**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**отчет**

**по лабораторной работе №5**

**по дисциплине «Организация ЭВМ и систем»**

**Тема: Знакомство с рабочей средой эмулятора Ripes для работы с процессором RISC-V. Базовый ISA, система команд, состав регистров. Разработка и выполнение простой программы на ассемблере RISC-V.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Снигирёв А.А |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Молодцев Д.А |

Санкт-Петербург

2024

**Цель работы**

1. Ripes: установка, настройка, трансляция ассемблерной программы, выполнение программы в автоматическом и отладочном режимах.

2. Изучение архитектуры RISC-V, базового набора инструкций и руализация функции на ассемблере

**Задание:**

1. Разработайте процедуру на ассемблере, которая для целочисленных 32-битных входных переменных x, y, z вычисляет выражение

R = f (x, y, z),

выбираемое по таблице в соответствии с вашим вариантом.

|  |
| --- |
| Выражение (представленное в формате С/С++) |
| (a4 & (a4 & (a3 | (a4 - (a4 | (a2 | (a3 | a4))))))) |

2. Напишите программу, которая для двух наборов исходных данных x, y, z выполняет вычисление заданного выражения с помощью разработанной процедуры (вызывая её два раза), сохраняет в регистрах и выводит на экран результаты вычислений. Один вызов процедуры должен выполняться с помощью псевдоинструкции call, другой – с помощью инструкции jal.

Начальные значения {x1, y1, z1} расположить в регистрах a2, a3, a4; значения {x2, y2, z2} расположить в регистрах a5, a6, a7; значения констант a, b, c расположить в регистрах s0, s1, s2. Результаты вычисления {r1, r2} записать в регистры а1, а2.

В исходном коде обязательно должны быть употреблены следующие псевдоинструкции: call (ровно 1 раз), ret (ровно 1 раз), mv (как минимум 1 раз), li (как минимум 2 раза: 1 раз – преобразующаяся в две инструкции; 1 раз – преобразующаяся в одну инструкцию).

Моделируемые вычисления (формула, входные данные, результаты) должны выводиться в консоль.

**Основные теоретические положения:**

**1. Описание состава используемых регистров:**

Первый регистр, x0 имеет специальное назначение, он содержит 0. Вне зависимости от того, какое значение вы в него записываете, при чтении из этого регистра вы всегда получите 0. Псевдонимом регистра x0 является zero.

Псевдонимы других регистров:

ra - Используется для записи адреса возврата перед вызовом подпрограммы.

t0 - t6 temporary registers (регистры временных переменных). Подпрограммы не обязаны их сохранять.

s0 - s11 saved registers (сохраняемые регистры). Подпрограммы обязаны сохранять их состояние.

a0 - a7 function arguments (аргументы функций). Перед вызовом подпрограммы вы передаёте аргументы в эти регистры.

**Описание базового набора команд процессора RISC-V:**

1. Арифметические и логические операции: add, sub, and, or, xor и др.
2. Операции с памятью: load(lb, lh, lw) , store(sb, sh, sw), fence и др.
3. Управление переходами: branch(beq, bne, blt, bge, ...), jump(jal, jalr), call, return и др.
4. Управление регистрами: move (mv) , lui, auipc и др.
5. Привилегированные(системные) операции: ecall, ebreak и др.

**2. Краткие сведения по ассемблеру RISC-V.**

В концепции RISC инженеры устранили главный недостаток CISC-архитектур: использование большого числа функционально сложных команд. В RISC-архитектурах используется небольшое число команд фиксированной длины, но при этом увеличивается число регистров, чтобы иметь большее пространство для работы с данными и реже обращаться к памяти.

