**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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Потоцкий С.С. |
| 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[0]) > 647)

ТО (arr[i] = arr[i - 1] & 79)

ИНАЧЕ (arr[i] = arr[i] ^ 28)

seed = 8131831136

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. **Структура программы**. Для улучшения читаемости и повторного использования кода были добавлены вспомогательные строки и процедуры:
   * **Строки**: str1 (“Start array:”), str2 (“Finish array:”) для отображения сообщений перед выводом массивов.
   * **Функции**: процедуры для вывода строк (print\_string), чисел (print\_number) и элементов массива (print\_arr).
3. **Процедура main**:
   * Загружает в регистры a1 и a2 адрес начала массива и его размер соответственно.
   * Выводит сообщение “Start array:” и исходный массив с помощью процедур print\_string и print\_arr.
   * Вызывает процедуру solution для обработки массива.
   * Выводит сообщение “Finish array:” и измененный массив аналогичным образом.
   * Завершает выполнение программы с помощью системного вызова.
4. **Процедура solution**:
   * Загружает в рабочие регистры адрес массива (a3), счетчик (a4), инициализирует пороговое значение (a7 = 647).
   * Выполняет обработку массива в цикле. На каждой итерации вычисляет:
     + Условие: (arr[9] - arr[3] - arr[0]).
     + Адрес текущего элемента массива и его значение.
   * Логика обработки:
     + Если вычисленное значение условия больше 647 (“greater”): элемент массива заменяется на результат операции побитового И над предыдущим элементом.
     + Если условие меньше или равно 647 (“less”): текущий элемент заменяется результатом побитовой операции XOR с числом 28.
   * Для первого элемента массива (“first\_elem”) используется специальный случай: он заменяется результатом побитовой операции И с числом 79.
   * Сохраняет обработанное значение обратно в массив, увеличивает счетчик и проверяет условие окончания цикла.
   * Завершает выполнение процедуры, возвращаясь в main.
5. **Процедура print\_arr**:
   * Печатает элементы массива поочередно.
   * Для каждого элемента массива выполняет системный вызов печати числа, добавляет пробел и переходит к следующему элементу.
   * По окончании вывода массива печатает символ новой строки.
6. **Процедура print\_string**:
   * Выводит строку, адрес которой передается через регистр a0, с помощью системного вызова.

### Заключение

Программа успешно обрабатывает массив данных в соответствии с заданным условием. Реализованные процедуры обеспечивают43, 48, 61, 10, 23, 94, 81, 68, 127, 114 структурированность и читаемость кода, а использование меток для управления логикой позволяет легко адаптировать алгоритм под различные задачи.

Тестирование

|  |  |
| --- | --- |
| Входной массив | Выходной массив |
| 7, 16, 149, 999, 37, 141, 543, 777, 9, 4, 1, 2 | 27, 12, 137, 1019, 57, 145, 515, 789, 21, 24, 29, 30 |
| 500, 600, 700, 800, 100, 200, 300, 400, 900, 1000 | 488, 580, 672, 828, 120, 212, 304, 396, 920, 1012 |
| 55, 44, 33, 22, 11, 66, 77, 88, 99, 110 | 43, 48, 61, 10, 23, 94, 81, 68, 127, 114 |
| 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 | 22, 8, 2, 52, 46, 32, 90, 76, 70, 120 |

**Вывод**

В процессе выполнения лабораторной работы были изучены различные способы адресации в архитектуре RISC-V. Получены практические навыки их применения при разработке программы, выполняющей обработку массива данных в соответствии с заданными условиями.

**Приложение А**

**Код программы**

.data

array: .word 7, 16, 149, 999, 37, 141, 543, 777, 9, 4, 1, 2 # Массив данных (12 элементов)

size: .word 12 # Размер массива

str1: .string "Start array: " # Строка перед печатью массива

str2: .string "Finish array: " # Строка после обработки массива

newline: .string "\n"

.text

.globl main

main:

la a1, array # Загружаем адрес массива в a1

lw a2, size # Загружаем размер массива в a2

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

call print\_string # Выводим строку

jal print\_arr # Печатаем исходный массив

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

call print\_string # Выводим строку

call solution # Вызываем функцию обработки массива

call print\_arr # Печатаем изменённый массив

li a7, 10 # Завершение программы

ecall

.globl solution

solution:

# a1 - адрес массива

# a2 - количество элементов

mv a3, a1 # сохраняем исходный адрес массива

li a4, 0 # счетчик i

loop:

# Вычисляем условие (arr[9] - arr[3] - arr[0])

lw a5, 36(a3) # arr[9] (36 = 9\*4)

lw a6, 12(a3) # arr[3] (12 = 3\*4)

lw a7, 0(a3) # arr[0]

sub a5, a5, a6 # arr[9] - arr[3]

sub a5, a5, a7 # (arr[9] - arr[3] - arr[0])

# Адрес текущего элемента = база + i\*4

mv a6, a4 # копируем счетчик

slli a6, a6, 2 # умножаем на 4

add a6, a3, a6 # адрес текущего элемента

li a7, 647

bgt a5, a7, greater # если условие > 647

less:

lw a5, 0(a6) # загружаем текущий элемент

xori a5, a5, 28 # arr[i] ^ 28

j store

greater:

beqz a4, first\_elem # если i == 0

lw a5, -4(a6) # загружаем предыдущий элемент

andi a5, a5, 79 # arr[i-1] & 79

j store

first\_elem:

li a5, 0 # arr[-1] = 0

andi a5, a5, 79 # 0 & 79

store:

sw a5, 0(a6) # сохраняем результат

addi a4, a4, 1 # увеличиваем счетчик

bne a4, a2, loop # продолжаем если не закончили

ret

# Процедура для вывода массива

print\_arr:

li t1, 0 # Индекс массива i = 0

print\_loop:

bge t1, a2, print\_end # Если i >= size, завершаем печать массива

slli t3, t1, 2 # t3 = i \* 4 (смещение)

add t3, t3, a1 # t3 = адрес arr[i]

lw t2, 0(t3) # t2 = arr[i]

mv a0, t2 # Подготовка к печати числа

li a7, 1 # Системный вызов печати числа

ecall

li a0, 32 # Печать пробела

li a7, 11

ecall

addi t1, t1, 1 # i+1

j print\_loop

print\_end:

la a0, newline

li a7, 4

ecall

ret

# Процедура для вывода строки

print\_string:

li a7, 4

ecall

ret

# Процедура для вывода числа

print\_number:

li a7, 1

ecall

la a0, newline

li a7, 4

ecall

ret