**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Организация ЭВМ и систем»**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Мохамед М.Х. |
| Преподаватель |  | Морозов С.М. |

Санкт-Петербург

2024

**Цель работы.**

Целями работы являются:

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. 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.

Константы:  
a = 22 (№ студ. билета 334228)

b = 7 (Мохамед)

c = 6 (Махмуд)

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

Вариант 24

ЕСЛИ (arr[5] + arr[7] + arr[2] < threshold)

ТО (res1 = arr[7] | arr[5])

ИНАЧЕ (res2 = arr[9] & c)

tahreshold -> t3

res1 -> a5

res2 -> s7

Здесь threshold – заданный порог.

**Основные теоретические положения.**

*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

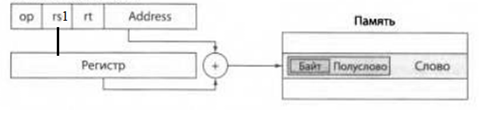
При использовании многоразрядных непосредственных операндов, если указанный в addi 12-битный непосредственный операнд отрицательный, старшая часть постоянного значения в lui должна быть увеличена на единицу. Помните, что знак addi расширяет 12-битное непосредственное значение, поэтому отрицательное непосредственное 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

**Выполнение работы.**

1. Формирование массива:

Сначала инициализируется значение a[0] = a + b + c, которое сохраняется в регистре s3, оттуда оно загружается в память по адресу 0, хранящемуся в x1, с помощью инструкции sw. В регистр s4, который используется как счетчик оставшихся элементов для записи в массив, записывается значение 10. В регистр s3 заносится значение, которое добавляется для получения последующего элемента массива. После этого в цикле, который продолжается до тех пор, пока значение в s4 не равно 0, из памяти по адресу, хранящемуся в x1, с помощью команды lw в s6 записывается значение последнего инициализированного элемента массива. К нему прибавляется необходимое значение, к значению в x1 прибавляется 4 для последующего доступа к следующей ячейке памяти, полученный результат из регистра s6 записывается в ячейку памяти по новому адресу в x1, счётчик количества элементов уменьшается на один.

2. Вычисление выражения с использованием 4-х режимов адресации:

Базовая адресация используется для загрузки элементов массива в регистры с помощью инструкции lw (чтение памяти). Например, загружаются значения из массива в регистры s7, s8, s9 для последующих вычислений.

Регистровая адресация применяется для выполнения арифметических операций с загруженными значениями. Например, значения, загруженные в регистры s7, s8, s9 складываются для получения суммы элементов массива.

Непосредственная адресация используется для загрузки значений, таких как номера системных вызовов, в регистр a7 для вывода символов и значений на консоль.

Адресация относительно счетчика команд применяется при выполнении условного перехода blt (меньше чем) и безусловного jal, чтобы перейти к завершению программы.

Тестирование программы для различных значений заданного порога представлены в таблице 1. Исходный код в приложении А.

Таблица 1 – результаты тестирования

|  |  |
| --- | --- |
| Входные данные | Результаты |
| threshold = 490 | Condition:  if arr[5] + arr[7] + arr[2] < 490 |
|  | then: result = arr[7] | arr[5] |
|  | else: result = arr[9] & 6  arr[5] + arr[7] + arr[2] = 427 |
|  | Array: 35 58 81 104 127 150 173 196 219 242 |
|  | result = 214 |
| threshold = 427 | Condition:  if arr[5] + arr[7] + arr[2] < 427 |
|  | then: result = arr[7] | arr[5] |
|  | else: result = arr[9] & 6  arr[5] + arr[7] + arr[2] = 427 |
|  | Array: 35 58 81 104 127 150 173 196 219 242 |
|  | result = 2 |
| threshold = 120 | Condition:  if arr[5] + arr[7] + arr[2] < 120 |
|  | then: result = arr[7] | arr[5] |
|  | else: result = arr[9] & 6  arr[5] + arr[7] + arr[2] = 427 |
|  | Array: 35 58 81 104 127 150 173 196 219 242 |
|  | result = 2 |
| threshold = -45 | Condition:  if arr[5] + arr[7] + arr[2] < -45  then: result = arr[7] | arr[5]  else: result = arr[9] & 6  arr[5] + arr[7] + arr[2] = 427  Array: 35 58 81 104 127 150 173 196 219 242  result = 2 |
|  |
|  |
|  |
|  |
|  |

**Выводы.**

В ходе выполнения лабораторной работы получены и закреплены знания по режимам адресации в ассемблере процессора RISC-V. Разработана программа преобразования данных, работающая с памятью и использующая все типы режимов адресации.

