Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе**

**Дисциплина**: Низкоуровневое программирование

**Тема**: RISC-V

Выполнила студентка гр. 3530901/90004

Р.И. Галиева

(подпись)

Преподаватель А.О.Алексюк

(подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

1. **Формулировка задачи**
   1. Разработать программу на языке ассемблера RISC-V, реализующую определенную 6 вариантом 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.
2. **Описание работы.**
   1. **Используемый алгоритм**

Counter – счетчик

i – индексация во внешнем цикле

j – индексация во внутреннем цикле

**Псевдокод алгоритма:**

int[] a = new int[];

int counter ;

int res = 0;

for(int i =0; i < a.lenght; i++){

counter = 0;

for(int j=0;j<a.length;j++){

if(a[i] < a[i] )counter -= 1;

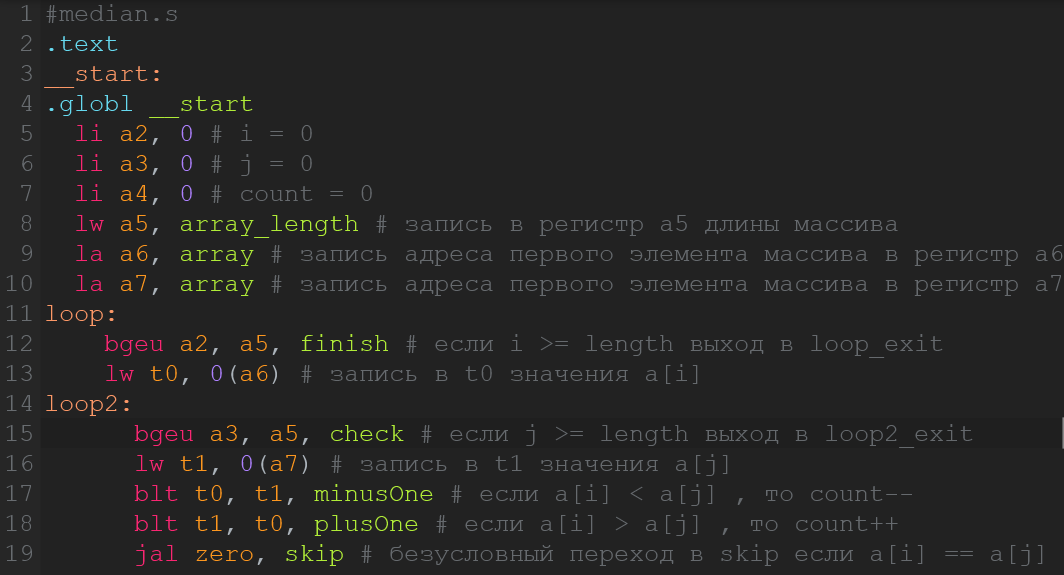
if(a[i] > a[i] ) counter += 1;

