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 **09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА**

О Т Ч Е Т

по лабораторной работе № 1

Дисциплина: Организация ЭВМ и систем

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Студент гр. ИУ6-72Б _____ И.С. Марчук
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

Задание 1

Собрать программу, перевести её в псевдокод.

Код на языке ассемблера:

```
        .section .text
        .globl _start;
len = 9 #Размер массива
enroll = 2 #Количество обрабатываемых элементов за одну итерацию
elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x0, (len-1)/enroll
        lw x31, 0(x1)
        addi x1, x1, elem_sz*1
lp:     lw x2, 0(x1)
        lw x3, 4(x1)
        bltu x2, x31, lt1
        add x31, x0, x2 #!
lt1:    bltu x3, x31, lt2
        add x31, x0, x3
lt2:    add x1, x1, elem_sz*enroll
        addi x20, x20, - 1
        bne x20, x0, lp
lp2:    j lp2

        .section .data
_x:     .4byte 0x1
        .4byte 0x2
        .4byte 0x3
        .4byte 0x4
        .4byte 0x5
        .4byte 0x6
        .4byte 0x7
        .4byte 0x8
        .4byte 0x9
```

Код на языке Си:

```
#include <stdio.h>

#define len 9
#define enroll 2
#define elem_sz 4
int _x[] = {1,2,3,4,5,6,7,8,9};
int x0 = 0;

int main(){
     int *x1 = _x;
     int x20 = (len - 1)/enroll;
     int x31 = x1[0];                    …., x2 = 0, x3 = 0;
     *x1 = *x1 + (elem_sz*1);

     while(x20 != 0){
          x2 = x1[0];
          x3 = x1[1];
          if (x2 > x31){
               if (!(x3 > x31)){
                     x31 = x3; // если x3 < x31
               }
          }else{
                x31 = x2; // если x2 < x31
          }
         *x1 = *x1 + (elem_sz * enroll);
         x20 = x20 - 1;
    }

    printf("%d", x31);// поиск наименьшего
    return 0;
}
```

В конце выполнения программы в регистре x31 будет значение 1.

Код дизассемблера:

```
80000000 <_start>:
80000000:       00000097            auipc   x1,0x0
80000004:       03c08093            addi    x1,x1,60 # 8000003c <_x>
80000008:       00400a13            addi    x20,x0,4
8000000c:       0000af83            lw      x31,0(x1)
80000010:       00408093            addi    x1,x1,4

80000014 <lp>:
80000014:       0000a103            lw      x2,0(x1)
80000018:       0040a183            lw      x3,4(x1)
8000001c:       01f16463            bltu    x2,x31,80000024 <lt1>
80000020:       00200fb3            add     x31,x0,x2
80000024 <lt1>:
80000024:       01f1e463            bltu    x3,x31,8000002c <lt2>
80000028:       00300fb3            add     x31,x0,x3

8000002c <lt2>:
8000002c:       00808093            addi    x1,x1,8
80000030:       fffa0a13            addi    x20,x20,-1
80000034:       fe0a10e3            bne     x20,x0,80000014 <lp>

80000038 <lp2>:
80000038:       0000006f            jal     x0,80000038 <lp2>
```

Задание 2

Получить снимок экрана, содержащий временную диаграмму выполнения стадий выборки (итерация 2) и диспетчеризации команды с указанным адресом (80000028).

Снимок экрана представлен рисунком 1.

