организация ЭВМ и систем»

Тема: «Изучение принципов работы микропроцессорного ядра RISC-V»

Студент группы ИУ6-73Б _____ В.К. Залыгин
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ М.П. Калитвенцев
(Подпись, дата) (И.О. Фамилия)

2025 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.

Теоретические сведения

В лабораторной рассматривается вычислительная модель RISC-V RV32I: программа исполняется как последовательность инструкций, адресуемых через pc, и работает с 32 регистрами x0–x31 (при этом x0 всегда равен 0). Семантика команд задаётся формально через псевдокод: для вычислительных инструкций описывается формирование результата, для ветвлений — условие и вычисление следующего pc, а для load/store — вычисление эффективного адреса как rs1 + imm и чтение/запись данных нужной разрядности.

Аппаратный стенд построен вокруг ядра Taiga и локальной памяти на BRAM: она синхронная, двухпортовая и имеет фиксированную задержку доступа (это позволяет проще связывать «что случилось» с «на каком такте»). Память команд и данных здесь в одном адресном пространстве; в конфигурации лабораторной она отображается, начиная с 0x80000000, поэтому обращения ядра к этому диапазону фактически идут в локальную BRAM.

Микроархитектурно Taiga — конвейерный процессор с 4 стадиями (выборка → диспетчеризация/буфер метаданных → декодирование и планирование → выполнение в исполнительных блоках). Важное, что наблюдается в симуляции: команды могут «застревать» из-за зависимостей по регистрам (когда следующая инструкция хочет читать регистр, который ещё не записан результатом предыдущей) и из-за задержек в блоке памяти (особенно для загрузок/выгрузок). Для ветвлений используется предсказание

следующего pc; при ошибке предсказания конвейер вынужденно очищает/перенаправляет поток, что даёт видимые потери тактов.

Выполнение работы

В листинге 1 приведен код исходной программы.

Листинг 1 – Исходный код программы (Вариант 10)

```
        .section .text
        .globl _start;
        len = 8 #Размер массива
        enroll = 4 #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)
        lw x3, 4(x1) #!
        add x31, x31, x2
        add x31, x31, x3
        lw x4, 8(x1)
        lw x5, 12(x1)
        add x31, x31, x4
        add x31, x31, x5
        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
```

В листинге 2 приведет дизассемблерный листинг программы.

Листинг 2 – Дизассемблерный листинг.

```
bash-5.2# make
riscv64-linux-musl-as --march=rv32i test10.s -o test10.o
riscv64-linux-musl-ld -b elf32-littleriscv -T link.ld test10.o
-o test10.elf
riscv64-linux-musl-ld: warning: test10.elf has a LOAD segment
with RWX permissions
riscv64-linux-musl-objdump -D -M numeric,no-aliases -t
test10.elf

test10.elf:     file format elf32-littleriscv

SYMBOL TABLE:
80000000 l    d  .text  00000000 .text
80000044 l    d  .data  00000000 .data
00000000 l    df *ABS*  00000000 test10.o
00000008 l       *ABS*  00000000 len
00000004 l       *ABS*  00000000 enroll
00000004 l       *ABS*  00000000 elem_sz
80000044 l       .data  00000000 _x
80000010 l       .text  00000000 lp
80000040 l       .text  00000000 lp2
80000000 g       .text  00000000 _start
80000064 g       .data  00000000 _end



Disassembly of section .text:

80000000 <_start>:
80000000:  00200a13            addi  x20,x0,2
80000004:  00000097            auipc  x1,0x0
80000008:  04008093            addi  x1,x1,64 # 80000044 <_x>
8000000c:  00000fb3            add  x31,x0,x0

80000010 <lp>:
80000010:  0000a103            lw  x2,0(x1)
80000014:  0040a183            lw  x3,4(x1)
80000018:  002f8fb3            add  x31,x31,x2
8000001c:  003f8fb3            add  x31,x31,x3
80000020:  0080a203            lw  x4,8(x1)
80000024:  00c0a283            lw  x5,12(x1)
80000028:  004f8fb3            add  x31,x31,x4
8000002c:  005f8fb3            add  x31,x31,x5
80000030:  01008093            addi  x1,x1,16
80000034:  fffa0a13            addi  x20,x20,-1
80000038:  fc0a1ce3            bne  x20,x0,80000010 <lp>
8000003c:  001f8f93            addi  x31,x31,1

80000040 <lp2>:
```

```
80000040:  0000006f            jal  x0,80000040 <lp2>

  ...
riscv64-linux-musl-objcopy -O binary --reverse-bytes=4
test10.elf test10.bin
xxd -g 4 -c 4 -p test10.bin test10.hex
rm test10.o test10.bin test10.elf
```

В листинге 3 представлен псевдокод, поясняющий работу программы.