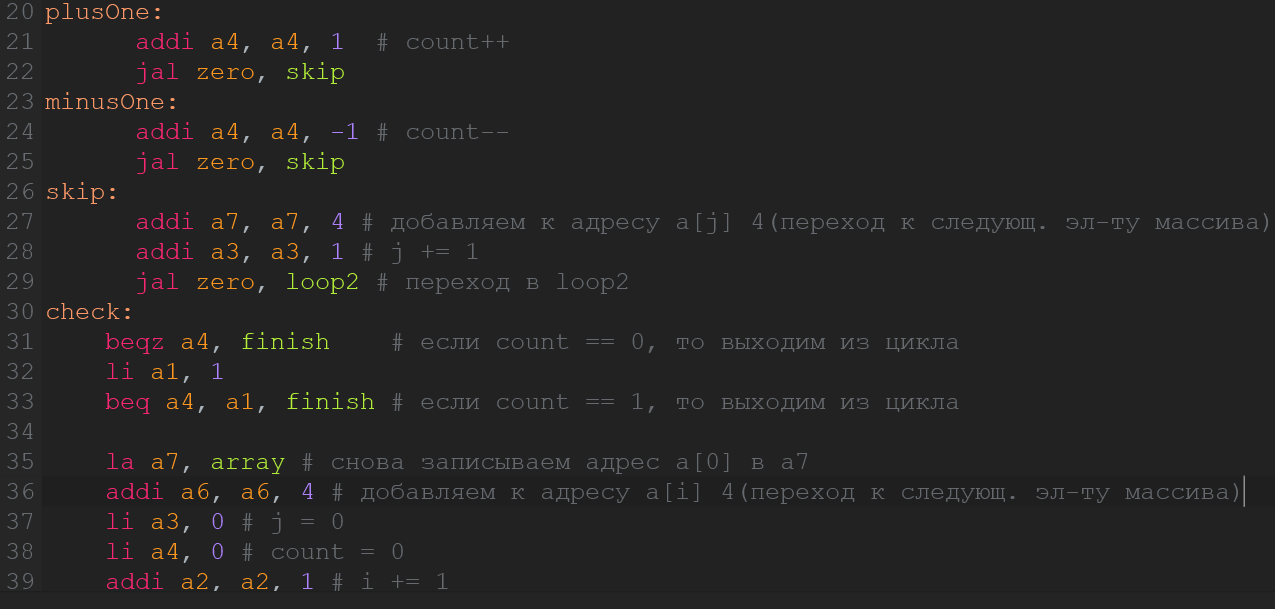
}

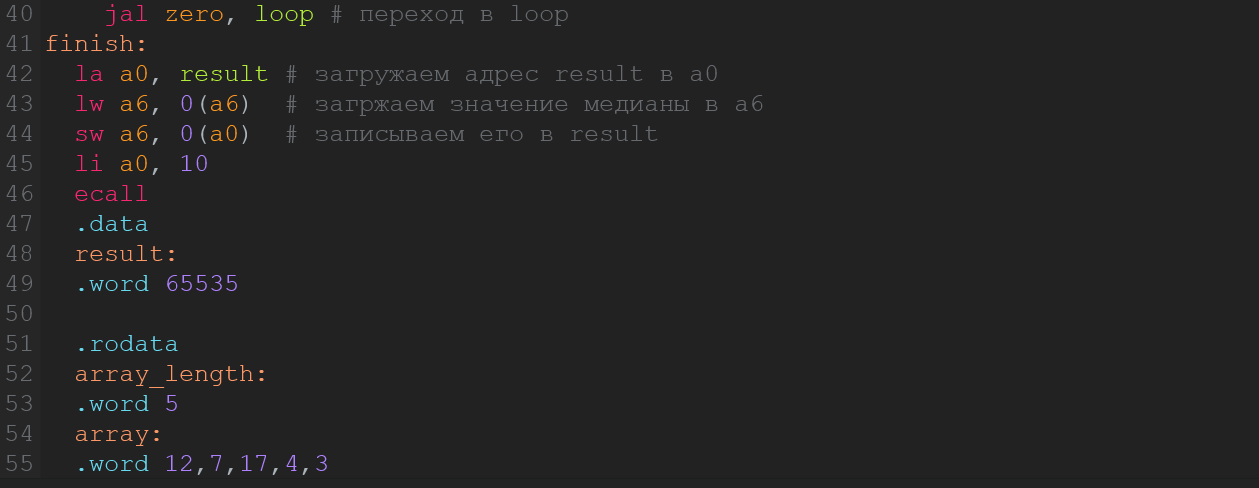
if(counter == 0 || counter == 1) res = a[i]

}

* 1. **Код программы**



****

****

**Результаты работы алгоритма:**

|  |  |
| --- | --- |
| Массив array = [ 12,7,17,4,3]. Length =5 | Массив array = [1,7,9,4]. Length = 4 |
|  |  |

