|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 2 |

**Название:**

Изучение принципов работы микропроцессорного ядра RISC-V

**Дисциплина:** Архитектура ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-56Б |  | 29.10.2021 | В.В. Леонов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.Ю. Попов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

# 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.

# Задание №1

Исходный текст исследуемой программы **test.s**.

.section .text

.globl \_start;

len = 8 ;Arr len

enroll = 4 ;Items per iteration

elem\_sz = 4 ;Size of one elem

\_start:

addi x20, x0, len/enroll ;Cycle count

la x1, \_x ;Begin of arr

loop:

lw x2, 0(x1) ;x2 = x1[0];

**add** x31, x31, x2 ;x31 += x2;

lw x2, 4(x1) ;x2 = x1[1];

**add** x31, x31, x2 ;x1 += x2;

lw x2, 8(x1) ;x2 += x1[2];

**add** x31, x31, x2 ;x31 += x2;

lw x2, 12(x1) ;x2 = x1[3];

**add** x31, x31, x2 ;x31 += x2;

addi x1, x1, elem\_sz\*enroll ;Update ptr x1: x1 += items per iter

addi x20, x20, -1 ;Cycle decrement

bne x20, x0, **loop** ;Jump to label loop if x20 != x0

addi x31, x31, 1 ;x1 += 1

forever: j forever ;Forever loop

.section .data ;Description of data in arr

\_x: .4byte 0x1

.4byte 0x2

.4byte 0x3

.4byte 0x4

.4byte 0x5

.4byte 0x6

.4byte 0x7

.4byte 0x8

Дизассемблерный листинг исследуемой программы **test.s**.

SYMBOL TABLE:

80000000 l d .text 00000000 .text

80000040 l d .data 00000000 .data

00000000 l df \*ABS\* 00000000 **test**.o

00000008 l \*ABS\* 00000000 len

00000004 l \*ABS\* 00000000 enroll

00000004 l \*ABS\* 00000000 elem\_sz

80000040 l .data 00000000 \_x

8000000c l .text 00000000 **loop**

8000003c l .text 00000000 forever

80000000 g .text 00000000 \_start

80000060 g .data 00000000 \_end

Disassembly of section .text:

80000000 <\_start>:

80000000: 00200a13 addi x20,x0,2

80000004: 00000097 auipc x1,0x0

80000008: 03c08093 addi x1,x1,60 # 80000040 <\_x>

8000000c <**loop**>:

8000000c: 0000a103 lw x2,0(x1)

80000010: 002f8fb3 **add** x31,x31,x2

80000014: 0040a103 lw x2,4(x1)

80000018: 002f8fb3 **add** x31,x31,x2

8000001c: 0080a103 lw x2,8(x1)

80000020: 002f8fb3 **add** x31,x31,x2

80000024: 00c0a103 lw x2,12(x1)

80000028: 002f8fb3 **add** x31,x31,x2

8000002c: 01008093 addi x1,x1,16

80000030: fffa0a13 addi x20,x20,-1

80000034: fc0a1ce3 bne x20,x0,8000000c <**loop**>

80000038: 001f8f93 addi x31,x31,1

8000003c <forever>:

8000003c: 0000006f jal x0,8000003c <forever>

Disassembly of section .data:

80000040 <\_x>:

80000040: 0001 c.addi x0,0

80000042: 0000 unimp

80000044: 0002 0x2

80000046: 0000 unimp

80000048: 00000003 lb x0,0(x0) # 0 <elem\_sz-0x4>

8000004c: 0004 c.addi4spn x9,x2,0

8000004e: 0000 unimp

80000050: 0005 c.addi x0,1

80000052: 0000 unimp

80000054: 0006 0x6

80000056: 0000 unimp

80000058: 00000007 0x7

8000005c: 0008 c.addi4spn x10,x2,0

...

Псевдокод, поясняющий работу программы **test.s**.

#define len 8

#define enroll 4

#define elem\_sz 4

**int** \_x[]={1,2,3,4,5,6,7,8};

**void** \_start() {

**int** x20 = len/enroll;

**int** \*x1 = \_x;

**do** {

**int** x2 = x1[0];

x31 += x2;

x2 = x1[1];

x31 += x2;

x2 = x1[2];

x31 += x2;

x2 = x1[3];

x31 += x2;

x1 += enroll;

x20--;

} **while**(x20 != 0);

x31++;

**while**(1){}

}

Рассмотрим задание по варианту. Исходный текст исследуемой программы **var61\_8.s** (по варианту).

.section .text

.globl \_start;

len = 8

enroll = 2

elem\_sz = 4

\_start:

la x1, \_x

addi x20, x1, elem\_sz\*len

lp:

lw x2, 0(x1)

lw x3, 4(x1)

addi x1, x1, elem\_sz\*enroll

**add** x31, x31, x2 #!

**add** x31, x31, x3

bne x1, x20, 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

Дизассемблерный листинг исследуемой программы **var61\_8.s** (по варианту).

80000000 <\_start>:

80000000: 00000097 auipc x1,0x0

80000004: 02c08093 addi x1,x1,44 # 8000002c <\_x>

80000008: 02008a13 addi x20,x1,32

8000000c <lp>:

8000000c: 0000a103 lw x2,0(x1)

80000010: 0040a183 lw x3,4(x1)

80000014: 00808093 addi x1,x1,8

80000018: 002f8fb3 **add** x31,x31,x2

8000001c: 003f8fb3 **add** x31,x31,x3

80000020: ff4096e3 bne x1,x20,8000000c <lp>

80000024: 001f8f93 addi x31,x31,1

80000028 <lp2>:

80000028: 0000006f jal x0,80000028 <lp2>

Disassembly of section .data:

8000002c <\_x>:

8000002c: 0001 c.addi x0,0

8000002e: 0000 unimp

80000030: 0002 0x2

80000032: 0000 unimp

80000034: 00000003 lb x0,0(x0) # 0 <enroll-0x2>

80000038: 0004 c.addi4spn x9,x2,0

8000003a: 0000 unimp

8000003c: 0005 c.addi x0,1

8000003e: 0000 unimp

80000040: 0006 0x6

80000042: 0000 unimp

80000044: 00000007 0x7

80000048: 0008 c.addi4spn x10,x2,0

...

