**­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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ижболдин А.В. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Молодцев Д.А |

Санкт-Петербург

2024

**Цель работы**

Разработка программы преобразования данных для приобретения практических навыков программирования на языке ассемблера. 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[0] + arr[0] & arr[4]) <= 436)

ТО (arr[i] = arr[i - 1] | 61)

ИНАЧЕ (arr[i] = arr[i] & 45)

Ваш seed = 4739956736

**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. Был написан вывод исходного массива и измененного за это отвечает процедура print\_array.

3. Процедура **main***.* В процедуре main происходит загрузка адреса массива в a1 и загрузка счетчика в a2, после этого вызываются процедуры печати исходного массива, вызов процедуры calculation - изменения массива, вывод измененного массива.

4. Процедура **solution**. Загрузка a1 в t0, a2 в t1.

Проверка условия (метка **if**).

Загружаются два элемента массива:

* arr[0] в регистр t2.
* arr[4] в регистр t3 (с использованием смещения 4 \* memsize).

Выполняются следующие операции:

* Значение arr[0] удваивается (arr[0] + arr[0]).
* Результат побитово "И" сравнивается с arr[4].
* Итоговое значение хранится в t3.
* Константа 436 загружается в t4.

Проводится сравнение: если (arr[0] + arr[0]) & arr[4] <= 436, то выполняется блок if\_true. Иначе происходит переход к false\_loop.

Условие ложно (**false\_loop**)

Эта часть выполняется, если условие ложно (то есть t3 > 436):

Для каждого элемента массива выполняется: arr[i] = arr[i] & 45

Загружается текущий элемент arr[i]. Выполняется побитовая операция "И" с числом 45. Результат сохраняется обратно в текущий элемент массива.

Указатель на текущий элемент сдвигается на следующий элемент (t0 += memsize). Счётчик элементов уменьшается (t1 -= 1).

Если счётчик элементов t1 равен 0, выполнение завершается (переход к метке end). Иначе цикл возвращается к началу блока if.

Условие истинно (**if\_true**)

Эта часть выполняется, если условие истинно (то есть t3 <= 436):

Проверяется, находится ли текущий элемент в начале массива:

Если текущий элемент — первый (t0 == a1), то значение первого элемента устанавливается в 61. Побитовое "ИЛИ" с 61 гарантирует, что результат всегда равен 61.

Указатель на текущий элемент сдвигается на следующий элемент массива.

Счётчик уменьшается (t1 -= 1). После идет переход на цикл истинного условия.

Цикл истинной ветки (**true\_loop**). Этот цикл выполняется для всех остальных элементов массива:

Для каждого элемента:

Загружается предыдущий элемент массива arr[i-1].

Выполняется побитовая операция "ИЛИ" с числом 61: arr[i] = arr[i-1] | 61

Результат сохраняется в текущий элемент массива arr[i].

Указатель на текущий элемент сдвигается на следующий элемент массива (t0 += memsize).

Счётчик уменьшается (t1 -= 1).

Если счётчик элементов t1 равен 0, выполнение завершается (переход к метке end). Иначе программа возвращается к началу блока if.

Возвращение из процедуры вычисления(метка end) - выполняется инструкция ret, которая завершает выполнение функции и возвращает управление вызывающей программе.

Разработанный код приведен в Приложении А, тестирование в Приложение Б.

**Вывод**

В ходе выполнения лабораторной работы были изучены методы адресации в ассемблере RISC-V. Также были приобретены практические навыки их применения при написании программы, изменяющей массив в зависимости от заданного условия. Полученные знания и навыки позволят более эффективно разрабатывать и оптимизировать программы на языке ассемблера RISC-V.