**ПРИЛОЖЕНИЕ А.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ.**

Название файла lb6.s:

.text # раздел с инструкциями программы

.global \_start

\_start:

.equ a, 22 # задаются константы

.equ b, 7

.equ c, 6

.equ threshold, 428

addi s0, x0, a # константы a, b, c и threshold размещаются

addi s1, x0, b # в регистры s0, s1

addi s2, x0, c # s2 и t3 соответственно

addi t3, x0, threshold

la a0, condition # вывод строки по адресу в а0

addi a7, x0, 4 # с помощью системного вызова PrintString

ecall # (a7=4)

la a0, sum\_cond # вывод строки по адресу в а0

addi a7, x0, 4

ecall

li a0, 60 # вывод символа <

addi a7, x0, 11

ecall

li a0, 32 # вывод пробела

addi a7, x0, 11

ecall

mv a0, t3 # вывод числа из t3(threshold)

addi a7, x0, 1

ecall

la a0, then # вывод строки по адресу в а0

addi a7, x0, 4

ecall

la a0, result # вывод строки по адресу в а0

addi a7, x0, 4

ecall

la a0, res1 # вывод строки по адресу в а0

addi a7, x0, 4

ecall

la a0, else # вывод строки по адресу в а0

addi a7, x0, 4

ecall

la a0, result # вывод строки по адресу в а0

addi a7, x0, 4

ecall

la a0, res2 # вывод строки по адресу в а0

addi a7, x0, 4

ecall

add s3, s0, s1 # теперь s3 = s0 + s1 = a + b

add s3, s3, s2 # теперь s3 = s3 + s2 = a + b + c

addi s4, x0, 10 # s4 = 9, это счётчик для элементов массива

sw s3, 0(x1) # значение из s3 сохраняется в память по адресу из x1

la a0, array # вывод строки по адресу в а0

addi a7, x0, 4

ecall

sub s3, s3, s2 #s3 = a + b + c - c = a + b

sub s3, s3, s2 #s3 = a + b - c

loop: # цикл для инициализации массива в памяти

lw s6, 0(x1) # в s6 сохраняется значение из памяти по адресу из x1, т.е. a[i]

mv a0, s6

addi a7, x0, 1

ecall

li a0, 32

addi a7, x0, 11

ecall

addi x1, x1, 4 # x1 = x1 + 4

add s6, s6, s3 # s6 = s6 + s3 = a[i] + a + b -c

sw s6, 0(x1) # значение из s6 сохраняется в память по адресу из x1

addi s4, s4, -1 # s4 = s4 -1, уменьшение счётчика

bnez s4, loop # повторение цикла, пока s4 != 0

add x1, x0, x0

lw s7, 20(x1) # в регистры s7, s8, s9 сохраняются

lw s8, 28(x1) # arr[5], arr[7], arr[2] соответственно

lw s9, 8(x1)

add s8, s7, s8 # s8 = arr[5] + arr[7]

add s8, s8, s9 # s8 = arr[5] + arr[7] + arr[2]

li a0, 10 # вывод переноса строки

addi a7, x0, 11

ecall

la a0, sum\_cond # вывод строки по адресу в а0

addi a7, x0, 4

ecall

li a0, 61 # вывод символа =

addi a7, x0, 11

ecall

li a0, 32 # вывод пробела

addi a7, x0, 11

ecall

mv a0, s8 # вывод значения из s8

addi a7, x0, 1

ecall

li a0, 10 # вывод пробела

addi a7, x0, 11

ecall

la a0, result # вывод строки по адресу в а0

addi a7, x0, 4

ecall

blt s8, t3, condition\_1 # если значение в s8 < значения в t3(threshold), то осуществляется переход на метку first

lw s11, 36(x1) # в s11 загружается arr[9]

and s7, s11, s2 # s7 = arr[9] & c

mv a0, s7 # вывод значения из s7

addi a7, x0, 1

ecall

jal x0, done # безусловный переход на метку done

condition\_1:

lw s10, 28(x1) # в s10 загружается arr[7]

or a5, s10, s7 # s7 = arr[7] | arr[5]

mv a0, a5 # вывод значения из s7

addi a7, x0, 1

ecall

done:

addi a0, x0, 0 # завершение программы системным вызовом

addi a7, x0, 93 # Exit (a7=93) с кодом возврата в регистре а0

ecall

.data # раздел с данными для переменных программы

condition: .asciz "Condition:\nif "

sum\_cond: .asciz "arr[5] + arr[7] + arr[2] "

res1: .asciz "arr[7] | arr[5]"

res2: .asciz "arr[9] & 6"

then: .asciz "\nthen: "

else: .asciz "\nelse: "

result: .asciz "result = "

array: .asciz "\nArray: "