Псевдокод, поясняющий работу программы **var61\_8.s** (по варианту).

#define len 8

#define enroll 2

#define elem\_sz 4

**int** \_x[]={1,2,3,4,5,6,7,8};

**void** \_start() {

**int** \*x1 = \_x;

**int** x20 = x1 + elem\_size \* len;

**do** {

**int** x2 = x1[0];

**int** x3 = x1[2];

x1 += elem\_sz \* enroll;

x31 += x2;

x31 += x3;

} **while**(x1 != x20);

x31++;

**while**(1){}

}

В регистре x31 по итогу выполнения программы должно содержаться значение, равное сумме всех элементов массива, увеличенное на 1, то есть 37.

Запустим симуляцию тестовой программы в среде Modelsim. Изучив временные диаграммы, выполним задания №2 - №4.

# Задание №2

Адрес команды по варианту: 80000028.

Итерация команды по варианту: 1.

На рисунке представлен снимок экрана, содержащий временную диаграмму выполнения стадий выборки и диспетчеризации команды с указанным адресом.

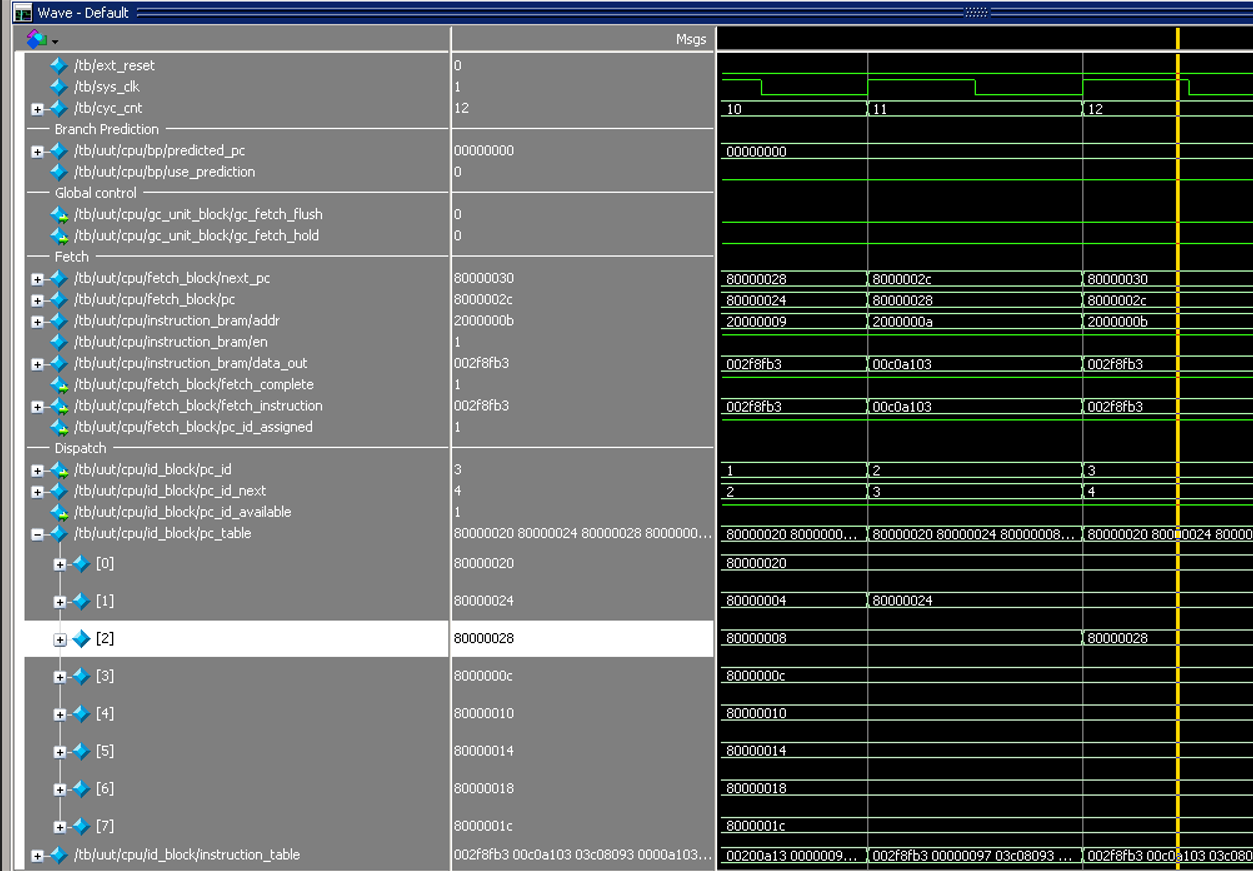


Рисунок 1. Выборка и диспетчеризация команды по варианту

Команда была выбрана в такте 11, а диспетчеризована в такте 12 (сигнал fetch\_complete = 1 в 12 такте, выборка команды с идентификатором 2 (см.pc\_table), т.е. команды по варианту).

# Задание №3

Адрес команды по варианту: 80000034.

Итерация команды по варианту: 1.

