**Лабораторная работа 2.**

**Изучение режимов адресации в ассемблере RISC-V.**

2.1. 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*.

2.2. Основные теоретические сведения

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 (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

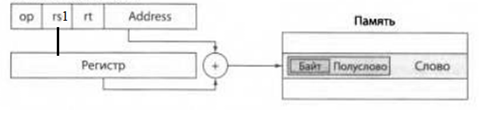
При использовании многоразрядных непосредственных операндов, если указанный в addi 12-битный непосредственный операнд отрицательный, старшая часть постоянного значения в lui должна быть увеличена на единицу. Помните, что знак addi расширяет 12-битное непосредственное значение, поэтому отрицательное непосредственное значение будет содержать все единицы в своих старших 20 битах. Поскольку в дополнительном коде все единицы означают число –1, добавление числа, у которого все разряды установлены в 1, к старшим разрядам непосредственного операнда приводит к вычитанию 1 из этого числа. Пример иллюстрирует ситуацию, когда мы хотим в s2 получить постоянное значение 0xFEEDA987:

lui s2, 0xFEEDB # s2 = 0xFEEDB000 (число, которое нужно записать в старшие 20 разрядов (0xFEEDA), предварительно увеличено на 1)

addi s2, s2, −1657 # s2 = 0xFEEDA987 (0x987 – это 12-битное представление числа -1657)(0xFEEDB000 + 0xFFFFF987 = 0xFEEDA987)

*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 командв 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. Самые старшие значимые биты смещения располагаются в 12- и 20-битных полях инструкций типа B и J. Наименьший значащий бит смещения всегда равен 0, поэтому он отсутствует в инструкции.

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

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

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. Но если адрес перехода хранится в регистре, вы можете сделать переход на любой адрес (инструкция jalr типа I).

jalr rd, imm (rs1) # rd = PC + 4, PC = rs1 + imm

Большая разница состоит в том, что переход JALR не происходит относительно PC. Вместо этого он происходит относительно rs1

Инструкция auipc типа U (сложить старшие разряды константы смещения с PC) также использует адресацию относительно счетчика команд.

auipc rd,imm # rd = PC + (imm << 12)

auipc s3, 0xABCDE # s3 = PC + 0xABCDE000

2.3. Задание к лабораторной работе

1. Для заданного набора констант

|  |  |
| --- | --- |
| Константа | Значение |
| a | [Сумма цифр студ. билета] |
| b | [Количество букв в фамилии] |
| c | [Количество букв в полном имени] |

сформировать массив array из 10 элементов, в котором

arr[0] = a+b+c

array[i+1] = arr[i] + a + b - c

Доступ к массиву (инициализация, запись, чтение) должен выполняться из памяти.

2. Написать программу, которая с использованием 4 режимов адресации: регистрового, непосредственного, базового и относительного к счетчику команд реализует вычисление выражения, выбираемого из таблицы 1 в соответствии с номером студента в списке группы