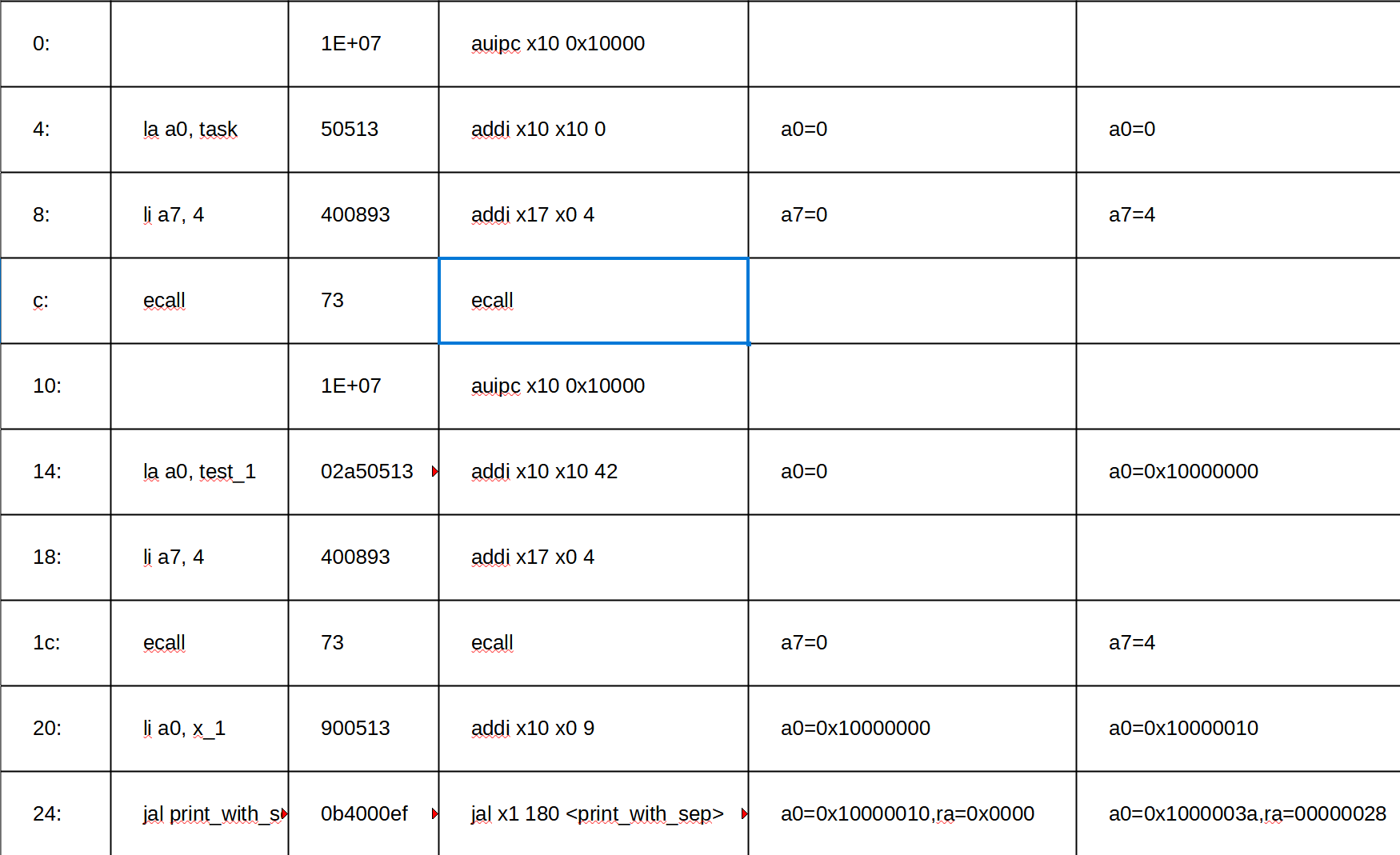
В архитектуре RISC-V имеется обязательное для реализации базовое подмножество в количестве 47 команд и несколько стандартных опциональных расширений. В базовый набор входят: минимальный набор команд арифметических/битовых операций на регистрах, команд для выполнения операций с памятью (load/store), команд условной и безусловной передачи управления/ветвления, а также небольшое число служебных инструкций (см. таблицу далее). Команды базового набора имеют длину 32 бита с выравниванием на границу 32-битного слова.