На рисунках представлены снимки экрана, содержащие временную диаграмму выполнения стадий выборки, диспетчеризации, декодирования и планирования на выполнение команды с указанным адресом.

В 15 такте произошла выборка команда, в 16 - диспетчеризация.

В такте 24 произошло декодирование команды, в 25 - планирование на выполнение (rs1\_conflict и rs2\_conflict равны 0, значит конфликта нет, идентификатор нужной команды равен 5, сигнал stage\_valid = 1 и id = 5 в 25 такте).

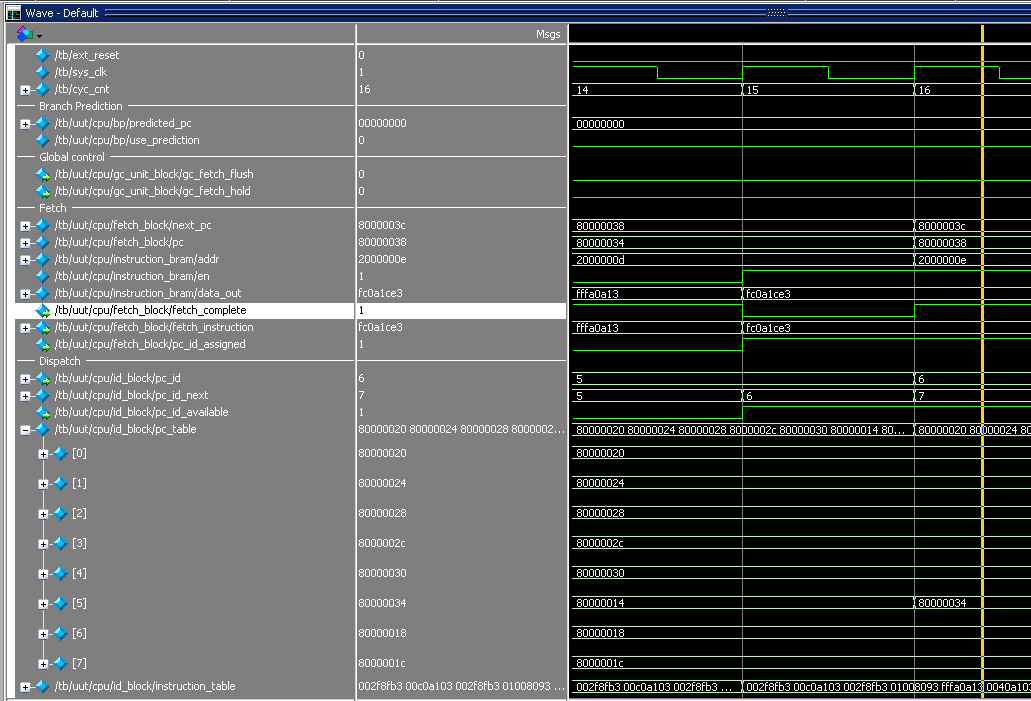


Рисунок 2. Выборка и диспетчеризация команды

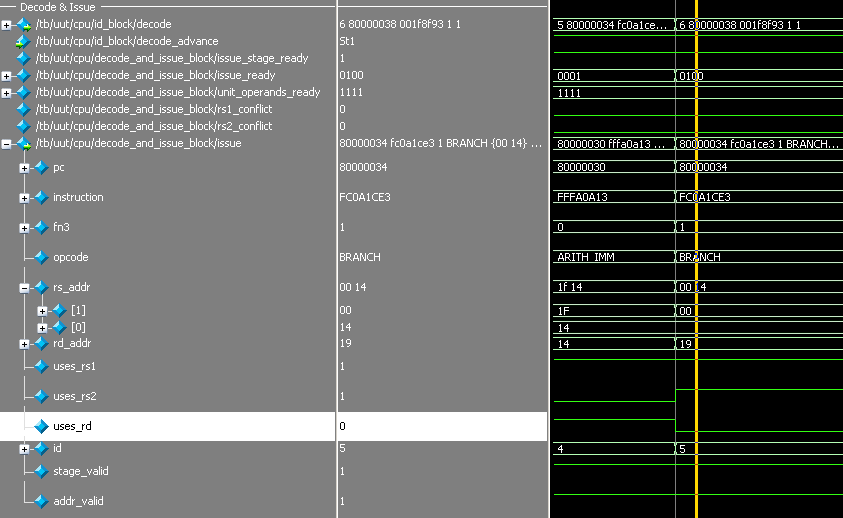


Рисунок 3. Декодирование команды и планирование на выполнение

# Задание №4

Адрес команды по варианту: 8000001с.

Итерация команды по варианту: 1.

На рисунках представлены снимки экрана, содержащие временную диаграмму выполнения стадии выполнения, диспетчеризации, декодирования и выполнения команды с указанным адресом.

В такте 8 команда была выбрана, в такте 9 диспетчеризована.

В такте 14 команда была декодирована, в такте 15 была запланирована на выполнение.

В тактах 15, 16 и 17 команда выполнялась блоком LSU (обращение к памяти).

