**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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Беннер В.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++) |
| a0 = (a3+((-a3)|(a4&(a2 - (a2|(a3|(a2|a2))))))) |

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**

0. Среда разработки была настроена

1. Исходное выражение:

a0 = (a3+((-a3)|(a4&(a2 - (a2|(a3|(a2|a2)))))))

2. Результаты отладки программы в пошаговом режиме под управлением отладчика. Таблица приведена не полностью. Многократные действия написаны только один раз.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0: | 00c00613 | addi x12 x0 12 | a2=0 | a2=12 |
| 4: | 500693 | addi x13 x0 5 | a3=0 | a3=5 |
| 8: | 600713 | addi x14 x0 6 | a4=0 | a4=6 |
| 10: | ff450513 | addi x10 x10 -12 | a0=0 | a0=1000000c |
| 18: | 09c080e7 | call print\_string: | ra=0 | ra=00000014 |
| b0: | 400893 | addi x17 x0 4 | a7=0 | a7=4 |
| b4: | 73 | ecall | ra=14 | ra=1c |
| b8: | 513 | addi x10 x0 0 | a0=1000000c | a0=0 |
| bc: | 8067 | jalr x0 x1 0 |  |  |
| 20: | 800800000000 | call print\_newline | ra=1c | ra=24 |
| 9c: | 10000517 | auipc x10 0x10000 |  |  |
| a0: | f9d50513 | addi x10 x10 -99 | a0=1000000c | a0=1000009c |
| a4: | 400893 | addi x17 x0 4 | a7=4 | a7=4 |
| a8: | 73 | ecall |  |  |
| ac: | 8067 | jalr x0 x1 0 |  |  |
|  |  | Повтор вызовов процедур |  |  |
| 38: | 400800000000 | call solution | ra=34 | ra=3c |
| 74: | 00c66333 | or x6 x12 x12 | t1=0 | t1=c |
| 78: | 0066e3b3 | or x7 x13 x6 | t2=0 | t2=d |
| 7c: | 7,66E+035 | or x28 x12 x7 | t3=0 | t3=d |
| 80: | 41c60333 | sub x6 x12 x28 | t1=c | t1=ffffffff |
| 84: | 006773b3 | and x7 x14 x6 | t2=d | t2=6 |
| 88: | 40d00333 | sub x6 x0 x13 | t1=ffffffff | t1=fffffffb |
| 8c: | 7,36E+035 | or x28 x6 x7 | t3=d | t3=ffffffff |
| 90: | 01c68533 | add x10 x13 x28 | a0=0 | a0=4 |
| 94: | 8067 | jalr x0 x1 0 |  |  |
| 98: | 8067 | jalr x0 x1 0 |  |  |
| 3c: | 100893 | addi x17 x0 1 | a7=4 | a7=1 |
| 40: | 73 | ecall |  |  |
|  |  | Повтор логики c jal вместо call |  |  |
| 6c: | 00a00893 | addi x17 x0 10 | a7=1 | a7=a |
| 70: | 73 | ecall |  |  |

3. Тестирование:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемые результаты | Полученный результат |
| 1 | A1=10, A2=20, A3=30 | res1 = 0  res2 = 0 | r1 = 0  r2 = 0 |
| 2 | A1=10, A2=7, A3=3 | r1 = 2  r2 = 2 | r1 = 2  r2 = 2 |
| 3 | A1=12,A2=5, A3=6 | r1 = 4  r2 = 4 | r1 = 4  r2 = 4 |
| 4 | A1=10, A2=20, A3=30 | r1 = 0  r2 = 0 | r1 = 0  r2 = 0 |

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

**Вывод**

В результате выполнения лабораторной работы были получены навыки работы с ассемблером Risc-v на эмуляторе Ripes. Разработана программа вычисления значения по заданной формуле.

**ПРИЛОЖЕНИЕ**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

.data # Начало сегмента данных

# Входные данные

.equ A2, 10 #

.equ A3, 20 #

.equ A4, 30 #

zadanie: .string "zadanie: a0 = (a3+((-a3)|(a4&(a2 - (a2|(a3|(a2|a2)))))))"

newline: .string "\n"

res1: .string "res1: "

res2: .string "res2: "

.text # Начало сегмента кода

.globl main # .globl обозначает, что часть кода будет видна из других файлов

main:

li a2,A2 # Загружаем в регистры константы

li a3,A3 #

li a4,A4 #

la a0, zadanie # Загружаем в a0 адрес нужной строки

call print\_string

call print\_newline

la a0, res1 # Загружаем в a0 адрес нужной строки

call print\_string

call solution # Вызов функции solution

li a7,1 # Загрузка в регитр a7 номера процедуры печати числа, лежащего в регистре a0

ecall # Системное прерывание

call print\_newline

la a0, res2 # Загружаем в a0 адрес нужной строки

call print\_string

li a0,0 # Обнуляем a0

jal solution # Вызываем через прыжок, jal сохраняет указатель на следующую инструкцию в регистр ra

li a7,1 # Загрузка в регитр a7 номера процедуры печати числа, лежащего в регистре a0

ecall # Системное прерывание

li a7,10 # Загрузка в регитр a7 номера процедуры завершения программы

ecall # Системное прерывание

.globl solution

solution: # Логика обработки

or t1,a2,a2 # a2|a2

or t2,a3,t1 # (a3|(a2|a2))

or t3,a2,t2 # (a2|(a3|(a2|a2)))

sub t1,a2,t3 # (a2 - (a2|(a3|(a2|a2))))

and t2,a4,t1 # (a4&(a2 - (a2|(a3|(a2|a2)))))

neg t1,a3 # t1 = -a3

or t3,t1,t2 # ((-a3)|(a4&(a2 - (a2|(a3|(a2|a2))))))

add a0, a3, t3 # a0 = (a3+((-a3)|(a4&(a2 - (a2|(a3|(a2|a2)))))))

jr ra # Команда, совершающая прыжок по регистру

ret

print\_newline: # Процедура перехода на новую строку

la a0, newline # Загружаем в a0 адрес переноса строки

li a7, 4 # Загрузка в a7 функции печати строки, адрес которой лежит в a0

ecall

ret

print\_string: # Процедура печати строки по адресу a0

li a7, 4 # Загрузка в a7 функции печати строки, адрес которой лежит в a0

ecall

li a0,0 # Обнуление a0

ret