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.

Студентка гр. 2382 _____ Ульянова Е.А.

Преподаватель _____ Морозов С.М.

Санкт-Петербург

2023

Цель работы.

Изучение режимов адресации в ассемблере 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]

a = 17 (№ студ. билета 238220)

b = 8 (Ульянова)

c = 9 (Екатерина)

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:

Вариант 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

Здесь threshold – заданный порог.

Основные теоретические положения.

Режимы адресации в ассемблере RISC-V:

1. 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.

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

2. Непосредственная адресация:

При непосредственной адресации в качестве операндов наряду с регистрами используют константы (непосредственные операнды).

Пример: addi rd,rs1,12 # rd = rs1 + 12

Чтобы использовать константы большего размера, следует использовать инструкцию непосредственной записи в старшие разряды lui (load upper immediate), за которой следует инструкция непосредственного сложения addi.

Пример: lui s2, 0xABCDE # s2 = 0xABCDE000
addi s2, s2, 0x123 # s2 = 0xABCDE123

3. 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.

Пример: lw rd, 36(rs1)

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

4. Адресация относительно счётчика команд:

Инструкции условного перехода (beq, bne, blt, bge, bltu, bgeu), или ветвления, а также jal (переход и связывание) 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.

Пример: beq rs1,rs2,imm # if(rs1 == rs2) 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:

Первым инициализируется значение a[0] = a + b + c, которое хранится в регистре s3, оттуда оно загружается в память по адресу 0, хранящемуся в x1, инструкцией sw. В регистр s4 записывается значение 9 – столько элементов осталось записать в массив. В регистр s3 заносится значение, которое нужно добавлять 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:

Используя базовую адресацию, инструкцией lw (чтение памяти) необходимые элементы массива загружаются в регистры для последующего выполнения действий с ними.

Регистровая адресация используется для того, чтобы производить необходимые арифметические и логические операции со значениями массива, загруженными в регистры, например, сложение и побитовое И.

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), а также безусловного jal, когда нужно перейти к завершению программы после того, как выполнены инструкции, необходимые в том случае, если заданное условие оказалось невыполненным.

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

Таблица 1 – результаты тестирования

Входные данные	Результаты
threshold = 200	Condition: if a[4] + a[1] + a[0] >= 200 then: result = a[5] + a[0] else: result = a[2] & 17 Array: 34 50 66 82 98 114 130 146 162 a[4] + a[1] + a[0] = 182 result = 0
threshold = 182	Condition: if a[4] + a[1] + a[0] >= 182 then: result = a[5] + a[0] else: result = a[2] & 17 Array: 34 50 66 82 98 114 130 146 162 a[4] + a[1] + a[0] = 182 result = 148
threshold = 20	Condition: if a[4] + a[1] + a[0] >= 20 then: result = a[5] + a[0] else: result = a[2] & 17 Array: 34 50 66 82 98 114 130 146 162 a[4] + a[1] + a[0] = 182 result = 148
threshold = -10	Condition: if a[4] + a[1] + a[0] >= -10 then: result = a[5] + a[0] else: result = a[2] & 17 Array: 34 50 66 82 98 114 130 146 162 a[4] + a[1] + a[0] = 182 result = 148

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, 17 # задаются константы
      .equ b, 8
      .equ c, 9
      .equ threshold, 182
      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 s3 соответственно
      addi t4, x0, threshold
      la a0, condition # вывод строки по адресу в a0
      addi a7, x0, 4 # с помощью системного вызова PrintString
      ecall # (a7=4)
      la a0, sum_cond # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      li a0, 62 # вывод символа >
      addi a7, x0, 11
      ecall
      li a0, 61 # вывод символа =
      addi a7, x0, 11
      ecall
      li a0, 32 # вывод пробела
      addi a7, x0, 11
      ecall
      mv a0, t4 # вывод числа из t4
      addi a7, x0, 1
      ecall
      la a0, then # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      la a0, result # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      la a0, res1 # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      la a0, else # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      la a0, result # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      la a0, res2 # вывод строки по адресу в a0
      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, 9 # s4 = 9, это счётчик для элементов массива
      sw s3, 0x0(x1) # значение из s3 сохраняется в память по адресу из
x1
```

```
      la a0, array # вывод строки по адресу в a0
      addi a7, x0, 4
      ecall
      addi s5, x0, -2 # s5 = -2
      mul s5, s5, s2 # s5 = -2*c
      add s3, s3, s5 # s3 = a + b - c
loop: # цикл для инициализации массива в памяти
      lw s6, 0x0(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, 0x0(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, 0x0(x1) # в регистры s7, s8, s9 сохраняются
      lw s8, 0x4(x1) # a[0], a[1], a[4] соответственно
      lw s9, 0x10(x1)
      add s8, s7, s8 # s8 = a[0] + a[1]
      add s8, s8, s9 # s8 = a[0] + a[1] + a[4]
      li a0, 10 # вывод переноса строки
      addi a7, x0, 11
      ecall
      la a0, sum_cond # вывод строки по адресу в a0
      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 a0
      addi a7, x0, 4
      ecall
      bge s8, t4, first # если значение в s8 >= значения в t4, то
осуществляется переход на метку first
      lw s11, 0x8(x1) # в s11 загружается a[2]
      and a6, s11, s0 # a6 = s11 & s0 = a[2] & a
      mv a0, a6 # вывод значения из a6
      addi a7, x0, 1
      ecall
      jal x0, done # безусловный переход на метку done
first:
      lw s10, 0x14(x1) # в s10 загружается a[5]
```

```
      add s7, s10, s7 # s7 = a[5] + a[0]
      mv a0, s7 # вывод значения из s7
      addi a7, x0, 1
      ecall
done:
      addi a0, x0, 1 # завершение программы системным вызовом
      addi a7, x0, 93 # Exit (a7=93) с кодом возврата в регистре a0
      ecall


.data # раздел с данными для переменных программы
      condition: .asciz "Condition:\nif "
      sum_cond: .asciz "a[4] + a[1] + a[0] "
      res1: .asciz "a[5] + a[0]"
      res2: .asciz "a[2] & 17"
      then: .asciz "\nthen: "
      else: .asciz "\nelse: "
      result: .asciz "result = "
      array: .asciz "\nArray: "
```