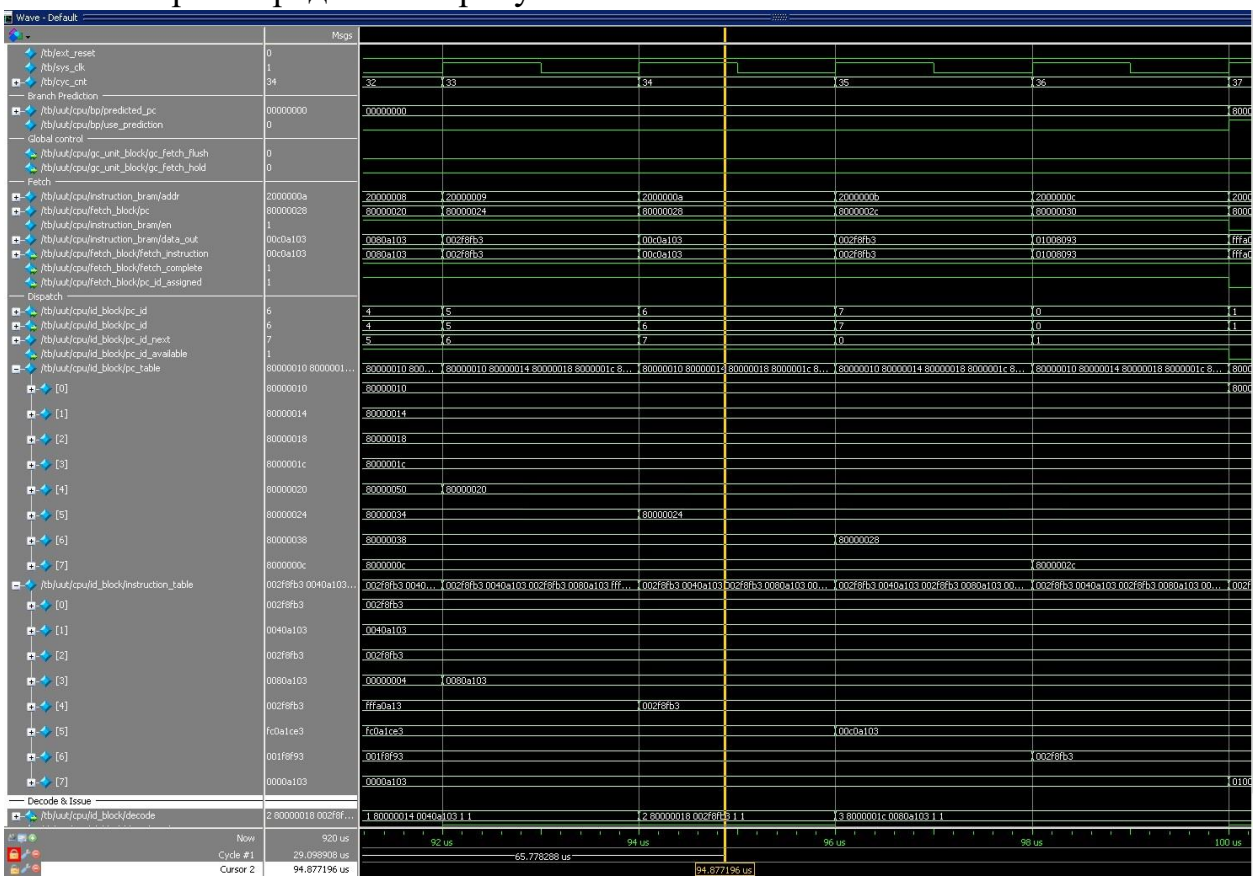

Рисунок 1 — Результат выполнения задания 2.

Сигнал `pc_id_available` равен 1, что подтверждает готовность блока управления метаданными принять результат выборки. `pc=0x80000000`. Адрес, соответствующий данному значению, а именно `0x2000000a` выставляется на ША (сигнал `addr`).

