Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 3**

Дисциплина: Низкоуровневое программирование

Тема: Программирование RISC-V

Вариант: 4

Выполнил студент гр. 3530901/90002 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.К. Борисов

(подпись)

Принял старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д.С. Степанов

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Цель работы:**

Постановка задачи

1. Разработать программу на языке ассемблера RISC-V, реализующую

определенную вариантом задания функциональность, отладить программу в

симуляторе VSim/Jupiter. Массив (массивы) данных и другие параметры

(преобразуемое число, длина массива, параметр статистики и пр.)

располагаются в памяти по фиксированным адресам.

2. Выделить определенную вариантом задания функциональность в

подпрограмму, организованную в соответствии с ABI, разработать

использующую ее тестовую программу. Адрес обрабатываемого массива

данных и другие значения передавать через параметры подпрограммы в

соответствии с ABI. Тестовая программа должна состоять из

инициализирующего кода, кода завершения, подпрограммы main и

тестируемой подпрограммы

**Вариант 4:**

Реализовать алгоритм сортировки вставкой.

**Задание:**

Необходимо смоделировать программу для EDSAC, которая реализует алгоритм сортировки массива вставкой. Отсортированный массив должен находится в тех же ячейках, где находился изначально.

**Решение:**

Для написания программы для RISC-V, был изначально написан алгоритм на языке python, комментарии в программе опираются на написанный на python-e алгоритм (рис.1).

Алгоритм сортировки вставками - алгоритм, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

|  |
| --- |
|  |
| Рис. 1 Алгоритм сортировки вставкой (python) |

**Алгоритм:**

1. **Получаем значения arr[i], (item\_to\_insert) , arr[j], устанавливаем j = i.**
2. **Проверяем условие item\_to\_insert > arr[j] , если больше увеличиваем значение i и переходим к следующей итерации. Иначе идем дальше.**
3. **Получаем значение элемента arr[j].**
4. **Проверяем j = 0, если true, то переходим к пункту 7, иначе идем дальше.**
5. **Проверяем item\_to\_insert > arr[j], если true, то переходим к пункту 7, иначе заходим в цикл while.**
6. **Устанавливаем в arr[j + 1] значение элемента arr[j], уменьшаем j на единицу и прыгаем обратно в пункт 3.**
7. **Устанавливаем в arr[j + 1] значение элемента для вставки, увеличиваем i на единицу.**
8. **Проверяем i >= arr.lenght, если true, завершаем работу программы, иначе переходим к следующей итерации.**

**Решение.**

Для написания кода программы был выбран редактор кода atom и добавлен синтаксис RISC-V с помощью библиотеки “ language-riscv”. Ниже приведен код написанной программы.

|  |
| --- |
|  |
| Рис. 2 Код первой программы. |

***“text”*** – указатель на основную часть кода, *“start”* – точка начала выполнения программы. В строках 4-7 написаны псевдоинструкции установки значений

регистров.

*a2* = i.

*a3* = длинна массива.

*a4* = адрес первого элемента массива.

Разберем далее программу построчно:

**8-35** - Основная часть программы.

**8** - Вход в цикл for. Инструкция в строке служит условием выхода из цикла for (i >= длинна массива), если true, переходим к 33 строчке. Иначе идем далее по программе.

**9-11 -** Инструкции, отвечающие за вычисление адреса arr[i] при помощи сдвига влево i-ого числа и прибавления этого выражения к нулевому элементу массива. Значение заносится в регистр **a5.**

**12-14 -** В регистры t1, t0, t2 записывается arr[i - 1], arr[i] и j = i соответственно. Далее в строке 15 проверяется условие (item\_to\_insert > arr[j]), если true – мы переходим к строке 29, где увеличиваем значение i и переходим к следующей итерации. Иначе заходим в цикл While.

**17-19** - получаем значение arr[j] и записываем его в регистр t6

**20-21 –** Условия цикла while: j = 0, item\_to\_insert > arr[j], если хоть одно из условий true – выходим из while в строку 27.

**22-25 –** Записываем в arr[j + 1] значение элемента arr[j], уменьшает j на единицу и записываем в регистр t5 ссылку на элемент arr[j + 1] для последующей вставки числа в это значение после выхода из цикла. Далее прыгаем в строку 16.

**26-30 –** Вставляем элемент для вставки (item\_to\_insert) в ячейку arr[j + 1], увеличиваем i на единицу и переходим к следующей итерации обратно в строку 8.

**33-35 –** Завершение работы программы

**36-41 –** Данные программы. *.word* - означает, что мы используем 32-битные слова (4 байта). То есть они занимают 4 восьмибитных секции. *.rodata* – константы, *.data* – переменные.

**Примеры работы программы**

C помощью команд locals, breakpoint и c посмотрим заданный массив в начале и конце работы программы.

|  |
| --- |
|  |
| Рис. 3 Значения элементов массива в начале работы программы. |

Arr = [10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4] – значение элементов в десятичном представлении

Перейдем с помощью команды breakpoint к метке finish, чтобы посмотреть значения массива в конце работы программы.

|  |
| --- |
|  |
| Рис. 3 Значения элементов массива в конце работы программы. |

Arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] - значение элементов в десятичном представлении.

Как мы видим, массив отсортировался корректно.

**Часть 2**

|  |
| --- |
|  |
| Рис.5 Тестирующая программа. |

В программе вызывается подпрограмма main с помощью команды call.

Псевдоинструкция call соответствует следующей паре инструкций:



Исполненные одна за другой, эти инструкции обеспечивают безусловный

переход (jump) на метку main с сохранением адреса следующей за jalr

инструкции в регистре ra (синоним x1).

Когда выполнение подпрограммы завершится, исполнение кода

тестирующей программы перейдет к метке finish, в которой работа программы

завершается.

|  |
| --- |
|  |
| Рис. 4 Подпрограмма main. |

Здесь задаются данные работы программы: адрес первого элемента массива и длинна массива в соответствующие регистры *a0*, *a1*. После этого

происходит вызов подпрограммы insertion. Однако ее вызов отличается от вызова

*main* в тестирующей подпрограмме. Это связано с тем, что мы уже находимся в подпрограмме, а регистр с кодом возврата (*ra*) один, поэтому перед тем, как мы вызовем еще одну подпрограмму, нам нужно где-то сохранить этот код.

*ret* - возврат из подпрограммы.

|  |
| --- |
|  |
| Рис. 5 Подпрограмма insertion |

Программа сортировки вставкой , оформленная в подпрограмму. В конце программы, вместо завершения – выход из подпрограммы инструкцией *ret.*

**Результаты работы программы:**

|  |
| --- |
|  |
| Рис. 6 Значения элементов массива в начале работы программы. |

Arr = [10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4] – значение элементов в десятичном представлении в начале работы программы.

|  |
| --- |
|  |
| Рис. 7 Значения элементов массива в конце работы программы. |

Arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] - значение элементов в десятичном представлении в конце работы.

Таким образом, мы видим, что программа работает корректно.

**Вывод**

В ходе выполнения лабораторной работы была реализована программа на RISCV, реализующая алгоритм сортировки вставкой. Были изучены основные аспекты RISC-V и изучена отладка программ на этом языке.