**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

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

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 |  | Еникеев А.А. |
| Преподаватель |  | Молодцев Д.А |

Санкт-Петербург

2024

## Цель работы

Целями работы являются:

1. Освоение работы с эмулятором Ripes: установка, настройка, трансляция ассемблерной программы, выполнение программы в автоматическом и отладочном режимах.
2. Изучение архитектуры RISC-V, базового набора инструкций и разработка простых программ 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) ,

представленное в таблице 1

Табл. 1.

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

1. 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. 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 теоритические положения

1. Краткая инструкция по работе с эмулятором Ripes

Вкладка редактора показывает два сегмента кода. С левой стороны можно написать программу ассемблера, написанную с использованием наборов инструкций RISC-V RV32(I/M/C). Далее, справа отображается второе представление кода. Это неинтерактивное представление текущей программы в собранном состоянии, обозначенное как средство просмотра программ . Мы можем просматривать собранную программу либо как дизассемблированные инструкции RISC-V, либо как необработанный двоичный код. Синюю боковую панель правого представления можно щелкнуть, чтобы установить точку останова по нужному адресу. Нажимая на «стрелочки», можно выполнять программу в автоматическом режиме или пошагово.

1. Описание состава используемых регистров и базового набора команд процессора RISC-V

Архитектура RISC-V имеет 32 регистра общего назначения (по 32 бита каждый).

* x0: всегда содержит значение 0.
* x1 (ra): регистр возврата (Return Address).
* x2 (sp): указатель стека (Stack Pointer).
* x3 (gp): указатель глобальных данных (Global Pointer).
* x4 (tp): указатель потока (Thread Pointer).
* x5–x7 (t0–t2): временные регистры.
* x8 (s0/fp): сохраненный регистр / указатель кадра.
* x9–x17 (s1–s11): сохраненные регистры.
* x10–x17 (a0–a7): регистры аргументов/результатов.
* x18–x27 (t3–t6): дополнительные временные регистры.

RISC-V использует сокращенную (RISC) инструкционную систему. Основные команды:

Арифметические и логические операции:

* add rd, rs1, rs2 — сложение.
* sub rd, rs1, rs2 — вычитание.
* and rd, rs1, rs2 — логическое И.
* or rd, rs1, rs2 — логическое ИЛИ.
* xor rd, rs1, rs2 — исключающее ИЛИ.

Команды работы с памятью:

* lw rd, offset(rs1) — загрузка слова из памяти.
* sw rs2, offset(rs1) — сохранение слова в память.

Управление потоком:

* beq rs1, rs2, label — переход при равенстве.
* bne rs1, rs2, label — переход при неравенстве.
* jal rd, label — безусловный переход с сохранением адреса возврата.
* jalr rd, offset(rs1) — косвенный переход.

1. Краткие сведения по ассемблеру RISC-V

RISC-V (Reduced Instruction Set Computing - V) — это открытая и расширяемая архитектура процессора с уменьшенным набором команд. Она разрабатывалась с целью обеспечения гибкости и простоты в проектировании процессоров, а также для использования в различных областях, от встраиваемых систем до высокопроизводительных вычислений.

Псевдокоманды — это абстракции, которые упрощают написание кода, но не являются настоящими инструкциями процессора. Ассемблер автоматически преобразует их в несколько инструкций.

## Выполнение работы

1. Шаги процедуры solution для вычисления выражения в табл.1

* mul t0, a3, a4: инструкция умножения a3\*a4, реузальтат в t0
* mul t1, a2, t0: инструкция умножения a2\*(a3\*a4), реузальтат в t1
* neg t2, a2 : инструкция, которая инвертирует знак числа -a2, результат в t2
* sub t3, a3, t2: инструкция вычитания a3 - (-a2), результат в t3
* sub t4, a3, t3: инструкция вычитания a3 - (a3 - (-a2)), результат в t4
* add t5, a3, t4: инструкция сложения a3 + (a3 - (a3 - (-a2))), результат в t5
* add t6, t1, t5: инструкция сложения (a2 \* (a3 \* a4) + (a3 + (a3 - (a3 - (-a2))))), результат в t6
* or a0, t6, a4: инструкция побитового ИЛИ ((a2 \* (a3 \* a4) + (a3 + (a3 - (a3 - (-a2))))) | a4), результат в a0
* ret: возвращает управление на адрес, который находится в регистре ra

1. 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 см. в табл. 2.