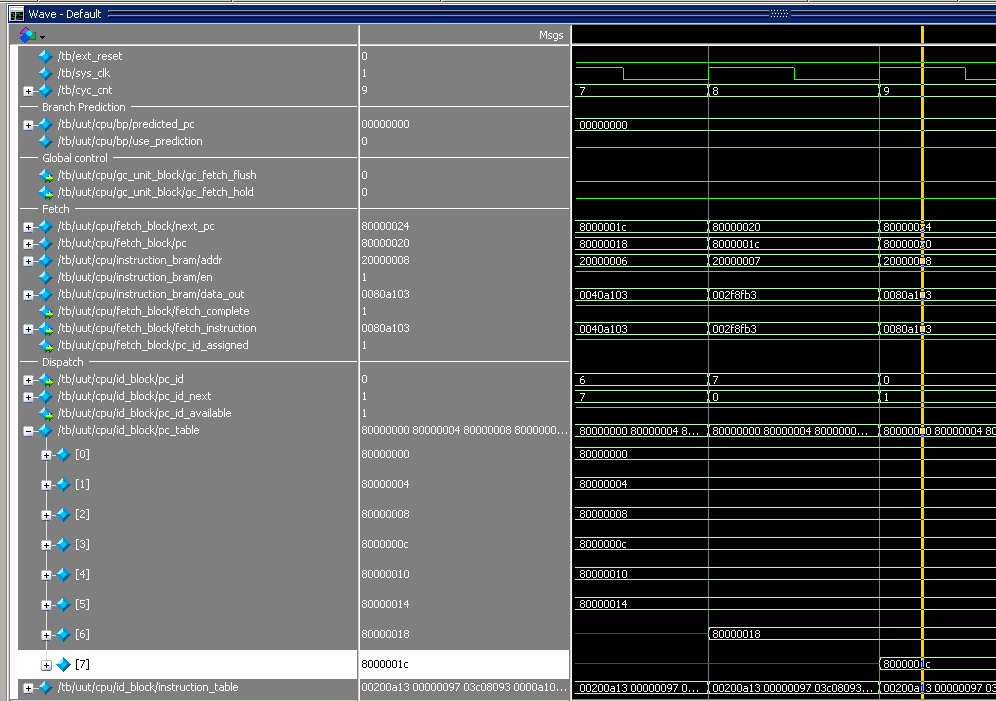


Рисунок 4. Выборка и диспетчеризация команды



Рисунок 5. Декодирование команды и планирование на выполнение

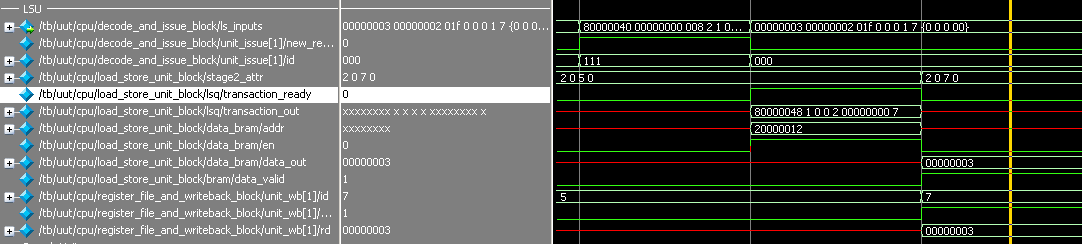


Рисунок 6. Выполнение

# Задание №5

Трассы первоначальной программы и оптимизированной программы приведены в приложении.

Временные диаграммы (в виде скриншотов окна Modelsim) сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом **#!**, приведены в приложении.

Значение в регистре х31 на момент окончания выполнения программы совпал с ожидаемым ().



Рисунок 7. Результат программы

# Вывод

Программа по варианту написана таким образом, что в ней не возникает конфликтов, однако ценой отсутствия конфликтов является ее сравнительно долгое выполнение. Программа обрабатывает массив из 8 чисел по 2 числа за итерацию, было решено оптимизировать программу таким образом, чтобы по возможности тоже избежать конфликтов, однако обрабатывать по 4 элемента за итерацию, таким образом количество итераций сократится с 4 до 2. Для оптимизации в цикле программы необходимо сначала выполнить обращения к памяти с целью получения элементов массива, и только потом произвести операции сложения. Таким образом, в этой программе тоже не будет конфликтов, так как между операцией доступа к памяти и операцией сложения для соответствующего элемента массива будут еще как минимум 3 операции(например, для первой переменной, между операцией обращения к памяти и операцией сложения будут находится еще 3 другие операции доступа к памяти для остальных элементов), а обращение к памяти выполняется именно 3 такта. Таким образом, удалось сократить количество тактов с 42 до 37.

