**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 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Трунов Б.Г. |
| 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:**

1. Написать программу, которая с использованием 4 режимов адресации: регистрового, непосредственного, базового и относительного к счётчику команд реализует вычисление выражения.

**Вариант:**

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

ЕСЛИ ((arr[2] - arr[5] | arr[9]) <= 566)

ТО (arr[i] = arr[i - 1] + 42)

ИНАЧЕ (arr[i] = arr[i] + 56)

**Основные теоретические положения:**

**Базовое подмножество команд** использует следующий набор регистров: специальный регистр x0 (zero), содержащий значение 0, всегда получаемое при чтении из этого регистра, 31 целочисленный регистр общего назначения (x1—x31), регистр счётчика команд PC (используется только косвенно).

**Используемые базовые вычислительные инструкции:**

- арифметические add, sub;

- логические and, or, xor;

**Используемые базовые инструкции управления программой:**

1) инструкции условного перехода

beq (=), bne (!=), blt (<), bge (≥), bltu (<u), bgeu(≥u)

2) инструкции безусловного перехода

- безусловный переход с сохранением адреса возврата

jal ra, label (jump and link)

**Регистровая адресация**

При регистровой адресации регистры используются для всех  
операндов-источников и операндов-назначений (иными словами – для всех операндов и результата). Все инструкции типа R используют именно такой режим адресации.

add rd,rs1,rs2 # rd = rs1 + rs2

**Непосредственная адресация**

При непосредственной адресации в качестве операндов наряду с регистрами используют константы (непосредственные операнды). Этот режим адресации используют некоторые инструкции типа I, такие как сложение с 12-битной константой (addi) и логическая операция andi.  
 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

**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]

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

**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 типа B и jal (переход и связывание) типа J используют для смещения 13- и 21-битные константы.

beq rs1,rs2,imm # if(rs1 == rs2) PC += imm

jal rd,imm #rd = PC+4; PC += imm

**Выполнение работы**

В самом начале программы задаются константы и помещаются в регистры. С помощью системных вызовов выводятся необходимые строки, символы и значения.

1. Формирование массива:

Массив инициализируется в сегменте .data, в котором указываются размеры массива и сами значения

2. Использование 4-х режимов адресации:

Используя **базовую адресацию**, инструкцией sw (запись в память) вычисленные значения элементов массива загружаются в память для инициализации массива. Также инструкция lw (считывание из памяти) для считывания значений элементов массива в регистры и последующего использования в вычислениях.

**Регистровая адресация** используется для того, чтобы производить необходимые арифметические и логические операции со значениями массива, загруженными в регистры, например, сложение(add), вычитание(sub) и побитовое И(and).

**Непосредственная адресация** используется тогда, когда в регистр a7 загружается номер системного вызова при выводе в консоль символов и значений.

**Адресация относительно счётчика команд** используется при выполнении условного перехода bge (больше или равно) при реализации заданного условия, а также безусловного перехода j, когда нужно перейти к завершению программы после того, как выполнены инструкции, необходимые в том случае, если заданное условие оказалось невыполненным.

Результаты тестирования представлены в таблице 1.

Исходный код разработанной программы представлен в Приложении.

**Тестирование**

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Массив | Ожидаемый результат | Полученный результат |
| 1 | 23, 33, 888, -10, 10, -400, -19, 20,-30,20 | 79, 89, 944, 46, 66, -344, 37, 76, 26, 76 | 79, 89, 944, 46, 66, -344, 37, 76, 26, 76 |
| 2 | 5,6,7,8,9,10,-11,-12,-13,-14,-15 | 42, 84, 126, 168, 210, 252, 294, 336, 378, 420,462 | 42, 84, 126, 168, 210, 252, 294, 336, 378, 420,462 |
| 3 | 1, 2, 2, 3, 4, 4, 5, 8, 8, 11 | 42, 84, 126, 168, 210, 252, 294, 336, 378, 420 | 42, 84, 126, 168, 210, 252, 294, 336, 378, 420 |

**Вывод**

В результате выполнения лабораторной работы была освоена работа с режимами адресации процессора RISC-V. Также по итогам работы была разработана программы преобразования данных для вычисления значения выражения в соответствии с заданным порогом.

**ПРИЛОЖЕНИЕ**

**Код программы**

.data

# моделируемые вычисления для вывода в консоль

array: .word 23, 33, 888, -10, 10, -400, -19, 20,-30,20 # Массив данных

size: .word 10 # Размер массива

cond1: .string "Condition:\nif ((arr[2] - arr[5]) | arr[9] <= 566) "

cond2: .string "\nthen: arr[i] = arr[i - 1] + 42\nelse: arr[i] = arr[i] + 56\n"

line\_before: .string "before proccesed array: " # Строка, которая будет выведена перед печатью массива

line\_after: .string "processed array: " # Строка, которая будет выведена после обработки массива

new\_line: .string "\n"

.text

.globl main

main:

# Печать условия

la a0, cond1 # Загрузка адреса строки cond1 в a0

li a7, 4 # Код системного вызова для печати строки

ecall #систем. вызов (а7)

