**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

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

Тема: Изучение режимов адресации в ассемблере RISC-V.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Еникеев А.А. |
| Преподаватель |  | Молодцев Д.А |

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

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). Порядок операций в формуле соответствует порядку операций в языке Си.

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. Размер каждого элемента массива - 8 байт (dword).

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[6] - arr[7] ^ arr[9]) <= 318)

ТО (arr[i] = arr[i - 1] & 38)

ИНАЧЕ (arr[i] = arr[i] ^ 36)

## Основные теоретические положения

1. **Регистры RISC-V**

RISC-V имеет 32 регистра общего назначения (x0-x31), каждый из которых имеет ширину 32 бита (в RV32I). Регистр x0 всегда содержит значение 0 и не может быть изменен. Остальные регистры используются для хранения данных, адресов и результатов вычислений.

Некоторые регистры имеют псевдонимы:

* x1 (ra) — регистр возврата из функции.
* x2 (sp) — указатель стека.
* x10-x17 (a0-a7) — регистры аргументов функций и возврата значений.
* x5-x7, x28-x31 (t0-t6) — временные регистры.

1. **Базовый набор команд**

Инструкции RISC-V делятся на несколько типов:

* R (Register) — операции над регистрами.

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

sub rd, rs1, rs2 # rd = rs1 - rs2

xor rd, rs1, rs2 # rd = rs1 ^ rs2

* I (Immediate) — операции с непосредственными значениями.

addi rd, rs1, imm # rd = rs1 + imm

lw rd, imm(rs1) # Загрузка слова из памяти

* S (Store) — команды сохранения данных в память.

sw rs2, imm(rs1) # Сохранение rs2 по адресу rs1 + imm

* B (Branch) — условные переходы (ветвления).

beq rs1, rs2, label # Переход, если rs1 == rs2

* U (Upper immediate) — работа с большими константами.

lui rd, imm # Загрузка 20-битного значения в старшие биты

* J (Jump) — безусловные переходы.

jal rd, offset # Переход на offset, сохранение PC+4 в rd

1. **Режимы адресации в RISC-V**

RISC-V поддерживает несколько основных режимов адресации:

* *Регистровая адресация*: в этом режиме операнды и результат хранятся в регистрах. Используется в инструкциях типа R.

add t0, t1, t2 # t0 = t1 + t2

* *Непосредственная адресация*: в инструкциях типа I константы используются как операнды.

addi t0, t1, 10 # t0 = t1 + 10

* *Базовая адресация*: используется для доступа к памяти. Адрес формируется как сумма базового адреса в регистре и смещения.

lw t0, 4(sp) # Загрузить слово по адресу sp + 4

sw t0, 8(sp) # Сохранить t0 по адресу sp + 8

* *Адресация относительно счетчика команд* (PC-relative): используется для условных и безусловных переходов. Смещение прибавляется к текущему значению счетчика команд (PC).

beq t0, t1, label # Переход к label, если t0 == t1

jal ra, label # Переход на label с сохранением PC+4 в ra

## Выполнение работы

1. Процедура main

* Загружает исходный массив и его размер.
* Печатает строку "Входные данные" и выводит элементы массива на экран.
* Вызывает функцию solution для обработки массива.
* Печатает строку "Результат" и выводит измененный массив.
* Завершает выполнение программы с помощью системного вызова ecall.

1. Процедура print\_arr для вывода массива в консоль

* Инициализирует счетчик t0 для перебора элементов массива.
* В цикле проверяет, достигнут ли конец массива (сравнивает t0 с размером массива a2).
* Вычисляет адрес текущего элемента массива, загружает его значение и выводит на экран с помощью процедуры print\_int.
* После каждого элемента печатает разделитель ", ".
* В конце печатает символ новой строки "\n".

1. Процедура solution для обработки массива в соответствии с заданным условием

* Инициализирует вспомогательные регистры:
  + - t0 — счетчик текущего индекса i.
    - t1 — переменная prev для хранения предыдущего значения.
    - t5 — смещение для вычисления адреса текущего элемента.
    - s5 — константа 318 для проверки условия.
* В цикле перебирает элементы массива:
  + - Загружает текущий элемент массива arr[i] и значения arr[6], arr[7], arr[9].
    - Выполняет промежуточные вычисления:
      * + s2 = arr[6] - arr[7]
        + s3 = (arr[6] - arr[7]) ^ arr[9]
    - Проверяет условие: если s3 <= 318, выполняется блок THEN, иначе — блок ELSE:
      * + THEN: arr[i] = prev & 38
        + ELSE: arr[i] = arr[i] ^ 36
    - Сохраняет результат обратно в массив и обновляет переменную prev.
    - Переходит к следующему элементу массива.
* Завершает выполнение после обработки всех элементов.

1. print\_string — процедура для вывода строки на экран с использованием системного вызова ecall.
2. print\_int — процедура для вывода целого числа на экран с использованием системного вызова ecall.

Исходный код программы см. в прил. A.

**Тестирование**

Результаты тестирования см. в табл. 1.