Таблица 1

|  |  |  |
| --- | --- | --- |
| Номер | Вычисляемое выражение | Используемые регистры |
| 1 | ЕСЛИ (arr[0] + arr[7] + arr[1] <= threshold) ТО (res1 = arr[0] + arr[1]) ИНАЧЕ (res2 = arr[5] ^ b) | threshold -> s1  res1 -> t7  res2 -> a2 |
| 2 | ЕСЛИ (arr[5] + arr[1] + arr[9] == threshold) ТО (res1 = arr[6] ^ arr[2]) ИНАЧЕ (res2 = arr[5] + a) | threshold -> s3  res1 -> t1  res2 -> a6 |
| 3 | ЕСЛИ (arr[4] + arr[5] + arr[2] < threshold) ТО (res1 = arr[8] - arr[4]) ИНАЧЕ (res2 = arr[5] & c) | threshold -> s7  res1 -> a3  res2 -> t2 |
| 4 | ЕСЛИ (arr[8] + arr[5] + arr[7] != threshold) ТО (res1 = arr[7] | arr[4]) ИНАЧЕ (res2 = arr[8] - b) | threshold -> s6  res1 -> a3  res2 -> t7 |
| 5 | ЕСЛИ (arr[6] + arr[3] + arr[8] == threshold) ТО (res1 = arr[5] + arr[4]) ИНАЧЕ (res2 = arr[1] ^ c) | threshold -> s3  res1 -> t1  res2 -> a4 |
| 6 | ЕСЛИ (arr[2] + arr[0] + arr[4] < threshold) ТО (res1 = arr[7] | arr[9]) ИНАЧЕ (res2 = arr[4] + a) | threshold -> t3  res1 -> a5  res2 -> s2 |
| 7 | ЕСЛИ (arr[1] + arr[7] + arr[2] < threshold) ТО (res1 = arr[2] - arr[3]) ИНАЧЕ (res2 = arr[5] | a) | threshold -> t4  res1 -> s2  res2 -> a5 |
| 8 | ЕСЛИ (arr[9] + arr[4] + arr[3] != threshold) ТО (res1 = arr[6] & arr[4]) ИНАЧЕ (res2 = arr[2] | b) | threshold -> t6  res1 -> a7  res2 -> s2 |
| 9 | ЕСЛИ (arr[3] + arr[6] + arr[0] >= threshold) ТО (res1 = arr[5] - arr[9]) ИНАЧЕ (res2 = arr[4] ^ c) | threshold -> s1  res1 -> t6  res2 -> a2 |
| 10 | ЕСЛИ (arr[7] + arr[4] + arr[1] <= threshold) ТО (res1 = arr[5] | arr[7]) ИНАЧЕ (res2 = arr[8] | a) | threshold -> a6  res1 -> t1  res2 -> s5 |
| 11 | ЕСЛИ (arr[5] + arr[3] + arr[9] <= threshold) ТО (res1 = arr[3] ^ arr[6]) ИНАЧЕ (res2 = arr[2] + c) | threshold -> a4  res1 -> t7  res2 -> s5 |
| 12 | ЕСЛИ (arr[1] + arr[2] + arr[3] == threshold) ТО (res1 = arr[3] ^ arr[1]) ИНАЧЕ (res2 = arr[9] ^ a) | threshold -> s2  res1 -> a6  res2 -> t4 |
| 13 | ЕСЛИ (arr[6] + arr[1] + arr[5] == threshold) ТО (res1 = arr[2] & arr[5]) ИНАЧЕ (res2 = arr[9] | a) | threshold -> a6  res1 -> s1  res2 -> t7 |
| 14 | ЕСЛИ (arr[9] + arr[3] + arr[2] > threshold) ТО (res1 = arr[0] - arr[1]) ИНАЧЕ (res2 = arr[2] - c) | threshold -> a1  res1 -> t6  res2 -> s4 |
| 15 | ЕСЛИ (arr[9] + arr[0] + arr[1] != threshold) ТО (res1 = arr[8] - arr[9]) ИНАЧЕ (res2 = arr[4] - a) | threshold -> a5  res1 -> t6  res2 -> s3 |
| 16 | ЕСЛИ (arr[4] + arr[1] + arr[0] >= threshold) ТО (res1 = arr[0] + arr[5]) ИНАЧЕ (res2 = arr[2] & a) | threshold -> t4  res1 -> s7  res2 -> a6 |
| 17 | ЕСЛИ (arr[3] + arr[4] + arr[1] == threshold) ТО (res1 = arr[4] + arr[0]) ИНАЧЕ (res2 = arr[6] & a) | threshold -> a6  res1 -> s7  res2 -> t5 |
| 18 | ЕСЛИ (arr[2] + arr[5] + arr[1] < threshold) ТО (res1 = arr[3] & arr[8]) ИНАЧЕ (res2 = arr[1] & a) | threshold -> t1  res1 -> s2  res2 -> a3 |