**ПРИЛОЖЕНИЕ А**

Файл: main.s

.data

.equ memsize, 4

arr\_after: .string "Array after: "

arr\_before: .string "Array before: "

size: .word 6

arr: .word 123,15,22,-15,30,40

.text

main:

la a1, arr # Загружаем адрес начала массива в a1

lw a2, size # Загружаем размер массива в a2

la a0, arr\_before # Загружаем адрес строки arr\_before в a0

call print\_array # Вызываем функцию печати массива

call solution # Вызываем функцию обработки массива

la a0, arr\_after # Загружаем адрес строки arr\_after в a0

call print\_array # Вызываем функцию печати массива

li a7, 10 # Загружаем номер функции завершения программы

ecall # Вызываем системную функцию

.globl solution

solution:

mv t0, a1 # Загружаем адрес массива в t0

mv t1, a2 # Загружаем размер массива в t1

if:

lw t2, 0(a1) # Загружаем arr[0] в t2

lw t3, 4\*memsize(a1) # Загружаем arr[4] в t3

add t2, t2, t2 # arr[0] + arr[0]

and t3, t2, t3 # (arr[0] + arr[0]) & arr[4]

li t4, 436

ble t3, t4, if\_true #ЕСЛИ ((arr[0] + arr[0] & arr[4]) <= 436)

false\_loop: #ИНАЧЕ (arr[i] = arr[i] & 45)

beqz t1, end # Если количество элементов = 0, переходим к end

lw t2, 0(t0) # Считываем arr[i]

andi t2, t2, 45 # arr[i] & 45

sw t2, 0(t0) # Записывам arr[i] = arr[i] & 45

addi t0, t0, memsize # Переходим на след. элемент

addi t1, t1, -1 # Увеличиваем счетчик

j if # Переходим к проверке условия

if\_true: #ТО (arr[i] = arr[i - 1] | 61)

bne t0, a1, true\_loop # Если не начало массива переходим к циклу

li t2, 61 # arr[0] = 61 - всегда, т.к. 0 | 61 = 61

sw t2, 0(t0) # Записываем arr[0] = 61

addi t0, t0, memsize # Переходим на след. элемент

addi t1, t1, -1 # Уменьшаем счетчик

true\_loop:

beqz t1, end # Если количество элементов = 0, переходим к end

lw t2, -memsize(t0) # Считываем arr[i-1]

ori t2, t2, 61 # arr[i] = arr[i-1] | 61

sw t2, 0(t0) # Записываем arr[i]

addi t0, t0, memsize # Переходим на след. элемент

addi t1, t1, -1 # Уменьшаем счетчик

j if # Переходим к проверке условия

end:

ret # Возвращаемся из функции solution

print\_array:

li a7, 4 # Устанавливаем функцию вывода строки

ecall # Вызываем системную функцию

mv t0, a1 # Записываем в t0 адрес начала массива

mv t1, zero # Обнуляем счетчик

print\_array\_loop:

beq t1, a2, print\_array\_end # Если счетчик = количество элементов, то переходим к print\_array\_end

li a7, 1 # Устанавливаем функцию вывода числа

lw a0, 0(t0) # Загружаем число для вывода в a0

ecall # Вызываем системную функцию

li a7, 11 # Устанавливаем функцию вывода символа

li a0, 32 # Загружаем символ ' ' для вывода в a0

ecall # Вызываем системную функцию

addi t0, t0, memsize # Переходим на след. элемент

addi t1, t1, 1 # Увеличиваем счетчик

j print\_array\_loop # Переход к началу цикла

print\_array\_end:

li a7, 11 # Устанавливаем функцию вывода символа

li a0, 10 # Загружаем символ '\n' для вывода в a0

ecall # Вызываем системную функцию

ret # Возвращаемся из функции print\_array

**ПРИЛОЖЕНИЕ Б**

**ТЕСТИРОВАНИЕ**

|  |  |
| --- | --- |
| **input** | **output** |
| 123 15 22 -15 30 40 | 61 61 61 61 61 61 |
| 666 666 666 666 666 666 | 61 61 61 61 61 61 |
| 555 12 34 2000 2000 555 | 41 61 61 61 61 61 |
| 1013 1234 423 1445 4345 8888 9282 | 61 61 61 61 61 61 61 |
| -81 230 809 565 877 1 | 45 61 61 61 61 61 |