**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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Кулач Д.В. |
| 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[3] + arr[3]) < 438)

ТО (arr[i] = arr[i - 1] + 11)

ИНАЧЕ (arr[i] = arr[i] | 46)

Ваш seed = 5318454656

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. Исходное задание было приведено ранее.

2. Для красоты и наглядности были определены несколько строк и: написаны несколько функций для вывода стартового и обработанного массива.

3. Процедура *main.*  Загружает в регистры *a1, a2* адрес начала массива и его размер соответственно. Далее выводит вспомогательные сообщения вызовом соответствующих процедур. После этого вызывает процедуру *solution* для обработки и выводит измененный массив. Завершает выполнение.

4. Процедура *solution.* Загружает в регистры все нужные счетчики и значение *treshold*.Далее запускает цикл, в котором сначала высчитывается адрес текущего элемента массива, затем актуальное значение для условия. Поскольку массив с каждой итерацией изменяется, нужно каждый раз расчитывать это значение заново.   
Дальше в зависимости от значения происходит переход на метку *second* или работа с логикой метки *first.*

*First:*

Из регистра a3, в котором лежит текущее значение элемента, вычитает 4. Совершает переход на метку *update*.

*Second:*

На момент входа в метку в регистре t4 лежит значение предыдущего элемента. В регистр a3 сохраняется операция «логическое ИЛИ» содержимого t3 и числа 49. Переходит на *update*

*update:*

Командой *sw* загружает в массив содержимое регистра *a3*. Помещает в регистр *t4* значение *a3*, обновляя его. Увеличивает счетчик *t0*. Проверяет, что счетчик меньше размера массива и если это не так, переходит на метку *finish*, которая применяет *ret*. Иначе сдвигает указатель на текущий элемент на 4 байта и переходит на следующую итерацию. 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**

.data

array: .word 100,201,130,456,50,60,90,80,45,43 # Массив данных (10 элементов)

size: .word 10 # Размер массива

help1: .string "start array: " # Строка, которая будет выведена перед печатью массива

help2: .string "finish array: " # Строка, которая будет выведена после обработки массива

newline: .string "\n"

.text

.globl main

# Главная функция

main:

la a1, array # Загружаем адрес массива в a1 (начало массива)

lw a2, size # Загружаем размер массива в a2

la a0, help1 # Загружаем строку "start array: " в a0

la a0, help2 # Загружаем строку "finish array: " в a0

call solution # Вызываем функцию обработки массива (solution)

li a7, 10 # Загружаем код системного вызова для завершения программы

ecall # Выполняем системный вызов (завершаем программу)

.globl solution

solution:

li t0, 0 # Инициализация индекса i

li t4, 0 # Устанавливаем arr[-1] = 0

li s4, 438 # Устанавливаем threshold = 438

# Загружаем arr[5] и arr[3] для вычисления условия

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

ld s3, 24(a1) # arr[3], так как 3 \* 8 = 24

add s5, s3, s3 # s5 = arr[3] + arr[3]

and s5, s5, s2 # s5 = arr[5] & (arr[3] + arr[3])

start:

# Проверяем условие s5 < threshold

blt s5, s4, first # Если s5 < 438, выполняем ветку first

j second # Иначе выполняем ветку second

first:

# arr[i] = arr[i-1] + 11

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

add t6, a1, t6 # Адрес arr[i]

addi a3, t4, 11 # a3 = arr[i-1] + 11

j update # Переход к записи данных

second:

# arr[i] = arr[i] | 46

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]

ori a3, a3, 46 # a3 = arr[i] | 46

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, 900, 80, 45, 43 | 114, 247, 176, 502, 54, 62, 950, 62, 91, 61 |
| 5, 6, 7, 8, 9, 10, 11, 12 | 51, 54, 55, 62, 61, 62, 57, 62 |
| 1, 1, 1, 1, 1, 1, 1, 1 | 47, 47, 47, 47, 47, 47, 47, 47 |
| 400, 800, 200, 150, 100, 50, 10, 5 | 446, 854, 246, 182, 146, 62, 56, 51 |
| 10, 20, 30, 40, 50, 60, 70, 80 | 56, 54, 62, 62, 62, 62, 78, 78 |