Табл. 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Адрес инстр. | (Псевдо-) интср. | Инструкция(и) | 16-ричный код инстр. | Содержимое регистров и ячеек памяти | |
| До выполн. инстр. | После выполн. инстр. |
| 0 | li a2, x\_1 | addi x12 x0 5 | 00500613 | a2=0x00000000 | a2=0x00000005 |
| 4 | li a3, y\_1 | addi x13 x0 3 | 00300693 | a3=0x00000000 | a3=0x00000003 |
| 8 | li a4, z\_1 | addi x14 x0 2 | 00200713 | a4=0x00000000 | a4=0x00000002 |
| c | li a5, x\_2 | addi x15 x0 7 | 00700793 | a5=0x00000000 | a5=0x00000007 |
| 10 | li a6, y\_2 | addi x16 x0 4 | 00400813 | a6=0x00000000 | a6=0x00000004 |
| 14 | li a7, z\_2 | addi x17 x0 1 | 00100893 | a7=0x00000000 | a7=0x00000001 |
| 18 | la a0, formula | auipc x10 0x10000 | 10000517 | a0=0x00000000 | a0=0x10000018 |
| 1c |  | addi x10 x10 -24 | fe850513 | a0=0x10000018 | a0=0x10000000 |
| 20 | jal print\_string | jal x1 316 <print\_string> | 13c000ef | ra=0x00000000 | ra=0x00000024 |
| 15c | addi sp, sp, -4 | addi x2 x2 -4 | ffc10113 | sp=0x7ffffff0 | sp=0x7fffffec |
| 160 | sw a7, 0(sp) | sw x17 0 x2 | 01112023 |  |  |
| 164 | li a7, 4 | addi x17 x0 4 | 00400893 | a7=0x00000001 | a7=0x00000004 |
| 168 | ecall | ecall | 00000073 |  |  |
| 16c | lw a7, 0(sp) | lw x17 0 x2 | 00012883 |  |  |
| 170 | addi sp, sp, 4 | addi sp, sp, 4 | 00410113 |  |  |
| 174 | jr ra | jalr x0 x1 0 | 00008067 |  |  |
| 24 |  | auipc x10 0x10000 | 10000517 | a0=0x10000000 | a0=0x10000024 |
| 28 | la a0, input1 | addi x10 x10 32 | 02050513 | a0=0x10000024 | a0=0x10000044 |
| 2c | jal print\_string | jal x1 304 <print\_string> | 130000ef | ra=0x00000024 | ra=0x00000030 |
| 30 | mv a0, a2 | addi x10 x12 0 | 00060513 | a0=0x10000044 | a0=0x00000005 |
| 34 | jal print\_int | jal x1 268 <print\_int> | 10c000ef | ra=0x00000030 | ra=0x00000038 |
| 140 | addi sp, sp, -4 | addi x2 x2 -4 | ffc10113 | sp=0x7ffffff0 | sp=0x7fffffec |
| 144 | sw a7, 0(sp) | sw x17 0 x2 | 01112023 |  |  |
| 148 | li a7, 1 | addi x17 x0 1 | 00100893 |  |  |
| 14c | ecall | ecall | 00000073 |  |  |
| 150 | lw a7, 0(sp) | lw x17 0 x2 | 00012883 |  |  |
| 154 | addi sp, sp, 4 | addi x2 x2 4 | 00410113 | sp=0x7fffffec | sp=0x7ffffff0 |
| 158 | jr ra | jalr x0 x1 0 | 00008067 |  |  |
| 3c | la a0, sep | addi x10 x10 142 | 08e50513 | a0=0x00000005 | a0=0x10000038 |
| 40 | jal print\_string | jal x1 284 <print\_string> | 11c000ef | ra=0x00000038 | ra=0x00000044 |
| 44 | mv a0, a3 | addi x10 x13 0 | 00068513 | a0=0x100000c6 | a0=0x00000003 |
| 48 | jal print\_int | jal x1 248 <print\_int> | 0f8000ef | ra=0x00000044 | ra=0x0000004c |
| 50 | la a0, sep | addi x10 x10 122 | 07a50513 | a0=0x00000003 | a0=0x1000004c |
| 54 | jal print\_string | jal x1 264 <print\_string> | 108000ef | ra=0x0000004c | ra=0x0000004c |
| 58 | mv a0, a4 | addi x10 x14 0 | 00070513 | a0=0x100000c6 | a0=0x00000002 |
| 5c | jal print\_int | jal x1 228 <print\_int> | 0e4000ef | ra=0x00000058 | ra=0x00000060 |
| 64 | la a0, nl | addi x10 x10 105 | 06950513 | a0=0x00000002 | a0=0x10000060 |
| 68 | jal print\_string | jal x1 244 <print\_string> | 0f4000ef | ra=0x00000060 | ra=0x0000006c |
| 6c |  | auipc x1 0x0 <main> | 00000097 | sp=0x7fffffec | sp=0x7ffffff0 |
| 70 | call solution | jalr x1 x1 176 | 0b0080e7 | ra=0x0000006c | ra=0x00000074 |
| 11c | mul t0, a3, a4 | mul x5 x13 x14 | 02e682b3 | t0=0x00000000 | t0=0x00000006 |
| 120 | mul t1, a2, t0 | mul x6 x12 x5 | 02560333 | t1=0x00000000 | t1=0x0000001e |
| 124 | neg t2, a2 | sub x7 x0 x12 | 40c003b3 | t2=0x00000000 | t2=0xfffffffb |
| 128 | sub t3, a3, t2 | sub x28 x13 x7 | 40768e33 | t3=0x00000000 | t3=0x00000008 |
| 12c | sub t4, a3, t3 | sub x29 x13 x28 | 41c68eb3 | t4=0x00000000 | t4=0xfffffffb |
| 130 | add t5, a3, t4 | add x30 x13 x29 | 01d68f33 | t5=0x00000000 | t5=0xfffffffe |
| 134 | add t6, t1, t5 | add x31 x6 x30 | 01e30fb3 | t6=0x00000000 | t6=0x0000001c |
| 138 | or a0, t6, a4 | or x10 x31 x14 | 00efe533 | a0=0x100000c9 | a0=0x0000001e |
| 13c | ret | jalr x0 x1 0 | 00008067 |  |  |
| 74 | mv a1, a0 | addi x11 x10 0 | 00050593 | a1=0x00000000 | a1=0x0000001e |
| 7c | la a0, result1 | addi x10 x10 30 | 01e50513 | a0=0x0000001e | a0=0x10000078 |
| 80 | jal print\_string | jal x1 220 <print\_string> | 0dc000ef | ra=0x00000074 | ra=0x00000084 |
| 84 | mv a0, a1 | addi x10 x11 0 | 00058513 | a0=0x10000096 | a=0x0000001e |
| 88 | jal print\_int | jal x1 184 <print\_int> | 0b8000ef | ra=0x00000084 | ra=0x0000008c |