# Приложение

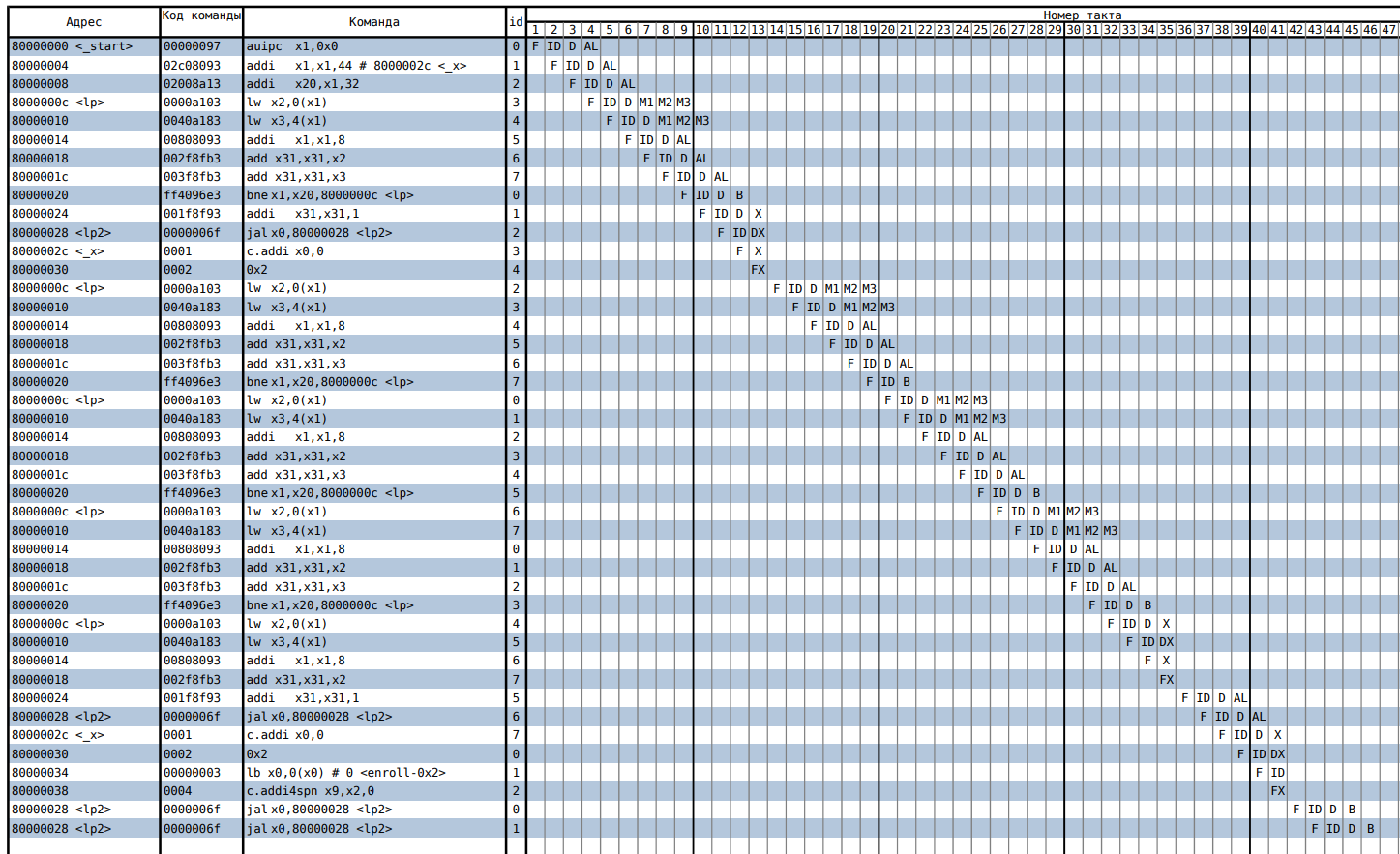


Рисунок 8. Трасса первоначальной программы

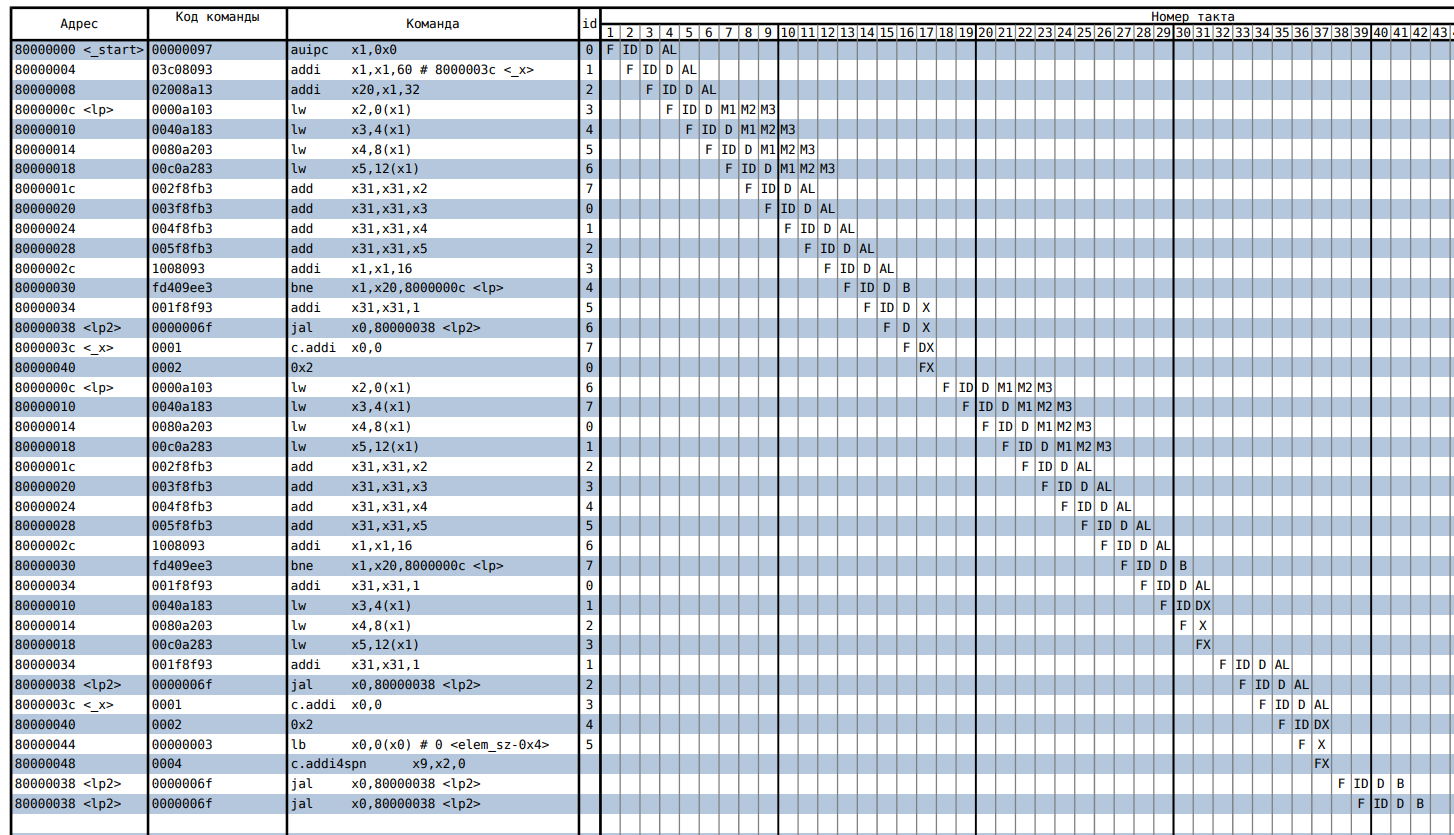


Рисунок 9. Трасса оптимизированной программы

На следующих рисунках представлены временные диаграммы (в виде скриншотов окна Modelsim) сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом **#!**.