# Печать условия 2

la a0, cond2 # Загрузка адреса строки cond2 в a0

li a7, 4 # Код системного вызова для печати строки

ecall #систем. вызов (а7)

la a1, array # Загружаем адрес начала массива в a1

lw a2, size # Загружаем размер массива в a2

la a0, line\_before # Загружаем строку "before proccesed array: " в a0

call print\_str # Выводим строку с помощью print\_str

jal print\_arr # Вызываем функцию для печати массива

la a0, line\_after # Загружаем строку "processed array: " в a0

call print\_str # Выводим строку с помощью print\_str

call solution # Вызываем функцию обработки массива

call print\_arr # Печатаем изменённый массив

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

ecall #систем. вызов (а7)

.globl solution

solution:

li t0, 0 # Инициализация счетчика t0 значением 0.

li t4, 0 # Инициализация переменной t4 значением 0.

loop:

bge t0, a2, end # Проверка: если t0 (счетчик цикла) больше или равен a2, переход к метке end (выход из цикла).

slli t5, t0, 2 # Логическая операция сдвига влево на 0 бит для t0, результат сохраняется в t5. Эффективно t5 будет равно t0.

add t6, a1, t5 # Сложение адреса a1 и t5. Результат (адрес элемента массива) сохраняется в t6.

lw a3, 0(t6) # Загрузка слова (4 байта) из памяти по адресу, указанному в t6, в регистр a3.

lw s4, 8(a1) # Загрузка слова из памяти по адресу a1 + 8 (предполагаем, что это второй элемент массива) в регистр s4.

lw s2, 20(a1) # Загрузка слова из памяти по адресу a1 + 20 в регистр s2.

lw s3, 36(a1) # Загрузка слова из памяти по адресу a1 + 36 в регистр s3.

neg s2,s2 # s2 = -arr[5]

add s5, s4, s2 # s5 = arr[2] - arr[5]

or s5, s5, s3 # s5 = arr[2] - arr[5] | arr[9]

li s2, 566 # Инициализация s2 значением 566. Это значение будет использоваться для сравнения позже.

ble s5, s2, do\_then # переход к метке do\_then если вычесленное значение в регистре s5 <= 566

do\_else:

li t1,56 # Инициализация t1 значением 56. Это значение будет добавлено к a3 в следующем шаге.

add a3, a3, t1 # Сложение значения из a3 и t1, результат сохраняется обратно в a3.

sw a3, 0(t6) # Сохранение обновленного значения a3 по адресу, указанному в t6.

j continue # Переход к метке continue, чтобы продолжить следующий шаг цикла.

do\_then:

li t1, 42 # Инициализация t1 значением 42. Это значение будет добавлено к t4 в следующем шаге.

add a3, t4, t1 # Сложение значения t4 и t1, результат сохраняется в a3.

sw a3, 0(t6) # Сохранение обновленного значения a3 по адресу, указанному в t6.

continue:

lw t4, 0(t6) # Загрузка слова из памяти по адресу, указанному в t6, в регистр t4. Получаем обновленное значение.

addi t0, t0, 1 # Увеличение счетчика t0 на 1 для перехода к следующему элементу цикла.

j loop # Переход к началу цикла (loop) для продолжения итераций.

end:

ret # Выход из функции

print\_arr:

li t1, 0 # Инициализация переменной t1, которая будет использоваться как индекс массива (i = 0)

print\_value:

bge t1, a2, print\_end # Проверяем условие: если индекс i больше или равен размеру массива , завершаем цикл

# Загружаем значение массива по индексу i в регистр t2

slli t3, t1, 2 # Умножаем индекс i на 4 (размер элемента int), результат помещаем в t3 (смещение для доступа)

add t3, t3, a1 # Вычисляем адрес элемента массива arr[i] и сохраняем его в t3

lw t2, 0(t3) # Загружаем значение элемента массива (arr[i]) в регистр t2

# Выполняем вывод значения на экран

mv a0, t2 # Перемещаем значение числа в регистр a0 (аргумент для системного вызова)

li a7, 1 # Устанавливаем код системного вызова для вывода числа (код 1)

ecall #систем. вызов (а7)

# Печатаем разделитель (пробел) после числа

li a0, 32 # Задаем код символа пробела (ASCII 32)

li a7, 11 # Устанавливаем код системного вызова для печати символа (код 11)

ecall #систем. вызов (а7)

addi t1, t1, 1 # Увеличиваем индекс i на 1 для перехода к следующему элементу массива (t1 = t1 + 1)

j print\_value # Переходим к началу цикла для печати следующего элемента массива

print\_end:

# Завершение печати массива, вывод новой строки

la a0, new\_line # Загружаем адрес строки новой строки в регистр a0

li a7, 4 # Устанавливаем код системного вызова для вывода строки (код 4)

ecall #систем. вызов (а7)

ret # Возвращаемся из процедуры print\_arr

# Процедура для вывода строки

print\_str:

li a7, 4 # Устанавливаем код системного вызова для вывода строки (код 4)

ecall #систем. вызов (а7)

ret # Возвращаемся из процедуры print\_string