Выставление сигнала `en` разрешает работу памяти команд. Одновременно с этим выставляется сигнал `pc_id_assigned`, указывающий блоку управления метаинформацией, что запрос в память отправлен, и информация о текущем `pc` должна быть записана в очередь команд. Текущее значение `pc` выдается в блоку управления метаинформацией через сигнал `if_pc`. В начале следующего такта это значение (80000028) будет записано в

`pc_table` по индексу, равному значению текущего первого свободного id (в нашем случае 6), а `pc_id` увеличится на 1. Это означает, что команде, которая будет выбрана по адресу
80000000, присвоен `id=0`

В следующем такте (2) данные по адресу `80000028` (00c0a103) выставляются памятью команда (сигнал `data_out`). Эти данные выдаются из блока выборки (сигнал `fetch_instruction`), при этом сигнал `fetch_complete` равен 1. Операция выборки данной команды завершается в такте 34, однако значение данных подается на вход блока управления метаинформацией только в такте 35.

По фронту, завершающему такт 35, выбраный код команды (то есть, сигнал `fetch_instruction`) записывается в таблицу `instruction_table` по индексу 5. Это завершает операцию диспетчеризации.

Задание 3

Получить снимок экрана, содержащий временную диаграмму выполнения стадии декодирования и планирования (итерация 2) на выполнение команды с указанным адресом(80000034).

Снимок экрана представлен рисунком 2.

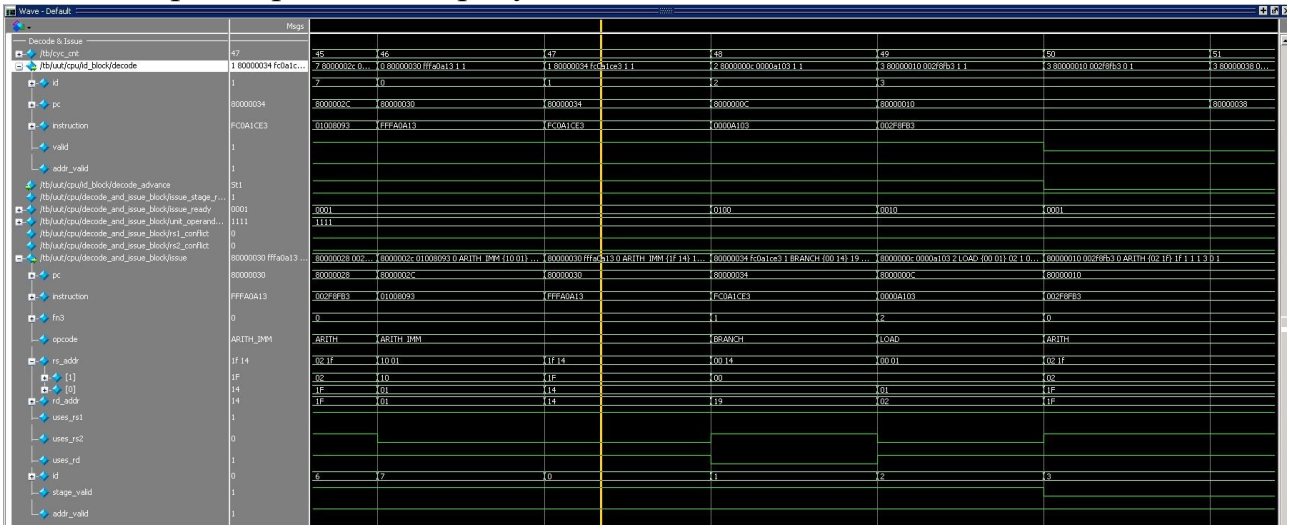

Рисунок 2 — Результат выполнения задания 3.

В такте 48 issue.pc = 80000034, issue.stage_valid = 1,

unit_issue[0].new_request = 1, значит в такте 47 команда декодировалась и запланировалась на выполнение.

Задание 4

Получить снимок экрана, содержащий временную диаграмму выполнения стадии выполнения команды(итерация 2) с указанным адресом(80000020).

Снимок экрана представлен рисунком 3.