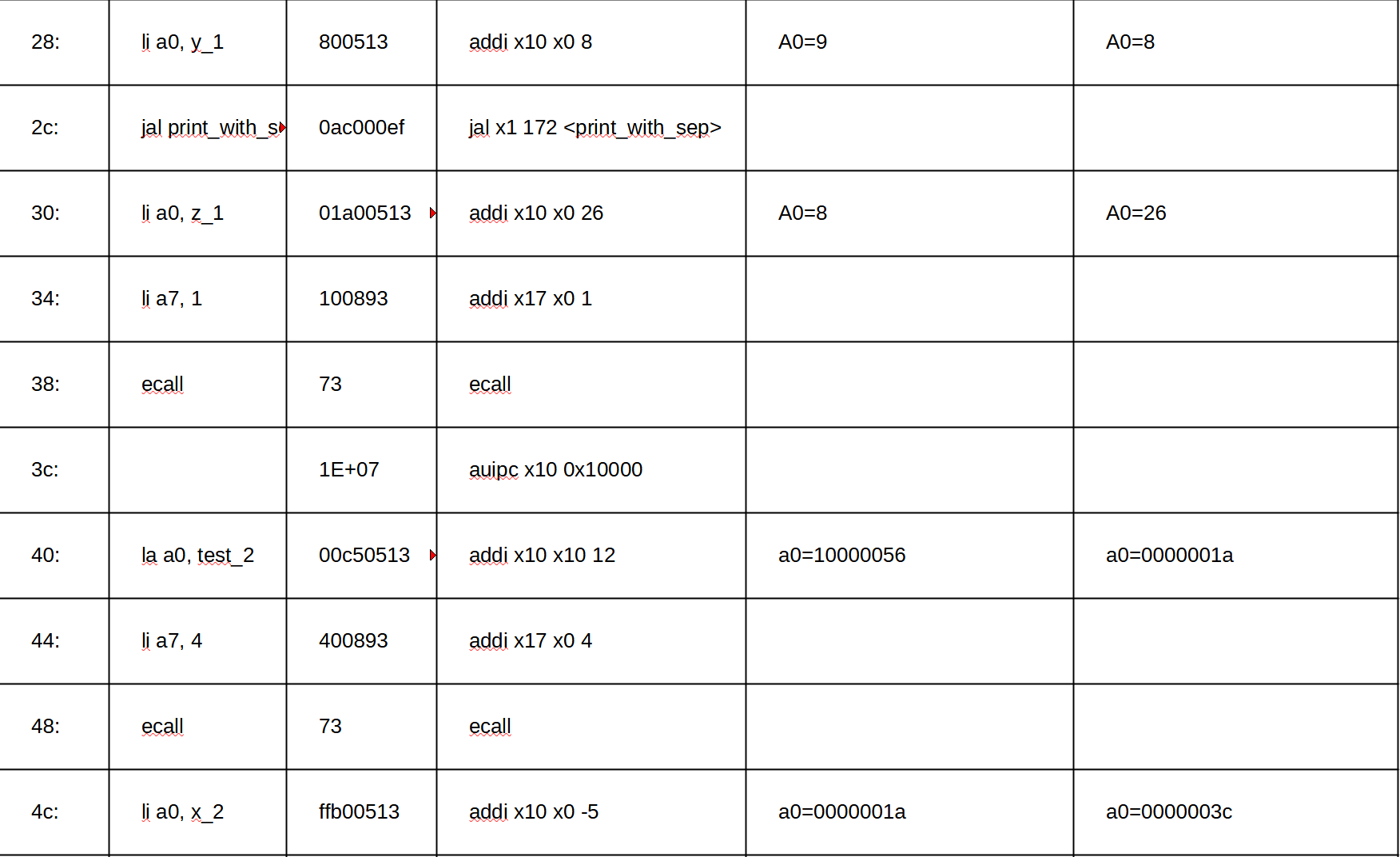
Операции условных переходов (ветвления) не используют каких-либо общих флагов, как результатов ранее выполненных операций сравнения, а непосредственно сравнивают свои регистровые операнды. Базис операций сравнения минимален, а для поддержки комплементарных операций операнды просто меняются местами.