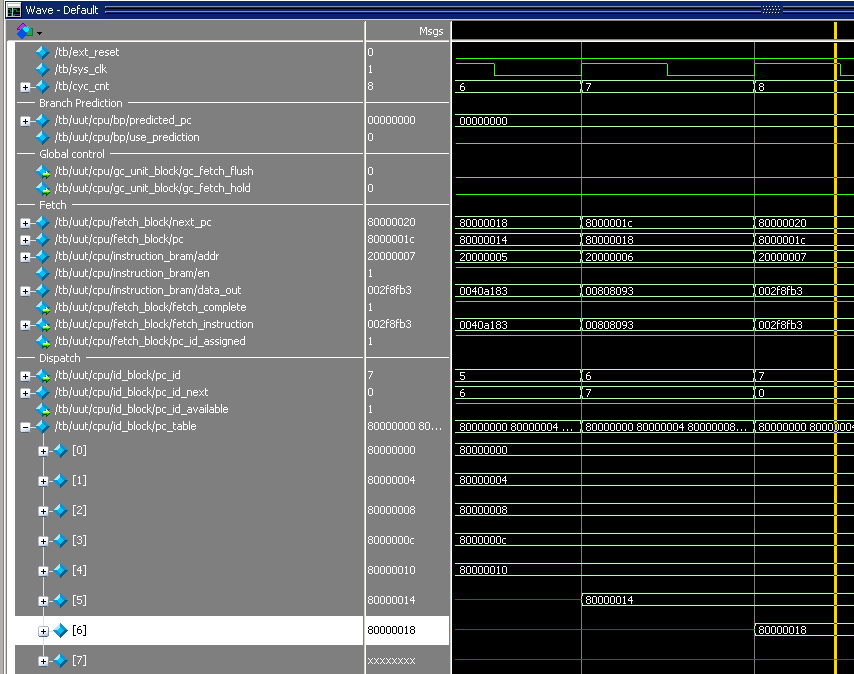


Рисунок 10. Выборка (7 такт) и диспетчеризация (8 такт) команды на первой итерации

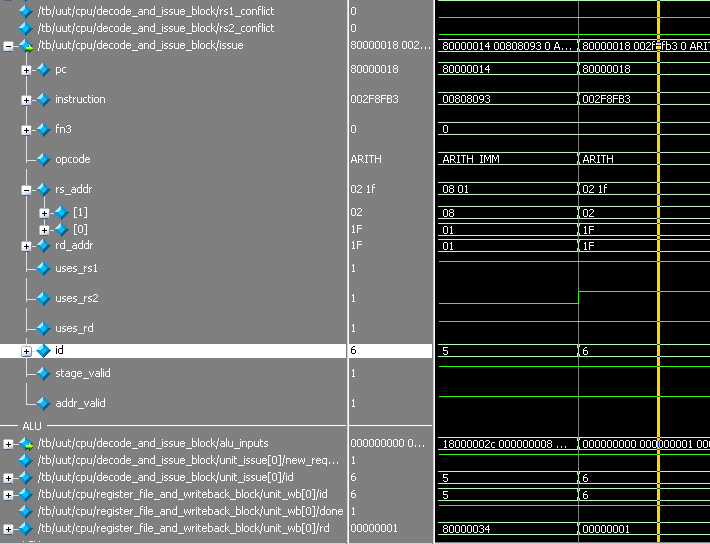


Рисунок 11. Декодирование (9 такт) и выполнение (10 такт) команды на первой итерации

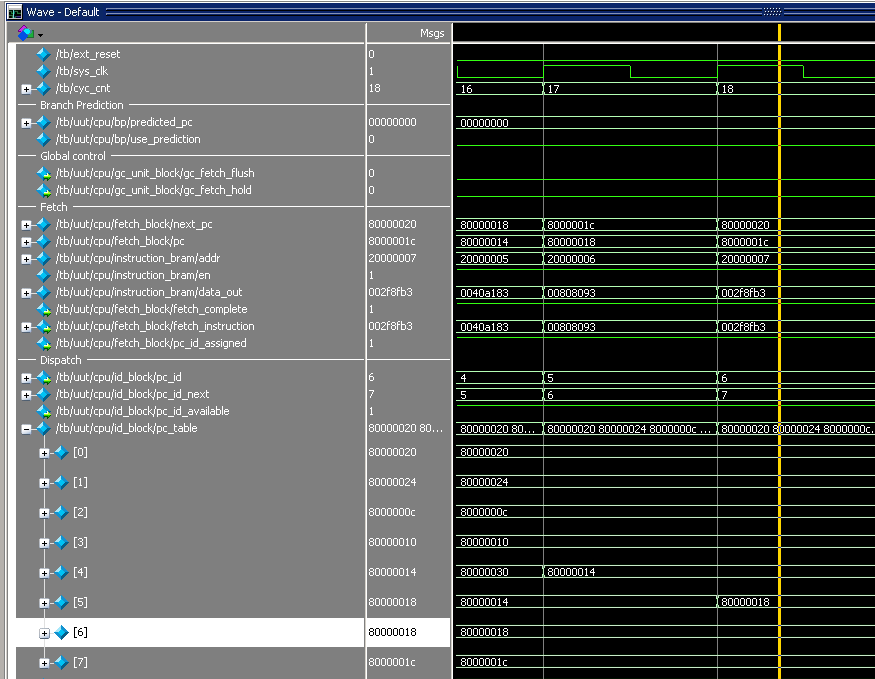


Рисунок 12. Выборка (17 такт) и диспетчеризация (18 такт) команды на второй итерации