1. Лог тестирования см. в табл. 3.

Табл. 3

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемые  результаты | Полученные  результаты |
| 1 | x1=150, y1=200, z1=500  x2=-1, y2=12, z2=28 | (15000054; -323) | (15000054; -323) |
| 2 | x1=150, y1=200, z1=-500  x2=1, y2=-12, z2=28 | (-386; -321) | (-386; -321) |
| 3 | x1=0, y1=1, z1=-1  x2=0, y2=0, z2=0 | (-1; 0) | (-1; 0) |

Исходный код программы см. в прил. A.

**Выводы**

В ходе лабораторной работы были освоены основы работы с эмулятором Ripes, включая его установку, настройку, трансляцию ассемблерных программ и выполнение их в автоматическом и отладочном режимах. Также была изучена архитектура RISC-V и базовый набор инструкций, что позволило разработать процедуру для вычисления заданного выражения с использованием целочисленных 32-битных переменных. В процессе работы были выполнены два вызова процедуры с использованием псевдокоманды call и инструкции jal, а также применены псевдокоманды mv, li, и ret. Результаты вычислений были корректно сохранены в регистрах и выведены на экран, что подтвердило правильность работы программы.

# Приложение A

**Исходный код программы**

**Файл LR5.s:**

# Главная программа

.text

main:

# ---------------------------

# Инициализация начальных данных

# ---------------------------

li a2, x\_1 # Загружаем значение x1 в регистр a2

li a3, y\_1 # Загружаем значение y1 в регистр a3

li a4, z\_1 # Загружаем значение z1 в регистр a4

li a5, x\_2 # Загружаем значение x2 в регистр a5

li a6, y\_2 # Загружаем значение y2 в регистр a6

li a7, z\_2 # Загружаем значение z2 в регистр a7

# ---------------------------

# Вывод формулы

# ---------------------------

la a0, formula # Загружаем адрес строки с формулой в a0

jal print\_string # Переход к процедуре вывода строки

# ---------------------------

# Вывод входных данных первого набора

# ---------------------------

la a0, input1 # Загружаем адрес строки "Входные данные x1, y1, z1:" в a0

jal print\_string # Вывод строки

mv a0, a2 # Передаем значение x1 в a0 для вывода

jal print\_int # Печать x1

la a0, sep # Загружаем адрес строки ", " (разделитель)

jal print\_string # Печать разделителя

mv a0, a3 # Передаем значение y1 в a0 для вывода

jal print\_int # Печать y1

la a0, sep # Загружаем адрес строки ", " (разделитель)