Листинг 3 – Псевдокод, поясняющий работу программы.

```
#define LEN     8
#define ENROLL  4

static uint32_t _x[LEN] = { 1, 2, 3, 4, 5, 6, 7, 8 };

int32_t result;

void _start(void) {
    int32_t sum = 0;
    uint32_t *p = _x;
    int32_t iters = LEN / ENROLL;

    while (iters != 0) {
        sum += (int32_t)p[0];
        sum += (int32_t)p[1];
        sum += (int32_t)p[2];
        sum += (int32_t)p[3];

        p += ENROLL;
        iters -= 1;
    }

    sum += 1;
    result = sum;

    for (;;);
}
```

В задании 2 требовалось запустить симуляцию тестовой программы test.s в Modelsim и, согласно таблице вариантов, снять временную диаграмму прохождения выбранной команды через стадии выборки (F) и диспетчеризации (ID) (для команд из тела цикла также учитывалась указанная итерация). Скриншот в ходе выполнения задания 2 представлен на рисунке 1.

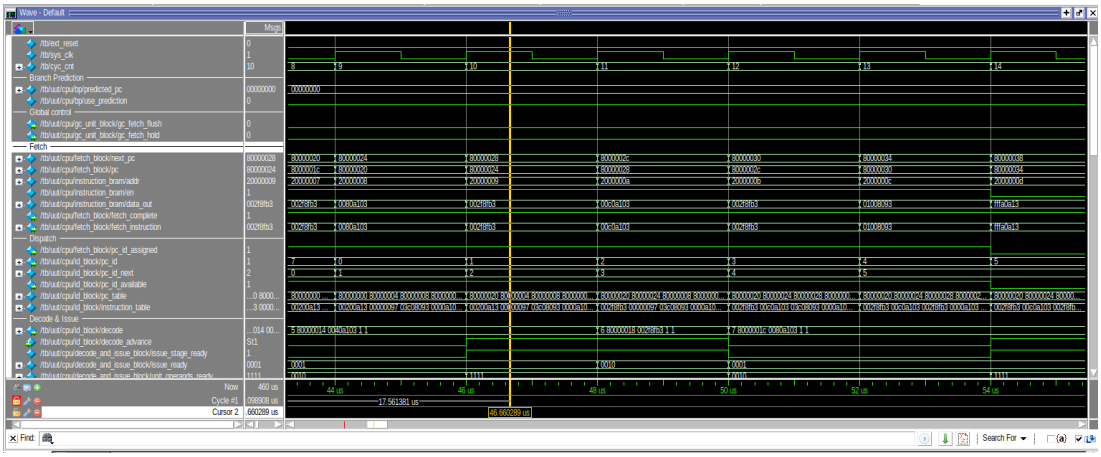


Рисунок 1 – Выполнение задания 2

В задании 3 требовалось по таблице вариантов получить снимок экрана с временной диаграммой стадии декодирования и планирования на выполнение (D) для команды с заданным адресом (и номером итерации, если команда относится к циклу), чтобы увидеть моменты выдачи/задержки декодирования при конфликтах и занятости конвейера. Скриншот в ходе выполнения задания 3 представлен на рисунке 2.

