**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по лабораторной работе №6**

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

**Тема: Изучение режимов адресации в ассемблере RISC-V.**

| Студентка гр. 3388 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Беннер В.А |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Молодцев Д.А |

Санкт-Петербург

2024

**Цель работы**

1. Разработка программы преобразования данных для приобретения практических навыков программирования на языке ассемблера.

2. Закрепление знаний по режимам адресации в процессоре RISC-V.

**Задание:**

Требуется написать программу, которая использует разные режимы адресации для вычислений по массиву данных в памяти. Результатом выполнения вашей программы будет измененный массив в памяти.

В качестве исходных данных дается стартовый адрес в памяти для хранения массива, количество элементов в массиве и формула для требуемых вычислений (Вычисления включают изменения каждого элемента массива в зависимости от условия).

При автоматической проверке вашей программы исходные данные располагаются в регистрах следующим образом:

- a1 - адрес памяти, где расположен массив

- a2 - количество элементов в массиве

Считайте, что массив уже инициализирован и заполнен данными.

Ваша программа должна иметь следующую структуру:

.globl solution

solution:

# при старте данной метки ваша программа должна выполнить

# необходимые вычисления и изменить элементы массива согласно ветке условия и формуле в ней

ret

Доступ к массиву (чтение, изменение) должен выполняться из памяти.

Формула для вычислений будет выведена ниже (arr[i] - элемент массива, считаем что arr[-1] == 0):

ЕСЛИ ((arr[9] + arr[3] + arr[8]) <= 411)

ТО (arr[i] = arr[i - 1] + 64)

ИНАЧЕ (arr[i] = arr[i] | 69)

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

**Основные теоретические положения:**

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

2. Краткие сведения по режимам адресации в ассемблере RISC-V.

*Регистровая адресация*

При регистровой адресации регистры используются для всех операндов-источников и операндов-назначений

*Непосредственная адресация*

При непосредственной адресации в качестве операндов наряду с регистрами используют константы (непосредственные операнды).

addi rd,rs1,12 # rd = rs1 + 12

andi rd,rs1,-8 # rd = rs1 & 0xFF8

Чтобы использовать константы большего размера, следует использовать инструкцию непосредственной записи в старшие разряды lui (load upper immediate), за которой следует инструкция непосредственного сложения addi Инструкция lui загружает 20-битное значение сразу в 20 старших битов и помещает нули в младшие биты:

lui s2, 0xABCDE # s2 = 0xABCDE000

addi s2, s2, 0x123 # s2 = 0xABCDE123

При использовании многоразрядных непосредственных операндов, если указанный в addi 12-битный непосредственный операнд отрицательный, старшая часть постоянного значения в lui должна быть увеличена на единицу.

*Базовая адресация*

Инструкции для доступа в память, такие как загрузка слова(чтение памяти) (lw) и сохранение слова(запись в память) (sw), используют базовую адресацию. Эффективный адрес операнда в памяти вычисляется путем сложения базового адреса в регистре rs1 и 12-битного смещения с расширенным знаком, являющегося непосредственным операндом. Операции загрузки (lw) – это инструкции типа I, а операции сохранения (sw) – инструкции типа S.

lw rd, 36(rs1) # rd = M[rs1+imm][0:31]

Поле rs1 указывает на регистр, содержащий базовый адрес, а поле rd указывает на регистр-назначение. Поле imm, хранящее непосредственный операнд, содержит 12-битное смещение, равное 36. В результате регистр rd содержит значение из ячейки памяти rs1+36

sw rs2, 8(rs1) # M[rs1+imm][0:31] = rs2[0:31]

Инструкция сохранения слова sw демонстрирует запись значения из регистра rs2 в слово памяти, расположенное по адресу rs1+8

*Адресация относительно счетчика команд*

Инструкции условного перехода, или ветвления, используют адресацию относительно счетчика команд для определения нового значения счетчика команд в том случае, если нужно осуществить переход. Смещение со знаком прибавляется к счетчику команд (PC) для определения нового значения PC, поэтому тот адрес, куда будет осуществлен переход, называют адресом относительно счетчика команд.

Инструкции перехода по условию (beq, bne, blt, bge, bltu, bgeu) типа B и jal (переход и связывание) типа J используют для смещения 13- и 21-битные константы со знаком соответственно.

Инструкция jal может быть использована как для вызова функций, так и для простого безусловного перехода. В RISC-V используется соглашение, что адрес возврата должен быть сохранён в регистре адреса возврата ra ( x1).

Инструкция jal не имеет достаточного места для кодирования полного 32-битного адреса. Это означает, что вы не можете сделать переход куда-либо в коде, если ваша программа больше максимального значения смещения.

**Выполнение работы**

1. Выделяется место на стеке для сохранения регистров ra, s0 (адрес массива), s1 (счетчик цикла), s2 (размер массива) и s3 (временная переменная для хранения arr[i-1]). Входные аргументы (адрес массива и размер) сохраняются в s0 и s2.

2. Цикл loop\_start: Цикл перебирает элементы массива от 0 до s2 - 1.

3. Проверка условия: В каждой итерации вычисляется сумма элементов arr[9], arr[3] и arr[8] (обратите внимание, что это всегда эти конкретные элементы, а не текущий и предыдущий). Если сумма меньше или равна 411, переходит в ветку then\_branch, иначе — в else\_branch.