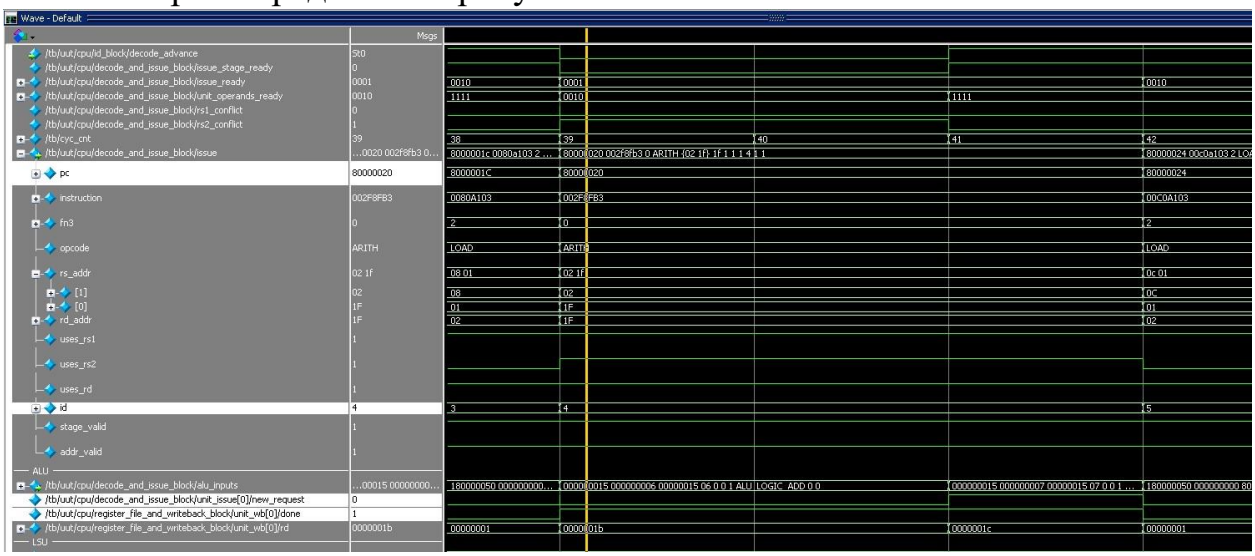

Рисунок 3 — Результат выполнения задания 4.

Для блока АЛУ признаком, служащим для подтверждения готовности результата является сочетание сигналов `unit_wb[0].done` и `unit_issue[0]/new_request` такое произошло в такте 41.

Задание 5

Получить снимок экрана, содержащий временные диаграммы сигналов, соответствующих всем стадиям выполнения команды, обозначенной в тексте программы символом # !

Команда - add x31, x0, x2

Дизассемблер: 80000020: 00200fb3 add x31,x0,x2

Снимок экрана представлен рисунком 4.