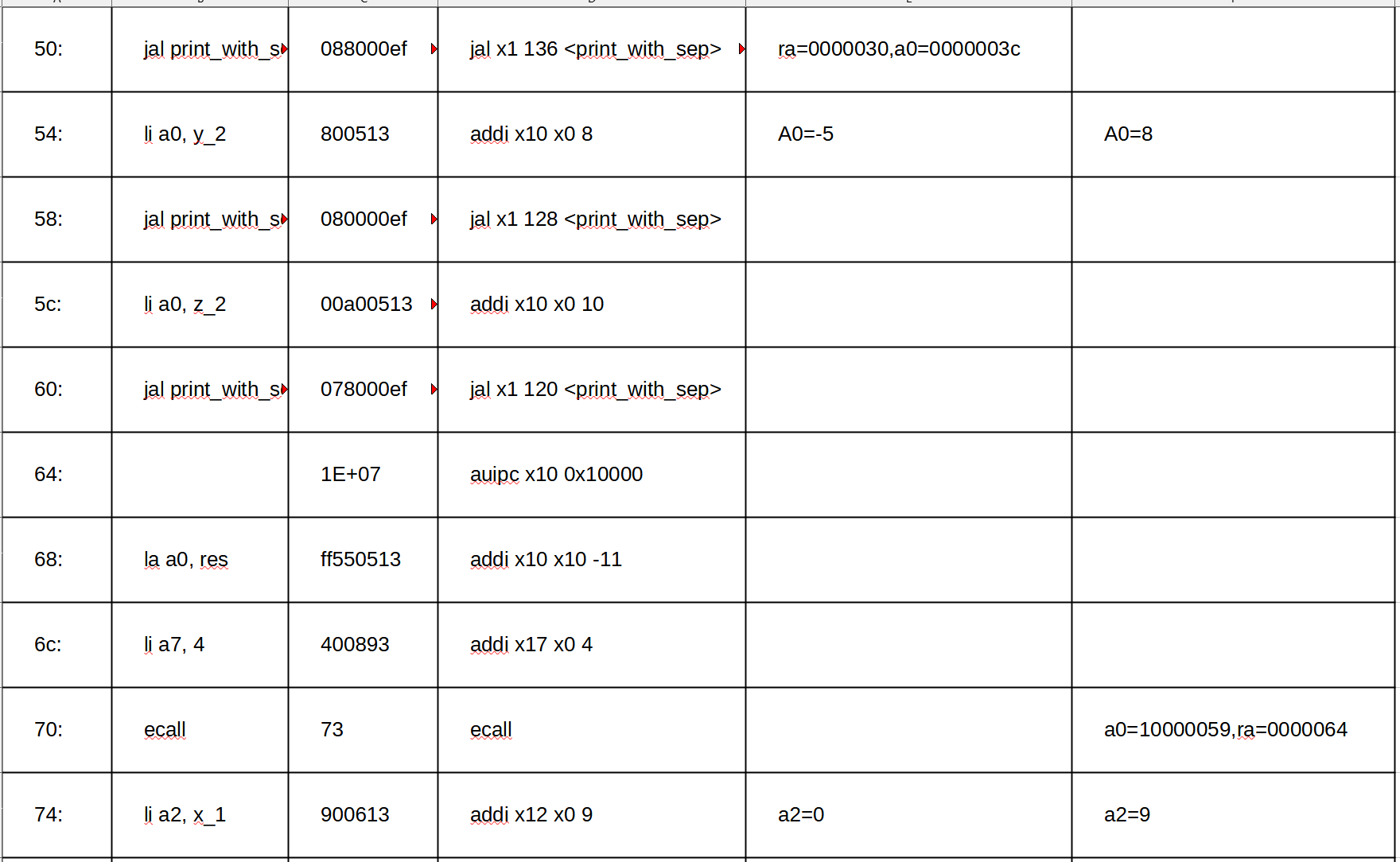
**Выполнение работы**