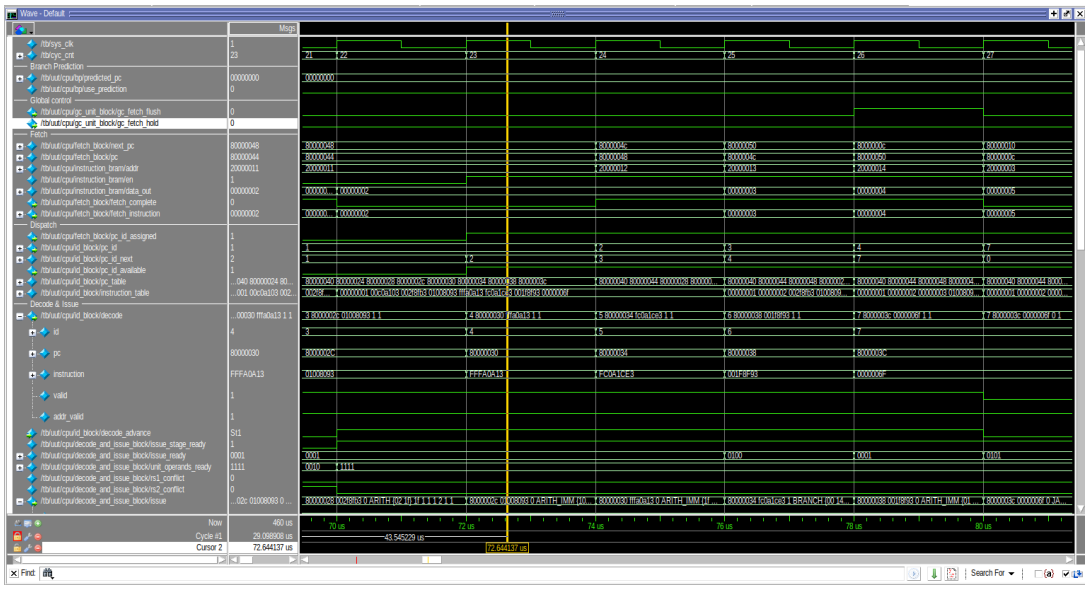


Рисунок 2 – Выполнение задания 3

В задании 4 требовалось, руководствуясь таблицей вариантов, получить временную диаграмму стадии выполнения выбранной команды с заданным адресом (и номером итерации для команд в цикле), то есть отследить фактическое исполнение в соответствующем исполнительном блоке. Скриншот в ходе выполнения задания 4 представлен на рисунке 3.

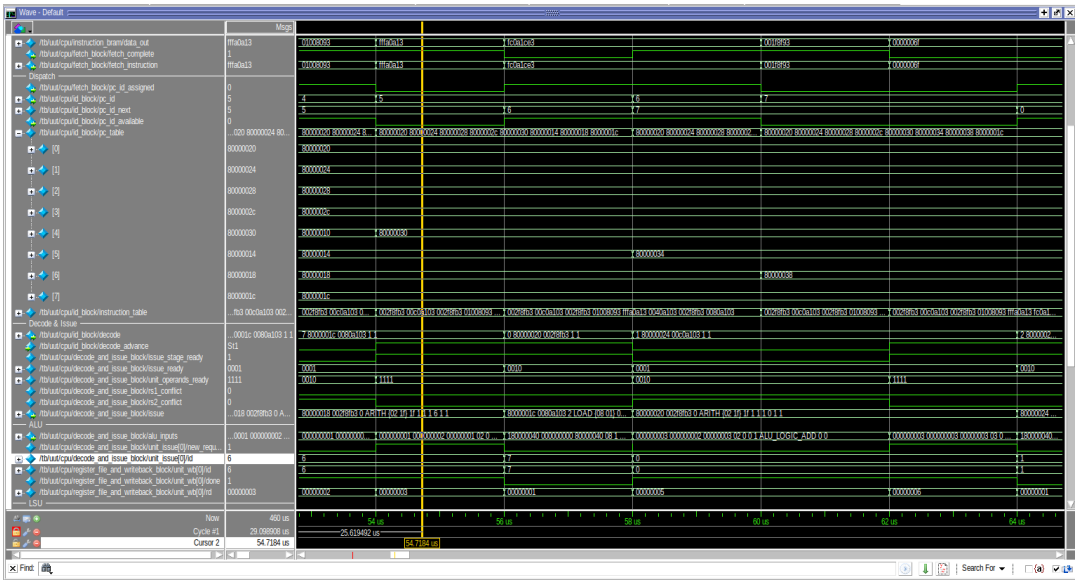
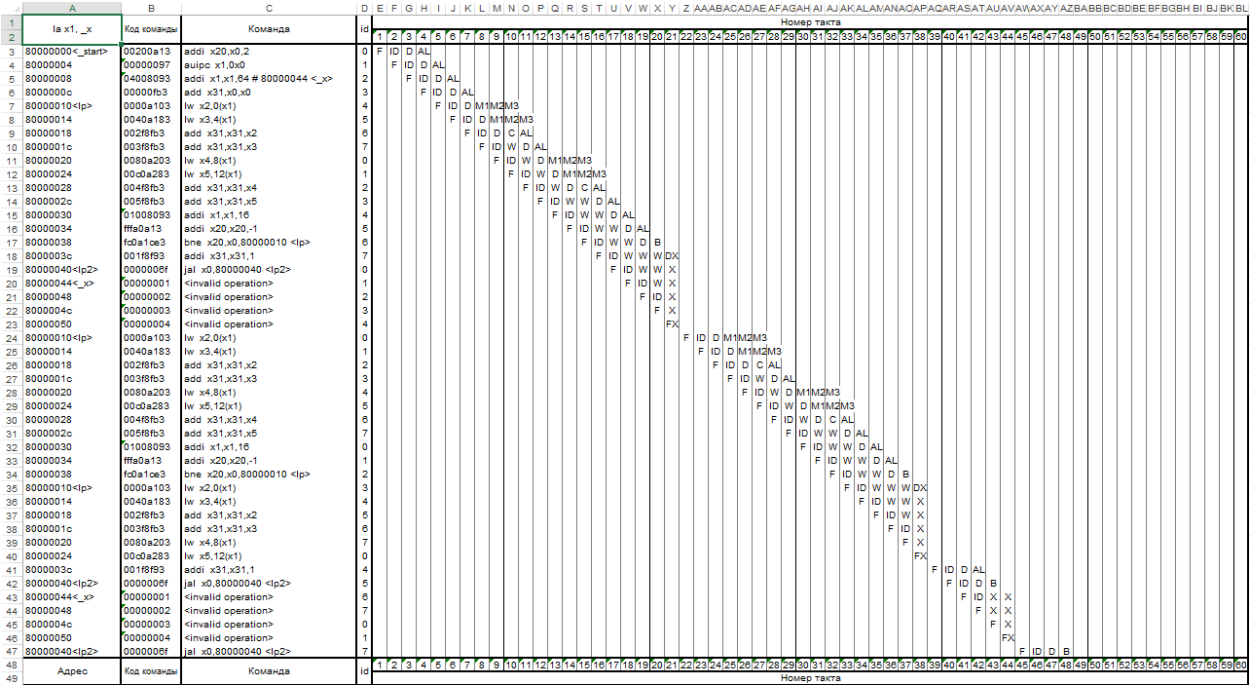