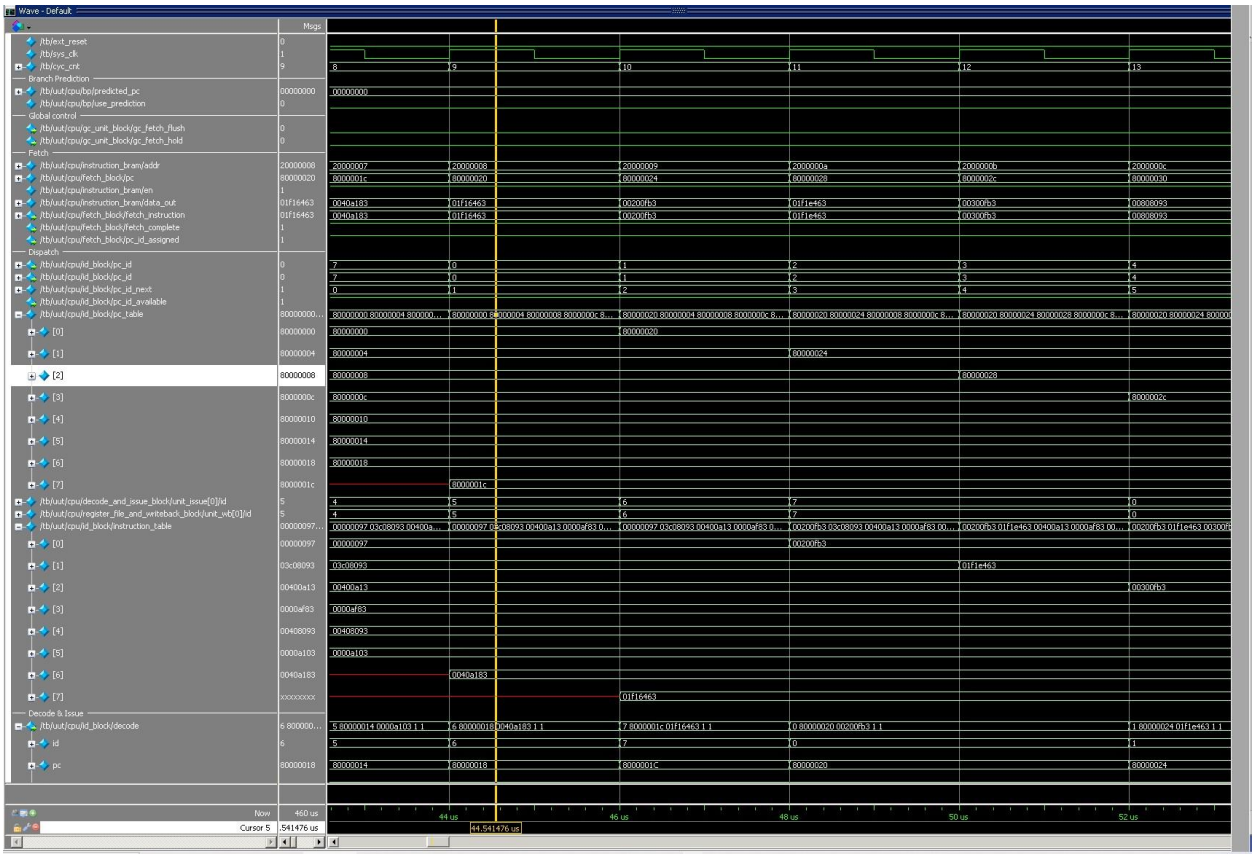

Рисунок 4 — Результат выполнения задания 5.

Номер такта	Действие
9	fetch
10	dispatch
11	Non decod
12	decod
13	issue

Анализируя диаграмму заполнить трассу выполнения программы.

Рекомендуется использовать для этого файл `pipeline.ods`, содержащий трассу тестового примера.

Результат представлен на рисунке 6.