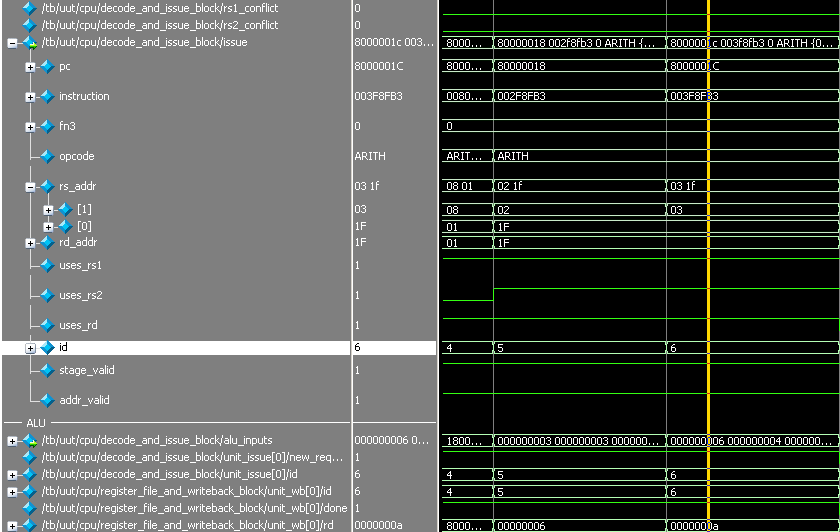


Рисунок 13. Декодирование (20 такт) и выполнение (21 такт) команды на второй итерации

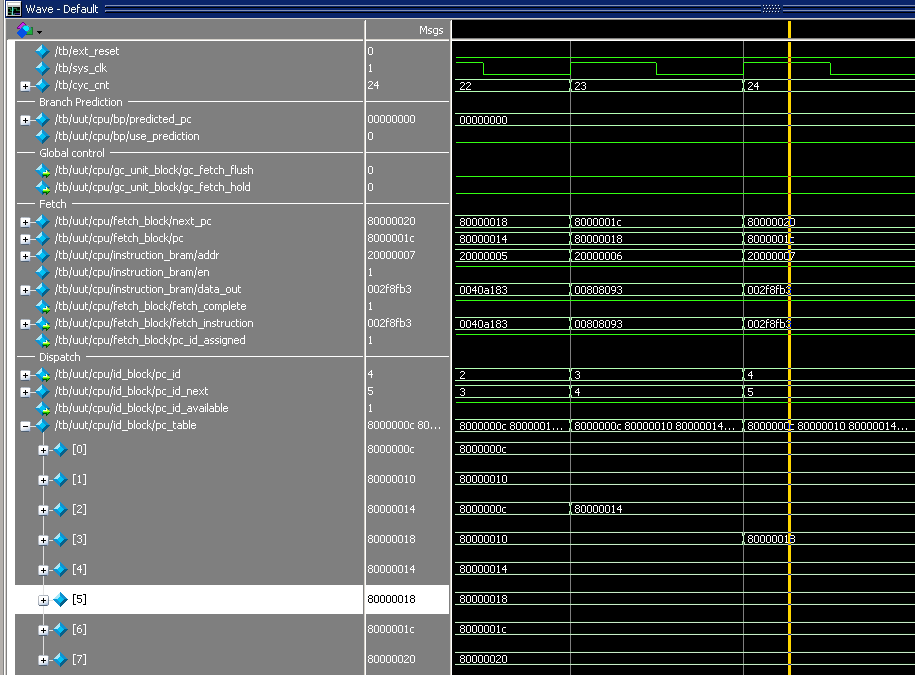


Рисунок 14. Выборка (23 такт) и диспетчеризация (24 такт) команды на третьей итерации

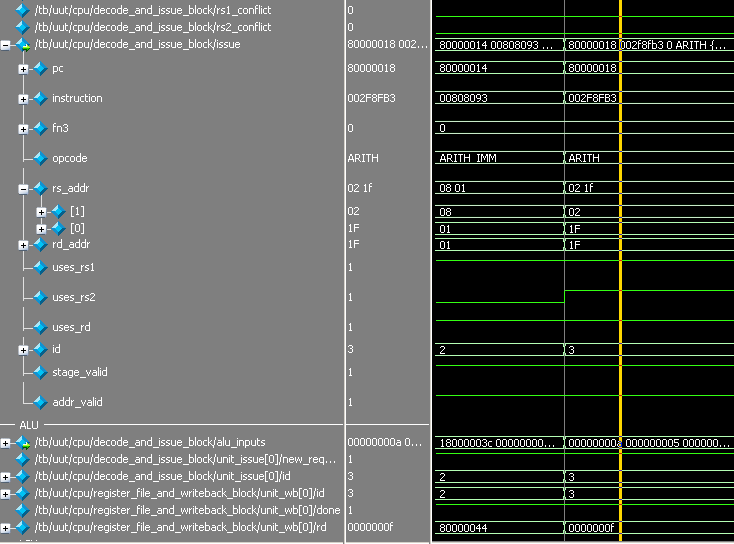


Рисунок 15. Декодирование (25 такт) и выполнение (26 такт) команды на третьей итерации