Рисунок 3 – Выполнение задания 4

Трасса работы неоптимизированной программы представлена на рисунке 4.



	Ia x1, _x	Код команды	Команда	Id	Номер такта
3	80000000<_start>	00200a13	addi x20,x0,2	0	F ID D AL
4	80000004	00000097	auipc x1,0x0	1	F ID D AL
5	80000008	04008093	addi x1,x1,64 # 80000044 <_x>	2	F ID D AL
6	8000000c	00000fb3	add x31,x0,x0	3	F ID D AL
7	80000010<lp>	0000a103	lw x2,0(x1)	4	F ID D M1 M2 M3
8	80000014	0040a183	lw x3,4(x1)	5	F ID D M1 M2 M3
9	80000018	002f8fb3	add x31,x31,x2	6	F ID D C AL
10	8000001c	003f8fb3	add x31,x31,x3	7	F ID W D AL
11	80000020	0080a203	lw x4,8(x1)	0	F ID W D M1 M2 M3
12	80000024	00c0a283	lw x5,12(x1)	1	F ID W D M1 M2 M3
13	80000028	004f8fb3	add x31,x31,x4	2	F ID W D C AL
14	8000002c	005f8fb3	add x31,x31,x5	3	F ID W W D AL
15	80000030	01008093	addi x1,x1,16	4	F ID W W D AL
16	80000034	fffa0a13	addi x20,x20,-1	5	F ID W W D AL
17	80000038	fc0a1ce3	bne x20,x0,80000010 <lp>	6	F ID W W D B
18	8000003c	001f8f93	addi x31,x31,1	7	F ID W W W DX
19	80000040<lp2>	0000006f	jal x0,80000040 <lp2>	0	F ID W W X
20	80000044<_x>	00000001	<invalid operation>	1	F ID W X
21	80000048	00000002	<invalid operation>	2	F ID X
22	8000004c	00000003	<invalid operation>	3	F X
23	80000050	00000004	<invalid operation>	4	FX
24	80000010<lp>	0000a103	lw x2,0(x1)	0	F ID D M1 M2 M3
25	80000014	0040a183	lw x3,4(x1)	1	F ID D M1 M2 M3
26	80000018	002f8fb3	add x31,x31,x2	2	F ID D C AL
27	8000001c	003f8fb3	add x31,x31,x3	3	F ID W D AL
28	80000020	0080a203	lw x4,8(x1)	4	F ID W D M1 M2 M3
29	80000024	00c0a283	lw x5,12(x1)	5	F ID W D M1 M2 M3
30	80000028	004f8fb3	add x31,x31,x4	6	F ID W D C AL
31	8000002c	005f8fb3	add x31,x31,x5	7	F ID W W D AL
32	80000030	01008093	addi x1,x1,16	0	F ID W W D AL
33	80000034	fffa0a13	addi x20,x20,-1	1	F ID W W D AL
34	80000038	fc0a1ce3	bne x20,x0,80000010 <lp>	2	F ID W W D B
35	80000010<lp>	0000a103	lw x2,0(x1)	3	F ID W W W DX
36	80000014	0040a183	lw x3,4(x1)	4	F ID W W X
37	80000018	002f8fb3	add x31,x31,x2	5	F ID W X
38	8000001c	003f8fb3	add x31,x31,x3	6	F ID X
39	80000020	0080a203	lw x4,8(x1)	7	F X
40	80000024	00c0a283	lw x5,12(x1)	0	FX
41	8000003c	001f8f93	addi x31,x31,1	4	F ID D AL
42	80000040<lp2>	0000006f	jal x0,80000040 <lp2>	5	F ID D B
43	80000044<_x>	00000001	<invalid operation>	6	F ID X X
44	80000048	00000002	<invalid operation>	7	F X X
45	8000004c	00000003	<invalid operation>	0	F X
46	80000050	00000004	<invalid operation>	1	FX
47	80000040<lp2>	0000006f	jal x0,80000040 <lp2>	7	F ID D B
48	Адрес	Код команды	Команда	Id	Номер такта