jal print\_string # Печать разделителя

mv a0, a4 # Передаем значение z1 в a0 для вывода

jal print\_int # Печать z1

la a0, nl # Загружаем адрес строки новой строки

jal print\_string # Печать новой строки

# ---------------------------

# Вычисление R1

# ---------------------------

call solution # Вызов процедуры solution для вычисления R1

mv a1, a0 # Сохраняем результат R1 в регистре a1

# ---------------------------

# Вывод результата R1

# ---------------------------

la a0, result1 # Загружаем адрес строки "Результат R1:"

jal print\_string # Печать строки

mv a0, a1 # Передаем значение R1 в a0 для вывода

jal print\_int # Печать R1

la a0, nl # Загружаем адрес новой строки

jal print\_string # Печать новой строки

# ---------------------------

# Установка второго набора данных

# ---------------------------

mv a2, a5 # Передаем значение x2 в a2

mv a3, a6 # Передаем значение y2 в a3

mv a4, a7 # Передаем значение z2 в a4

# ---------------------------

# Вывод входных данных второго набора

# ---------------------------

la a0, input2 # Загружаем адрес строки "Входные данные x2, y2, z2:"

jal print\_string # Печать строки

mv a0, a2 # Передаем значение x2 в a0 для вывода

jal print\_int # Печать x2

la a0, sep # Загружаем адрес строки ", " (разделитель)

jal print\_string # Печать разделителя

mv a0, a3 # Передаем значение y2 в a0 для вывода

jal print\_int # Печать y2

la a0, sep # Загружаем адрес строки ", " (разделитель)

jal print\_string # Печать разделителя

mv a0, a4 # Передаем значение z2 в a0 для вывода

jal print\_int # Печать z2

la a0, nl # Загружаем адрес новой строки

jal print\_string # Печать новой строки

# ---------------------------

# Вычисление R2

# ---------------------------

jal solution # Вызов процедуры solution для вычисления R2

mv a2, a0 # Сохраняем результат R2 в регистре a2

# ---------------------------

# Вывод результата R2

# ---------------------------

la a0, result2 # Загружаем адрес строки "Результат R2:"

jal print\_string # Печать строки

mv a0, a2 # Передаем значение R2 в a0 для вывода

jal print\_int # Печать R2

la a0, nl # Загружаем адрес новой строки

jal print\_string # Печать новой строки

# ---------------------------

# Завершение программы

# ---------------------------

li a7, 10 # Код ecall для завершения программы

ecall # Завершение выполнения

# ---------------------------

# Процедура solution: вычисление результата

# ---------------------------

solution:

# Входные данные: x = a2, y = a3, z = a4

# Выходной результат: записывается в a0

mul t0, a3, a4 # a3 \* a4

mul t1, a2, t0 # a2 \* (a3 \* a4)

neg t2, a2 # -a2

sub t3, a3, t2 # a3 - (-a2)

sub t4, a3, t3 # a3 - (a3 - (-a2))

add t5, a3, t4 # a3 + (a3 - (a3 - (-a2)))

add t6, t1, t5 # (a2 \* (a3 \* a4) + (a3 + (a3 - (a3 - (-a2)))))

or a0, t6, a4 # ((a2 \* (a3 \* a4) + (a3 + (a3 - (a3 - (-a2))))) | a4)

ret # Возврат управления

# ---------------------------

# Процедура print\_int: вывод целого числа

# ---------------------------

print\_int:

addi sp, sp, -4 # Выделяем 4 байта в стеке для сохранения данных

sw a7, 0(sp) # Сохраняем регистр a7 (код ecall)

li a7, 1 # Код ecall для вывода целого числа

ecall # Системный вызов для печати

lw a7, 0(sp) # Восстанавливаем значение a7 из стека

addi sp, sp, 4 # Освобождаем 4 байта в стеке

jr ra # Переход по адресу возврата

# ---------------------------

# Процедура print\_string: вывод строки

# ---------------------------

print\_string:

addi sp, sp, -4 # Выделяем 4 байта в стеке для сохранения данных

sw a7, 0(sp) # Сохраняем регистр a7 (код ecall)

li a7, 4 # Код ecall для вывода строки

ecall # Системный вызов для печати

lw a7, 0(sp) # Восстанавливаем значение a7 из стека

addi sp, sp, 4 # Освобождаем 4 байта в стеке

jr ra # Переход по адресу возврата

# ---------------------------

# Данные

# ---------------------------

.data

.equ x\_1, 5

.equ y\_1, 3

.equ z\_1, 2

.equ x\_2, 7

.equ y\_2, 4

.equ z\_2, 1

formula: .string "Формула: (a2 \* (a3 \* a4) + (a3 + (a3 - (a3 - (-a2)))) | a4)\n"

input1: .string "Входные данные x1, y1, z1: "

input2: .string "Входные данные x2, y2, z2: "

result1: .string "Результат R1: "

result2: .string "Результат R2: "

sep: .string ", "

nl: .string "\n"