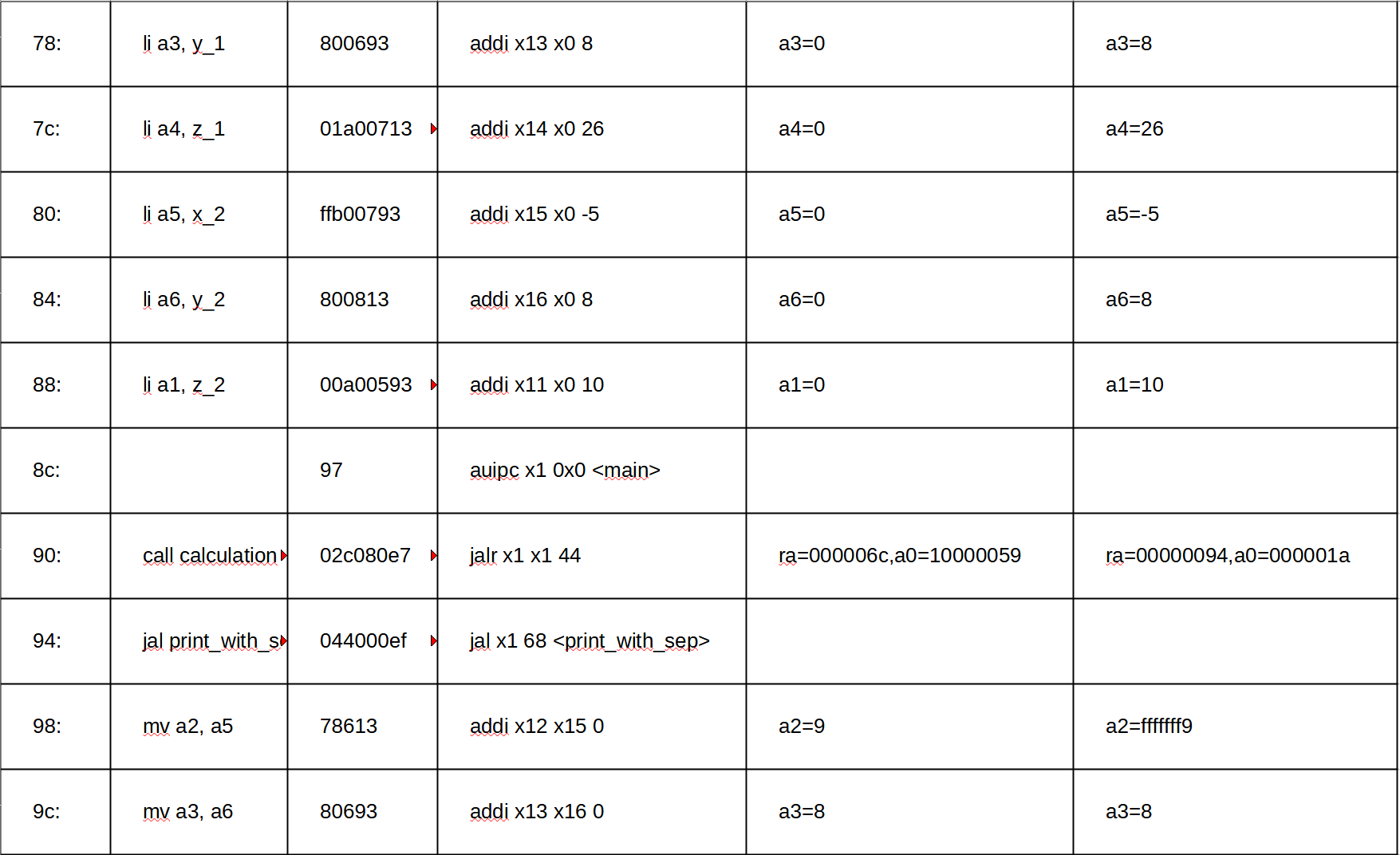
1. Исходное задание было приведено ранее.