Рисунок 4 – Трасса неоптимизированной программы

В исходной версии после команды загрузки lw x2, 0(x1) сразу выполняется add x31, x31, x2. В конвейерном ядре это приводит к зависимости и, как следствие, к вынужденным простоям.

Для оптимизации следует выполнить перестановку инструкций так, чтобы увеличить расстояние между lw и первой add. Практически это достигается переносом второй загрузки lw x3, 4(x1) и инкремента указателя addi x1, x1, ... между загрузками и операциями суммирования.

Код оптимизированный программы представлен в листинге 4.

Листинг 4 – Код оптимизированной программы

```
riscv64-linux-gnu-as --march=rv32i 10.s -o 10.o
riscv64-linux-gnu-ld -b elf32-littleriscv -T link.ld 10.o -o
10.elf
riscv64-linux-gnu-objdump -D -M numeric,no-aliases -t 10.elf

10.elf:     file format elf32-littleriscv

SYMBOL TABLE:
80000000 l    d  .text   00000000 .text
80000044 l    d  .data   00000000 .data
00000000 l    df *ABS*   00000000 10.o
00000008 l       *ABS*   00000000 len
00000004 l       *ABS*   00000000 enroll
00000004 l       *ABS*   00000000 elem_sz
80000044 l       .data   00000000 _x
80000010 l       .text   00000000 lp
80000040 l       .text   00000000 lp2
80000000 g       .text   00000000 _start
80000064 g       .data   00000000 _end

Disassembly of section .text:

80000000 <_start>:
80000000: 00200a13           addi x20,x0,2
80000004: 00000097           auipc     x1,0x0
80000008: 04008093           addi x1,x1,64 # 80000044 <_x>
8000000c: 00000fb3           add  x31,x0,x0

80000010 <lp>:
80000010: 0000a103           lw   x2,0(x1)
80000014: 0040a183           lw   x3,4(x1)
80000018: 0080a203           lw   x4,8(x1)
8000001c: 00c0a283           lw   x5,12(x1)
80000020: 002f8fb3           add  x31,x31,x2
80000024: 003f8fb3           add  x31,x31,x3
80000028: 004f8fb3           add  x31,x31,x4
```

```
8000002c: 005f8fb3            add  x31,x31,x5
80000030: 01008093            addi x1,x1,16
80000034: fffa0a13            addi x20,x20,-1
80000038: fc0a1ce3            bne  x20,x0,80000010 <lp>
8000003c: 001f8f93            addi x31,x31,1

80000040 <lp2>:
80000040: 0000006f            jal  x0,80000040 <lp2>

Disassembly of section .data:

80000044 <_x>:
80000044: 0001                     .insn     2, 0x0001
80000046: 0000                     .insn     2, 0x
80000048: 0002                     .insn     2, 0x0002
8000004a: 0000                     .insn     2, 0x
8000004c: 00000003            lb   x0,0(x0) # 0 <elem_sz-0x4>
80000050: 0004                     .insn     2, 0x0004
80000052: 0000                     .insn     2, 0x
80000054: 0005                     .insn     2, 0x0005
80000056: 0000                     .insn     2, 0x
80000058: 0006                     .insn     2, 0x0006
8000005a: 0000                     .insn     2, 0x
8000005c: 00000007            .insn     4, 0x0007
80000060: 0008                     .insn     2, 0x0008
     ...
riscv64-linux-gnu-objcopy -O binary --reverse-bytes=4 10.elf
10.bin
xxd -g 4 -c 4 -p 10.bin 10.hex
rm 10.o 10.bin 10.elf
```