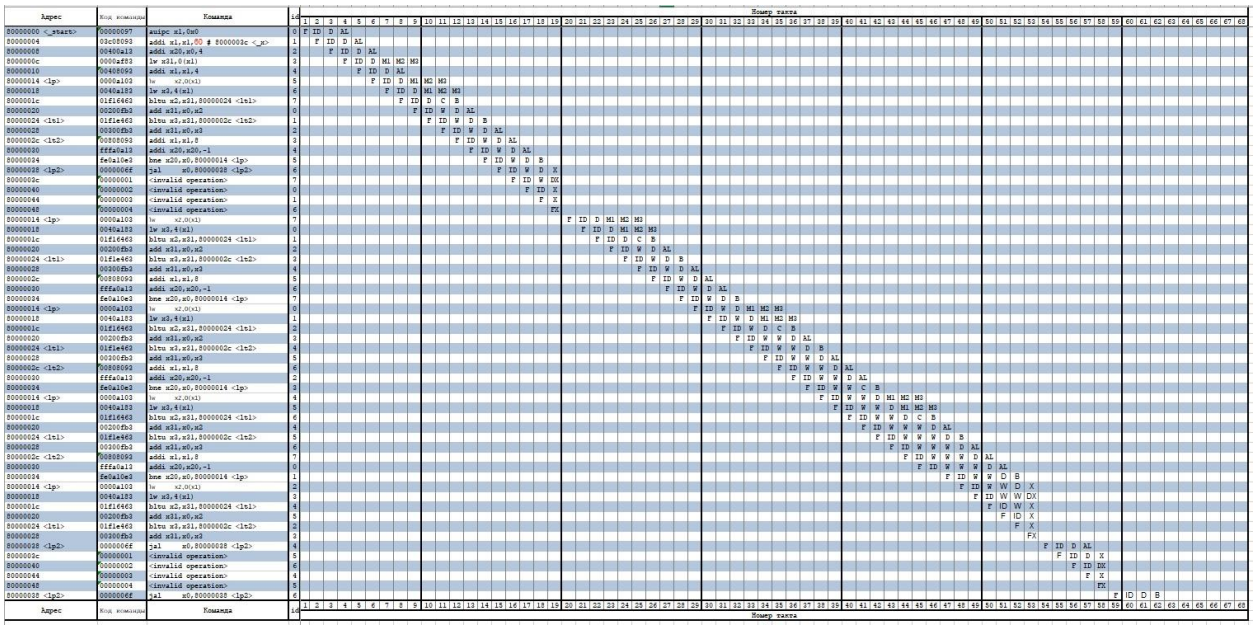

Рисунок 6 — Диаграмма неоптимизированного кода

По трассе видно наличие множественных rs конфликтов, в момент декодирования. Конфликты возникают при декодировании команд из блока АЛУ или из блока обращения к памяти после декодирования команд обращения к памяти. Повысить эффективность программы можно двумя способами: уменьшить количество конфликтных ситуаций или уменьшить количество ветвлений.

Конфликты возникают при декодировании команд из блока АЛУ или из блока обращения к памяти после декодирования команд обращения к памяти. Провести оптимизацию программы путем перестановки команд для устранения конфликтов.

Код на ассмеблере:

```
        .section .text
        .globl _start;          len = 9
#Размер массива
        enroll = 2 #Количество обрабатываемых элементов за одну итерацию
elem_sz = 4 #Размер одного элемента массива

_start:
        la x1, _x
        addi x20, x0, (len-1)/enroll
lw x31, 0(x1)          addi x1, x1,
```

```
elem_sz*1 lp:        lw x2, 0(x1)
lw x3, 4(x1)        addi x20,
x20, - 1

      bltu x2, x31, lt1
add x31, x0, x2 #! lt1:
bltu x3, x31, lt2
add x31, x0, x3 lt2:
      add x1, x1, elem_sz*enroll
bne x20, x0, lp lp2: j lp2

      .section .data
_x:    .4byte 0x1
      .4byte 0x2
      .4byte 0x3
      .4byte 0x4
      .4byte 0x5
      .4byte 0x6
      .4byte 0x7
      .4byte 0x8
      .4byte 0x9
```

Код на языке Си:

```
#include <stdio.h>

#define len 9
#define enroll 2 #define
elem_sz 4 int _x[] =
{1,2,3,4,5,6,7,8,9}; int x0 =
0;

int main()
{
        int *x1 = _x; int x20 = x0 +
        (len - 1)/enroll; int x31 =
        x1[0], x2 = 0, x3 = 0; *x1 =
        *x1 + (elem_sz*1); while(x20
        != x0)
        { x2 = x1[0]; x3
                = x1[1];
                x20 = x20 - 1;
                if (x2 > x31)
                {
```

```
                    if (x3 > x31)
                    {

                    }
                    else
                    { x31 = x0 + x3;
                    }
                }
                else
                { x31 = x0 + x2;
                }
            *x1 = *x1 + (elem_sz * enroll);
        }
   printf("%d", x31);
return 0;
}
```