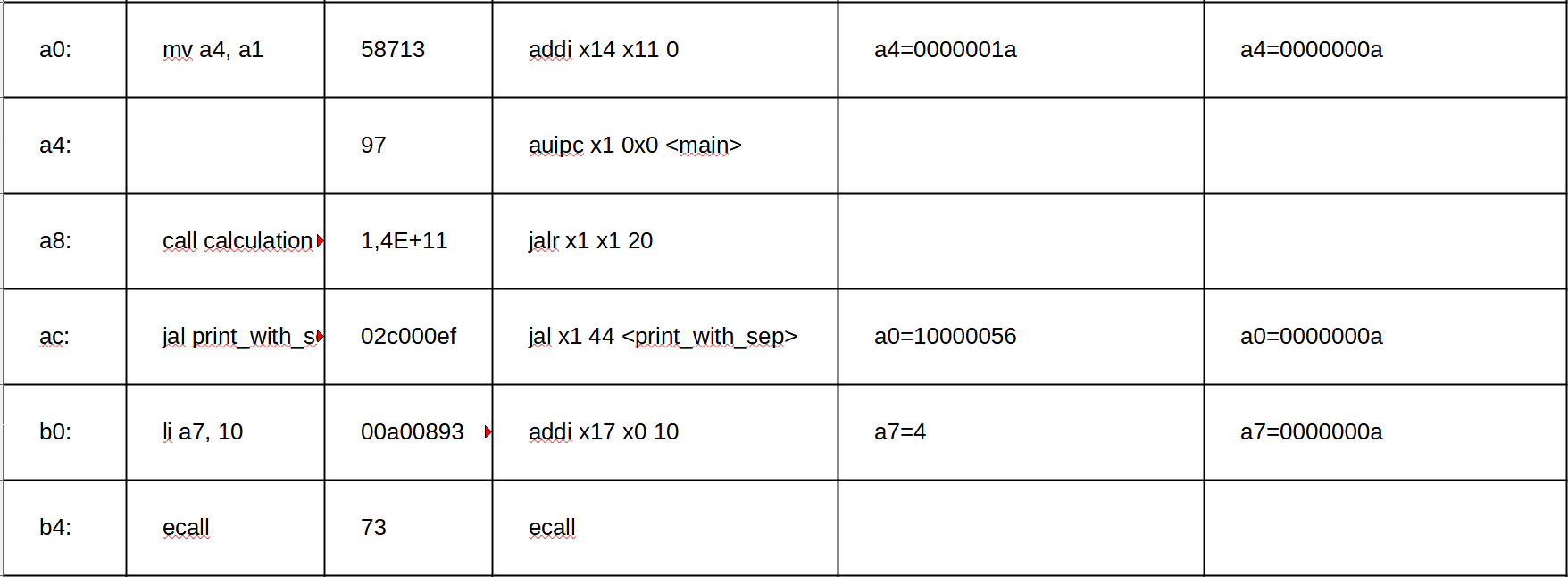
2. Результаты отладки программы в пошаговом режиме под управлением отладчика с фиксацией содержимого используемых регистров и ячеек памяти до и после выполнения команды в форме таблицы:









Таблица 1 — Программа в режиме отладчика

3. Лог тестирования программы в виде таблицы с входными данными и результатами:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемые результаты | Полученный результат |
| 1 | x1 = 9, y1 = 8, z1 = 26  x2 = 9, y2 = 8, z2 = 26 | r1 = 26  r2 = 26 | r1 = 26  r2 = 26 |
| 2 | x1 = 0, y1 = 0, z1 = 0  x2 = 0, y2 = 0, z2 = 0 | r1 = 0  r2 = 0 | r1 = 0  r2 = 0 |
| 3 | x1 = 30, y1 = 777, z1 = -100  x2 = 30, y2 = 777, z2 = -100 | r1 = -100  r2 = -100 | r1 = -100  r2 = -100 |
| 4 | x1 =33, y1 = 88, z1 = 22  x2 =33, y2 = 88, z2 = 22 | r1 = 22  r2 = 22 | r1 = 22  r2 = 22 |

Таблица 2 - Тестирование

4. Исходный код программы см. в **Приложении.**

**Вывод**

В результате выполнения лабораторной работы были получены навыки работы с эмулятором Ripes, включающие выполнение ассемблерной программы в автоматическом и отладочном режимах. Также была изучена архитектура RISC-V, базовый набор инструкций. Была разработана программа вычисления значений по заданной формуле на ассемблере RISC-V.