Табл. 1.

|  |  |
| --- | --- |
| Входные данные | Результат |
| 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000 | 0, 0, 0, 0, 0, 0, 0, 0, 8972, 10036 |
| 100, 200, 300, 400, 500, 600, 1000, 350, 800, 450 | 64, 236, 264, 436, 464, 636, 972, 378, 772, 486 |
| -100, -200, -300, -400, -500, -600, -350, -700, -800, -450 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 |
| 2147483647, 2147483646, 2147483645, 2147483644, 2147483643, 2147483642, 2147483641, 2147483640, 2147483639, 2147483638 | 2147483611, 2147483610, 2147483609, 2147483608, 2147483615, 2147483614, 2147483613, 4, 4, 4 |

**Выводы**

В ходе выполнения лабораторной работы были закреплены навыки программирования на ассемблере RISC-V и изучены режимы адресации, применяемые для работы с данными в памяти. Реализованная программа продемонстрировала использование различных подходов к обработке массивов, а также работу с условиями и логическими операциями. Работа позволила углубить понимание архитектуры RISC-V и принципов её функционирования.

# Приложение A

**Исходный код программы**

**Файл LR6.s:**

.text

main:

la a1, arr # a1 = адрес массива arr

lw a2, size # a2 = размер массива (количество элементов)

la a0, input1 # a0 = адрес строки "Входные данные: "

call print\_string # Вызов функции для печати строки

call print\_arr # Вызов функции для печати массива

la a0, result1 # a0 = адрес строки "Результат: "

call print\_string # Вызов функции для печати строки

call solution # Вызов функции solution для обработки массива

call print\_arr # Вызов функции для печати измененного массива

# Завершаем программу с помощью ecall

li a7, 10 # Код ecall для завершения программы

ecall

# Функция для печати элементов массива

print\_arr:

li t0, 0 # t0 = счетчик элементов массива (i = 0)

mv s0, ra # Сохраняем адрес возврата (ra) в s0

print\_elem:

# Проверка: если i >= size, завершаем печать

bge t0, a2, print\_endline

# Вычисляем адрес текущего элемента: t1 = a1 + (i \* 4)

slli t1, t0, 2 # t1 = i \* 4 (смещение в байтах)

add t1, t1, a1 # t1 = адрес arr[i]

lw t2, 0(t1) # Загружаем arr[i] в t2

# Печатаем элемент массива

mv a0, t2 # a0 = arr[i]

call print\_int # Вызов функции для печати целого числа

# Печатаем разделитель ", "

la a0, sep # a0 = адрес строки ", "

call print\_string # Вызов функции для печати строки

# Переходим к следующему элементу

addi t0, t0, 1 # i++

j print\_elem # Переход к следующей итерации

print\_endline:

# Печатаем символ новой строки

la a0, nl # a0 = адрес строки "\n"

call print\_string # Вызов функции для печати строки

# Восстанавливаем адрес возврата и завершаем функцию

mv ra, s0 # Восстанавливаем ra

ret

# Функция solution: модификация массива по заданному условию

solution:

# Инициализация переменных

li t0, 0 # t0 = счетчик i = 0

li t1, 0 # t1 = prev = 0 (предыдущее значение)

li t5, 0 # t5 = смещение для доступа к элементам массива

li s5, 318 # s5 = константа для проверки условия

start:

# Вычисляем адрес текущего элемента arr[i]

add t6, a1, t5 # t6 = адрес arr[i]

lw a3, 0(t6) # a3 = arr[i] (текущее значение)

# Загружаем значения arr[6], arr[7] и arr[9]

lw s2, 24(a1) # s2 = arr[6]

lw s3, 28(a1) # s3 = arr[7]

lw s4, 36(a1) # s4 = arr[9]

# Выполняем промежуточные вычисления

sub s2, s2, s3 # s2 = arr[6] - arr[7]

xor s3, s2, s4 # s3 = (arr[6] - arr[7]) ^ arr[9]

# Проверяем условие: if ((arr[6] - arr[7]) ^ arr[9]) <= 318

ble s3, s5, branch\_then # Если условие выполняется, переход к THEN

branch\_else:

# ИНАЧЕ: arr[i] = arr[i] ^ 36

li t2, 36 # t2 = константа 36

xor a3, a3, t2 # a3 = arr[i] ^ 36

j save\_next # Переход к сохранению результата

branch\_then:

# ТО: arr[i] = arr[i - 1] & 38

li t2, 38 # t2 = константа 38

and a3, t1, t2 # a3 = prev & 38 (предыдущее значение & 38)

save\_next:

# Сохраняем результат обратно в массив

sw a3, 0(t6) # arr[i] = a3

mv t1, a3 # prev = a3 (обновляем предыдущее значение)

# Переходим к следующему элементу

addi t0, t0, 1 # i++

bge t0, a2, end # Если i >= size, завершаем

addi t5, t5, 4 # Смещение += 4 (следующий элемент)

j start # Переход к следующей итерации

end:

ret # Возвращаемся из функции

# Функция для печати строки

print\_string:

li a7, 4 # Код ecall для печати строки

ecall # Вызов системного вызова

ret

# Функция для печати целого числа

print\_int:

li a7, 1 # Код ecall для печати целого числа

ecall # Вызов системного вызова

ret

.data:

arr: .word 2147483647, 2147483646, 2147483645, 2147483644, 2147483643, 2147483642, 2147483641, 2147483640, 2147483639, 2147483638

size: .word 10

# Строки для вывода

input1: .string "Входные данные: "

result1: .string "Результат: "

sep: .string ", "

nl: .string "\n"