4. Ветка else\_branch: Элемент arr[i] подвергается побитовой дизъюнкции с числом 69 (arr[i] |= 69).

5. Ветка then\_branch: Элемент arr[i] получает значение arr[i-1] + 64. Для arr[0] значение присваивается равное 64.

6. Инкремент счетчика и переход: Счетчик цикла s1 увеличивается на 1, и происходит переход к началу цикла loop\_start.

7.Восстанавливаются значения сохраненных регистров, освобождается место на стеке, и функция завершается с помощью ret.

В целом, функция модифицирует элементы массива на основе условия, при этом использует фиксированные индексы (9, 3, 8) для проверки условия и отличается обработкой первого элемента массива. Код написан на ассемблере RISC-V.

Исходный код программы см. в приложении А**.** Тестирование см. в приложении В.

**Вывод**

В результате выполнения лабораторной работы были изучены способы адресации в ассемблере risc-v. Также были получены навыки в их использовании в ходе написания программы, выполняющей изменение массива в соответствии с условием.

**ПРИЛОЖЕНИЕ А**

**Код программы**

.globl solution

solution:

addi sp, sp, -40 # Выделение места под регистры

sd ra, 0(sp) # Сохраняем ra (адрес возврата)

sd s0, 8(sp) # Сохраняем s0 (адрес массива)

sd s1, 16(sp) # Сохраняем s1 (счетчик цикла)

sd s2, 24(sp) # Сохраняем s2 (количество элементов)

sd s3, 32(sp) # Сохраняем s3 (для arr[i-1])

# Сохраняем входные аргументы

mv s0, a1 # s0 = адрес массива

mv s2, a2 # s2 = количество элементов

# Инициализируем счетчик цикла

li s1, 0 # i = 0

j loop\_start

loop\_start:

bge s1, s2, loop\_end # if (i >= len(arr)) goto loop\_end

# Вычисление условия (arr[9], arr[3], arr[8] - оригинальные значения - в каждой итерации!)

# arr[9]

li t0, 9

slli t0, t0, 3 # t0 = 9 \* 8

add t0, t0, s0 # t0 = arr + 9 \* 8

ld t1, 0(t0) # t1 = arr[9]

# arr[3]

li t0, 3

slli t0, t0, 3 # t0 = 3 \* 8

add t0, t0, s0 # t0 = arr + 3 \* 8

ld t2, 0(t0) # t2 = arr[3]

# arr[8]

li t0, 8

slli t0, t0, 3 # t0 = 8 \* 8

add t0, t0, s0 # t0 = arr + 8 \* 8

ld t3, 0(t0) # t3 = arr[8]

add t1, t1, t2 # t1 = arr[9] + arr[3]

add t1, t1, t3 # t1 = arr[9] + arr[3] + arr[8]

# Проверка условия (используем t1)

li t4, 411 # t4 = 411

ble t1, t4, then\_branch # if ((arr[9] + arr[3] + arr[8]) <= 411), then\_branch

# else branch

else\_branch:

# arr[i] = arr[i] | 69

slli t0, s1, 3 # t0 = i \* 8

add t0, t0, s0 # t0 = &arr[i]

ld t5, 0(t0) # t5 = arr[i]

li t6, 69 # t6 = 69

or t5, t5, t6 # t5 = arr[i] | 69

sd t5, 0(t0) # arr[i] = t5

j loop\_next

# then branch

then\_branch:

# arr[i] = arr[i-1] + 64

# Обработка arr[0] - присваиваем 64

beqz s1, handle\_arr\_0

slli t0, s1, 3 # t0 = i\*8

add t0, t0, s0 # t0 = &arr[i]

ld s3, -8(t0) # s3 = arr[i-1]

j calculate\_new\_value

handle\_arr\_0:

li s3, 0

calculate\_new\_value:

addi s3, s3, 64 # s3 = arr[i-1] + 64

slli t0, s1, 3 # t0 = i\*8

add t0, t0, s0 # t0 = &arr[i]

sd s3, 0(t0) # arr[i] = arr[i-1] + 64

loop\_next:

# i++

addi s1, s1, 1 # i++

j loop\_start

loop\_end:

end:

ld ra, 0(sp) # Восстанавливаем ra

ld s0, 8(sp) # Восстанавливаем s0

ld s1, 16(sp) # Восстанавливаем s1

ld s2, 24(sp) # Восстанавливаем s2

ld s3, 32(sp) # Восстанавливаем s3

addi sp, sp, 40 # Освобождение стека

ret

**ПРИЛОЖЕНИЕ Б**

**ТЕСТИРОВАНИЕ**

| **in** | **out** |
| --- | --- |
| 12, 323, 44, 55, 11, 65, 12, 4, 5, 6 | 12 12 12 12 12 12 12 12 12 12 |
| 1, 98, 50, 174, 64, 100, 28, 69, 14, 164, 82, 171 | 49 49 49 49 49 49 49 76 41 39 |
| 123, 35, 156, 21, 51, 56, 180, 72, 100, 53 | 123 123 123 123 123 123 123 123 123 123 |
| 76, 110, 152, 180, 135, 11, 72, 152, 193, 104 | 76 76 76 76 76 76 76 76 76 76 |