| 19 | ЕСЛИ (arr[6] + arr[7] + arr[5] < threshold) ТО (res1 = arr[8] & arr[9]) ИНАЧЕ (res2 = arr[1] & c) | threshold -> s1  res1 -> t7  res2 -> a3 |
| 20 | ЕСЛИ (arr[4] + arr[2] + arr[1] > threshold) ТО (res1 = arr[5] - arr[4]) ИНАЧЕ (res2 = arr[6] + c) | threshold -> t1  res1 -> s7  res2 -> a5 |
| 21 | ЕСЛИ (arr[1] + arr[9] + arr[4] <= threshold) ТО (res1 = arr[1] & arr[5]) ИНАЧЕ (res2 = arr[8] | a) | threshold -> a3  res1 -> s5  res2 -> t4 |
| 22 | ЕСЛИ (arr[0] + arr[2] + arr[5] <= threshold) ТО (res1 = arr[9] - arr[6]) ИНАЧЕ (res2 = arr[4] | c) | threshold -> s3  res1 -> t2  res2 -> a4 |
| 23 | ЕСЛИ (arr[2] + arr[9] + arr[6] > threshold) ТО (res1 = arr[4] | arr[7]) ИНАЧЕ (res2 = arr[0] | c) | threshold -> t6  res1 -> s4  res2 -> a5 |
| 24 | ЕСЛИ (arr[5] + arr[7] + arr[2] < threshold) ТО (res1 = arr[7] | arr[5]) ИНАЧЕ (res2 = arr[9] & c) | threshold -> t3  res1 -> a5  res2 -> s7 |
| 25 | ЕСЛИ (arr[4] + arr[1] + arr[5] != threshold) ТО (res1 = arr[3] | arr[1]) ИНАЧЕ (res2 = arr[7] - c) | threshold -> s5  res1 -> t1  res2 -> a4 |
| 26 | ЕСЛИ (arr[6] + arr[7] + arr[3] <= threshold) ТО (res1 = arr[5] & arr[4]) ИНАЧЕ (res2 = arr[2] ^ a) | threshold -> t5  res1 -> s2  res2 -> a6 |
| 27 | ЕСЛИ (arr[2] + arr[5] + arr[9] > threshold) ТО (res1 = arr[7] & arr[0]) ИНАЧЕ (res2 = arr[1] | a) | threshold -> a2  res1 -> s7  res2 -> t5 |
| 28 | ЕСЛИ (arr[5] + arr[0] + arr[1] != threshold) ТО (res1 = arr[9] & arr[7]) ИНАЧЕ (res2 = arr[8] + c) | threshold -> a1  res1 -> s2  res2 -> t4 |
| 29 | ЕСЛИ (arr[5] + arr[0] + arr[4] != threshold) ТО (res1 = arr[5] | arr[3]) ИНАЧЕ (res2 = arr[4] & c) | threshold -> a3  res1 -> s4  res2 -> t6 |
| 30 | ЕСЛИ (arr[2] + arr[7] + arr[6] == threshold) ТО (res1 = arr[8] | arr[6]) ИНАЧЕ (res2 = arr[7] + b) | threshold -> s4  res1 -> t1  res2 -> a7 |

Здесь threshold – заданный порог.

**Вопросы для защиты лабораторной работы.**

1.Что такое регистровый файл?

2.В чем отличие, удобство и проблемы регистровой и непосредственной

адресаций ?

3. В каких командах обычно применяется непосредственная адресация?

4. Как реализуется непосредственная адресация с использованием больших констант (длиной более 12 бит) ?

5. Как реализуются команды чтения данных из памяти в регистры ?

6. Как реализуются команды записи данных в память из регистров ?

7. В чем состоит назначение и особенность базовой адресации ?

8. Приведите пример формирования массива натуральных чисел

от 1 до 100.

9. Приведите пример суммирования элементов массива натуральных чисел

от 1 до 100.

10. Назначение и особенности использования адресации относительно счетчика команд.

11. В чем особенность реализации условных переходов в RISC-V ?

12. В чем отличие безусловных переходов при 12-бтных и 32-битных смещениях ?

13. Как и для чего используется команда auipc в ассемблере RISC-V?

14. Какие из видов адресации, используемых в iX86 не применяются в RISC-V?