**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.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Шубин П.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). Порядок операций в формуле соответствует порядку операций в языке Си.

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. Размер каждого элемента массива - 8 байт (dword).

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[5] & arr[5] + arr[0]) <= 817)

ТО (arr[i] = arr[i - 1] | 16)

ИНАЧЕ (arr[i] = arr[i] - 100)

Ваш seed = 5405373824

**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**

Исходное задание было приведено ранее.

* Инициализация данных:

В начале процедуры solution устанавливаются начальные значения:

* + t0 — счётчик итераций (индекс i) устанавливается в 0.
  + t4 — значение для arr[-1] (предыдущий элемент массива) инициализируется как 0.
  + s4 — пороговое значение (threshold) задаётся как 817.
* Вычисление условия:
* Загружаются элементы массива:
  + arr[5] загружается в регистр s2 (вычисляется с помощью смещения 5 \* 8 = 40).
  + arr[0] загружается в регистр s3.
* Производятся арифметические операции:
  + s5 = arr[5] + arr[0].
  + s5 обновляется как побитовое И с arr[5] (т.е. arr[5] & (arr[5] + arr[0])).
* Запуск цикла:
* Основной цикл начинается с проверки условия:
  + Если s5 <= threshold, переход к ветке first.
  + Иначе выполняется ветка second.
* Ветка first:

Выполняется операция для обновления текущего элемента массива:

* + Вычисляется адрес arr[i] с помощью смещения (i \* 8).
  + Значение arr[i] обновляется как arr[i-1] | 16 (побитовое ИЛИ предыдущего элемента с числом 16).

После этого выполняется переход на метку update.

* Ветка second:

Выполняется операция для уменьшения текущего элемента массива:

* + Вычисляется адрес arr[i].
  + Загружается значение текущего элемента arr[i].
  + Из него вычитается 100, и результат сохраняется в регистре.

После этого выполняется переход на метку update.

* Обновление данных (update):

Записывается новое значение arr[i] в массив. Переменная arr[-1] (регистр t4) обновляется значением текущего элемента arr[i]. Увеличивается счётчик t0 (индекс i).

Проверяется условие завершения:

* + Если i >= size (t0 >= a2), выполняется переход к метке finish.
  + В противном случае возвращаемся к началу цикла.
* Завершение работы (finish):

После обработки всех элементов массива выполняется команда ret для завершения функции.

Исходный код программы см. в **ПРИЛОЖЕНИИ А .**

Тестирование см. в **ПРИЛОЖЕНИИ Б**.

**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:

li t0, 0 # Инициализация индекса i

li t4, 0 # Устанавливаем arr[-1] = 0

li s4, 817 # Устанавливаем threshold = 817

# Загружаем arr[5] и arr[0] для вычисления условия

ld s2, 40(a1) # arr[5], так как 5 \* 8 = 40

ld s3, 0(a1) # arr[0]

add s5, s2, s3 # s5 = arr[5] + arr[0]

and s5, s5, s2 # s5 = arr[5] & (arr[5] + arr[0])

start:

# Проверяем условие s5 <= threshold

ble s5, s4, first # Если s5 <= 817, выполняем ветку first

j second # Иначе выполняем ветку second

first:

# arr[i] = arr[i-1] | 16

slli t6, t0, 3 # Вычисление смещения для arr[i] (i \* 8)

add t6, a1, t6 # Адрес arr[i]

ori a3, t4, 16 # a3 = arr[i-1] | 16

j update # Переход к записи данных

second:

# arr[i] = arr[i] - 100

slli t6, t0, 3 # Вычисление смещения для arr[i] (i \* 8)

add t6, a1, t6 # Адрес arr[i]

ld a3, 0(t6) # Загружаем arr[i]

li t1, 100 # Загружаем 100

sub a3, a3, t1 # a3 = arr[i] - 100

update:

# Записываем результат и обновляем arr[-1]

sd a3, 0(t6) # Сохраняем новое значение arr[i]

mv t4, a3 # Обновляем arr[-1] (t4 = arr[i])

addi t0, t0, 1 # Увеличиваем индекс i (i = i + 1)

bge t0, a2, finish # Если i >= size, завершаем

j start # Иначе обрабатываем следующий элемент

finish:

ret # Завершение функции

**ПРИЛОЖЕНИЕ Б**

**ТЕСТИРОВАНИЕ**

|  |  |
| --- | --- |
| **in** | **out** |
| 100,201,130,456,50,60,90,80,45,43 | 16 201 16 456 16 60 16 80 16 43 |
| 10,10,10,10,10,10,10,10 | -90 10 -90 10 -90 10 -90 10 |
| 1,2,3,4,5,6,7,8,9 | -99 2 -97 4 -95 6 -93 8 -91 |