Трасса работы оптимизированной программы представлена на рисунке 5.

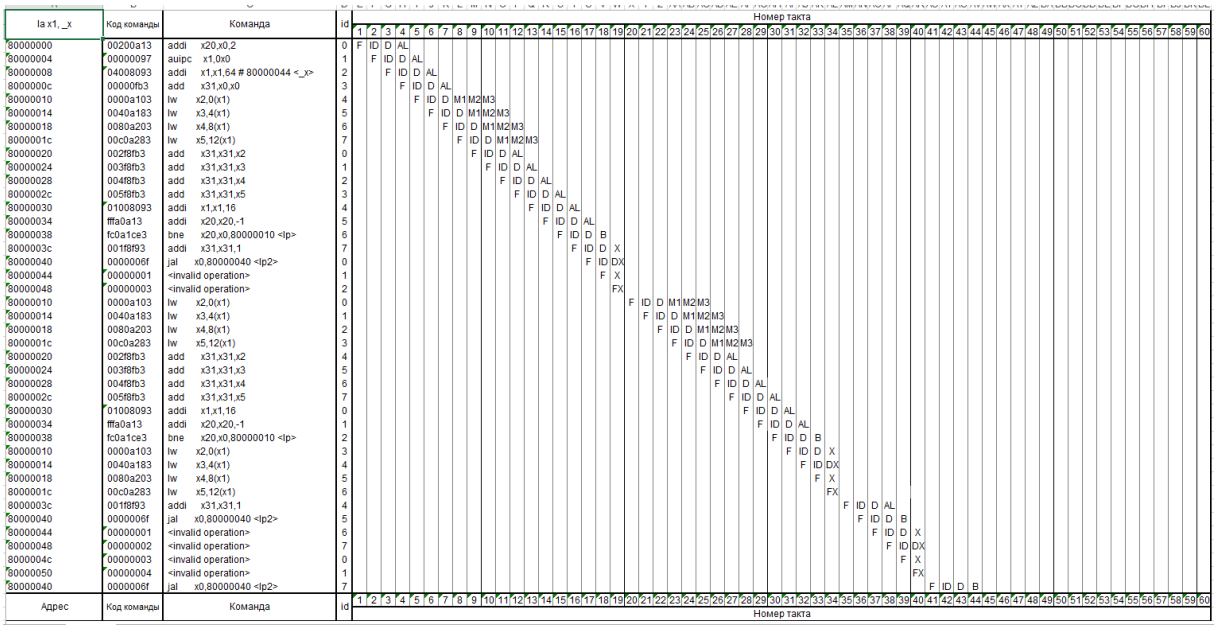


Рисунок 5 – Трасса оптимизированной программы

За счет оптимизации удалось ускорить программу на 4 такта.

Вывод

В ходе лабораторной работы были изучены принципы выполнения программ на ядре Taiga (RV32I) и влияние конвейерной организации на временные характеристики исполнения: по временным диаграммам в Modelsim были проанализированы прохождение инструкций по стадиям, появление задержек из-за зависимостей по данным и особенности выполнения команд доступа к памяти и ветвлений. Также на примере суммирования элементов массива была рассмотрена оптимизация на уровне ассемблера за счёт планирования инструкций.