**ПРИЛОЖЕНИЕ**

**Код программы**

.data

# константы

.equ x\_1, 33

.equ y\_1, 88

.equ z\_1, 22

.equ x\_2, 33

.equ y\_2, 88

.equ z\_2, 22

# моделируемые вычисления для вывода в консоль

task: .string "task: (a4 & (a4 & (a3 | (a4 - (a4 | (a2 | (a3 | a4)))))))"

test\_1: .string "\nx1, y1, z1: "

test\_2: .string "\nx2, y2, z2: "

sep: .string ", "

res: .string "\nres: "

.text

main:

la a0, task # загрузка адреса строки formula в a0

li a7, 4 # загрузка 4 (код системного вызова для печати строки) в а7

ecall # систем. вызов (а7)

la a0, test\_1 # загрузка адреса строки test\_1 в a0

li a7, 4 # загрузка 4 (код системного вызова для печати строки) в а7

ecall # систем. вызов (а7)

li a0, x\_1 # загрузка значения константы x\_1 в а0

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

li a0, y\_1 # загрузка значения константы y\_1 в а0

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

li a0, z\_1 #загрузка значения константы z\_1 в а0

li a7, 1 # загрузка 1 (код систем. вывода для печати целого числа) в а7

ecall # систем. вызов (а7)

la a0, test\_2 # загрузка адреса строки test\_2 в a0

li a7, 4 # загрузка 4 (код системного вызова для печати строки) в а7

ecall # систем. вызов (а7)

li a0, x\_2 # загрузка значения константы x\_2 в а0

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

li a0, y\_2 # загрузка значения константы y\_2 в а0

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

li a0, z\_2 # загрузка значения константы z\_2 в а0

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

la a0, res # загрузка адреса строки res в a0

li a7, 4 # загрузка 4 (код системного вызова для печати строки) в а7

ecall # систем. вызов(а7)

# загрузка констант

li a2, x\_1 # загрузка значения константы x\_1 в регистр x12

li a3, y\_1 # загрузка значения константы y\_1 в регистр x13

li a4, z\_1 # загрузка значения константы z\_1 в регистр x14

li a5, x\_2 # загрузка значения константы x\_2 в регистр x15

li a6, y\_2 # загрузка значения константы y\_2 в регистр x16

li a1, z\_2 # загрузка значения константы z\_2 в регистр x17

call calculation # вызов процедуры calculation

jal print\_with\_sep

#Второй набор данных

mv a2, a5 # копирование значения регистра x12 в регистр x11 (переносим первый результат в a1 = 0)

mv a3, a6 # копирование значения регистра x15 в регистр x12

mv a4, a1 # копирование значения регистра x16 в регистр x13

call calculation # переход на метку calculation(вызов процедуры)

jal print\_with\_sep # переход на метку print\_with\_sep и сохранение адреса возврата в ra

li a7, 10 # загрузка 10 (код систем. завершения программы) в а7

ecall # систем. вызов (а7)

calculation:

or t0, a3, a4

or t1, a2, t0

or t2, a4, t1

sub t3, a4, t2

or t4, a3, t3

and t5, a4, t4

and a0, a4, t5

jr ra # переход по значению регистра ra (замена псевдокоманды ret)

print\_with\_sep:

li a7, 1 # загрузка 1 (код систем. вывода для печати целого числа) в а7

ecall # систем. вызов (а7)

la a0, sep # загрузка значения константы sep (", ") в а0

li a7, 4 # загрузка 4 (код системного вызова для печати строки) в а7

ecall # систем. вызов (а7)

ret # возврат из процедуры, переход по значению регистра ra