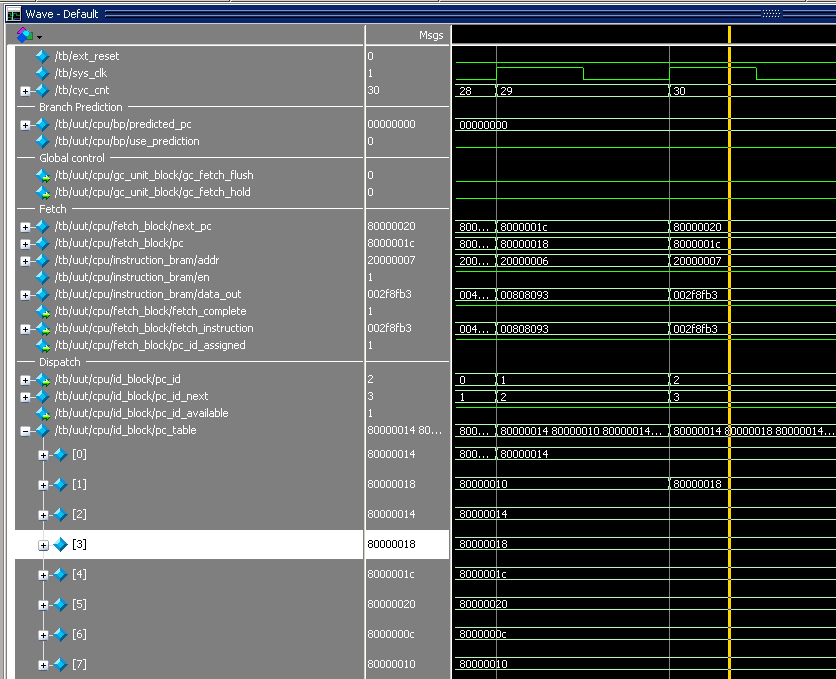


Рисунок 16. Выборка (29 такт) и диспетчеризация (30 такт) команды на четвертой итерации

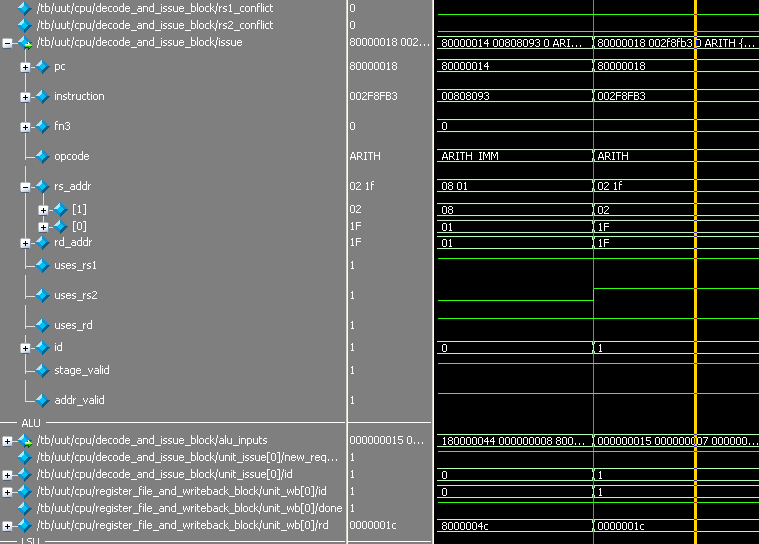


Рисунок 17. Декодирование (31 такт) и выполнение (32 такт) команды на четвертой итерации

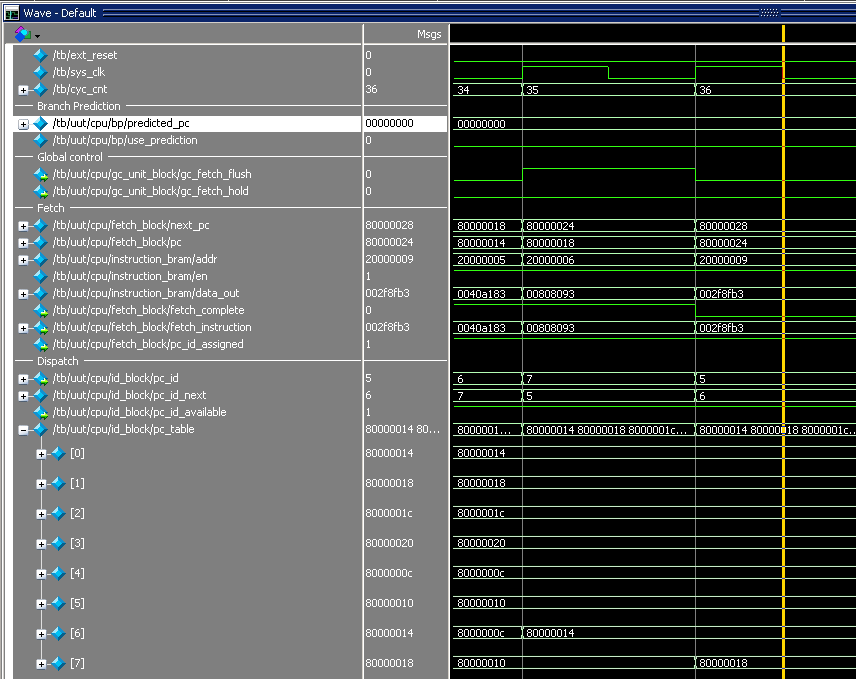


Рисунок 18. Сброс выборки (35 такт)