Код дизассемблера:

```
80000000 <_start>:
80000000:       00000097            auipc   x1,0x0
80000004:       03c08093            addi    x1,x1,60 # 8000003c <_x>
80000008:       00400a13            addi    x20,x0,4
8000000c:       0000af83            lw      x31,0(x1)
80000010:       00408093            addi    x1,x1,4

80000014 <lp>:
80000014:       0000a103            lw      x2,0(x1)
80000018:       0040a183            lw      x3,4(x1)
8000001c:       fffa0a13            addi    x20,x20,-1
80000020:       01f16463            bltu    x2,x31,80000028 <lt1>
80000024:       00200fb3            add     x31,x0,x2

80000028 <lt1>:
80000028:       01f1e463            bltu    x3,x31,80000030 <lt2>
8000002c:       00300fb3            add     x31,x0,x3

80000030 <lt2>:
80000030:       00808093            addi    x1,x1,8
80000034:       fe0a10e3            bne     x20,x0,80000014 <lp>

80000038 <lp2>:
80000038:       0000006f            jal     x0,80000038 <lp2>
```

Составим новую трассу выполнения.

Снимок экрана представлен рисунком 7.

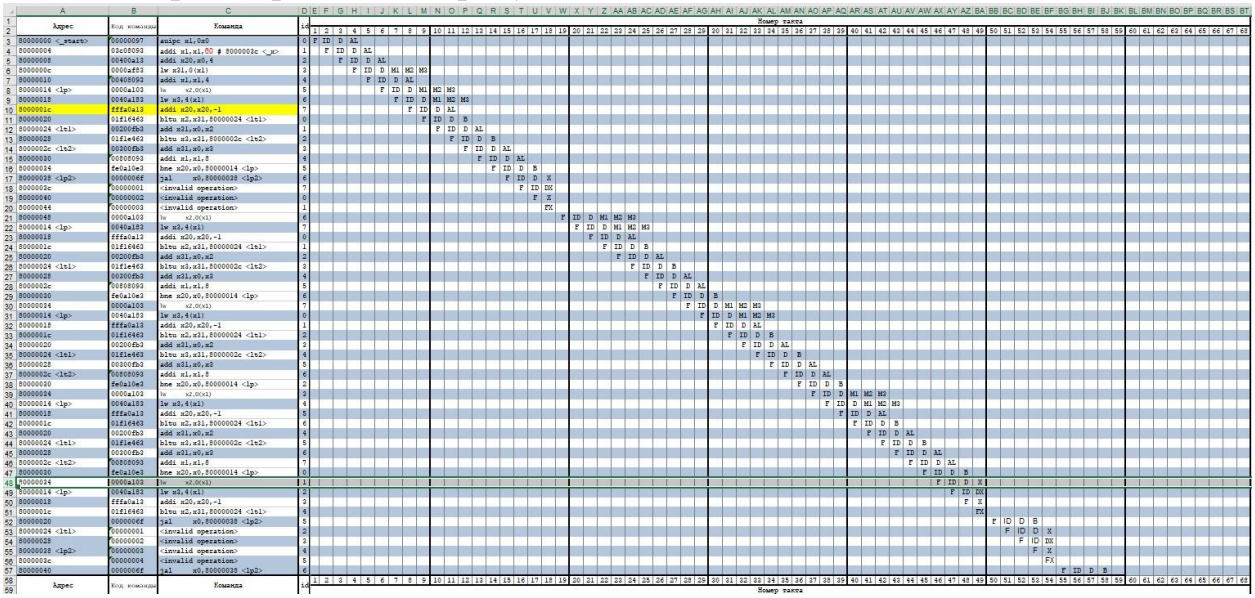

Рисунок 7 — Результат выполнения задания 4.

Вывод: в ходе выполнения лабораторной работы были изучены принципы функционирования, построения и особенности архитектуры супер скалярных конвейерных микропроцессоров, принципы проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС. Проанализирована и рассмотрены способы улучшения программы на языке Assembler с помощью ядра Taiga, развернутого под САПР Intel Quartus.