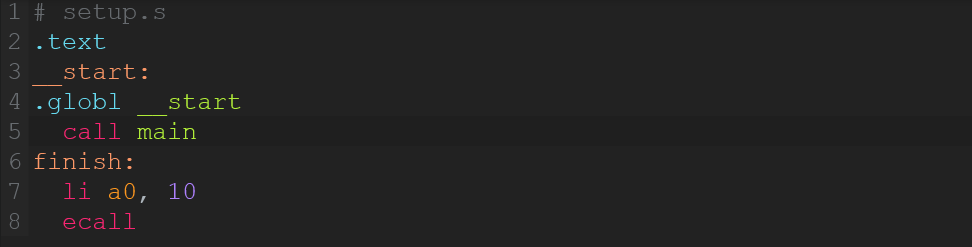
**Руководство программиста**

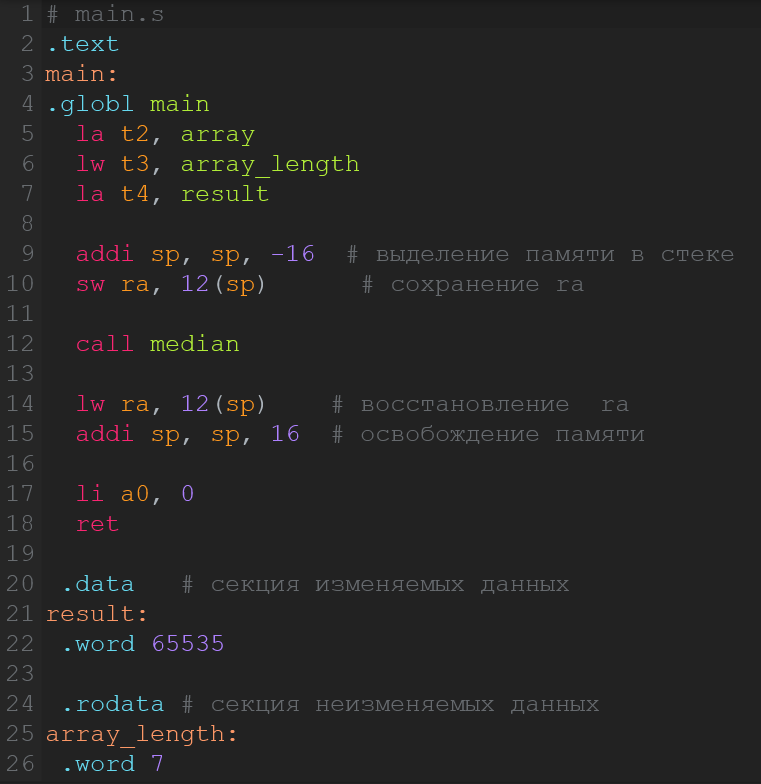
* **.text** - указание ассемблеру размещать последующие инструкции в секции кода.
* Метка **\_\_start:** - точка начала выполнения программы.
* В строках 5-12 написаны псевдоинструкции установки значений регистров **а2-а7**. В регистре **а2** хранится счетчик i, в **a3 – счетчик** j, в **a4 – счетчик** count. В **а5** хранится длина массива. В регистрах **а6-а7** хранится адрес нулевого элемента массива.
* В строках 14-49 записана основная часть программы.
* Под меткой **loop** происходит проверка, прошлись ли мы по всему внешнему циклу, далее записывается значение текущего элемента массива a[i] в регистр **t0.**
* Под меткой **loop2** выполняется та же проверка, но для внутреннего цикла. Записывается другой текущий элемент массива a[j] в регистр **t1** и сравниваются два элемента массива (**t0 и t1). Если t0 < t1, то происходит переход в minusOne**, если t1<t0(t0>t1), то в **plusOne**, если они равны, то в **skip**.
* В **plusOne** счетчик count увеличивается на 1, в **minusOne** – уменьшается на 1, а после из обеих меток программа переходит в **skip**, где к адресу массива **a7** добавляется 4, для перехода к следующему элементу. К **а3** (j) прибавляется 1, и программа возвращается в начало внутреннего цикла.
* Когда будет пройден весь массив(i = array.length), программа перейдет к метке **check**, в которой проверит «была ли найдена медиана» (**count** == 0 или count ==1), если проверка успешна, то программа перейдет к метке **finish**, иначе запишет в **а7** адрес нулевого элемента массива, обнулит счетчики **count** и **j**, добавит к счетчику **i** 1 и вернется в начало внешнего цикла.
* В метке **finish** программа загрузит медиану в **result** и закончит свою работу.
  1. **Выделение подпрограммы**

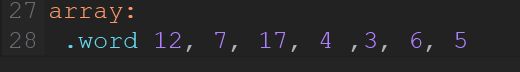
В результате «исполнения» псевдоинструкции call управление будет передано на метку main, а в регистр ra будет записан адрес следующей за call инструкции.

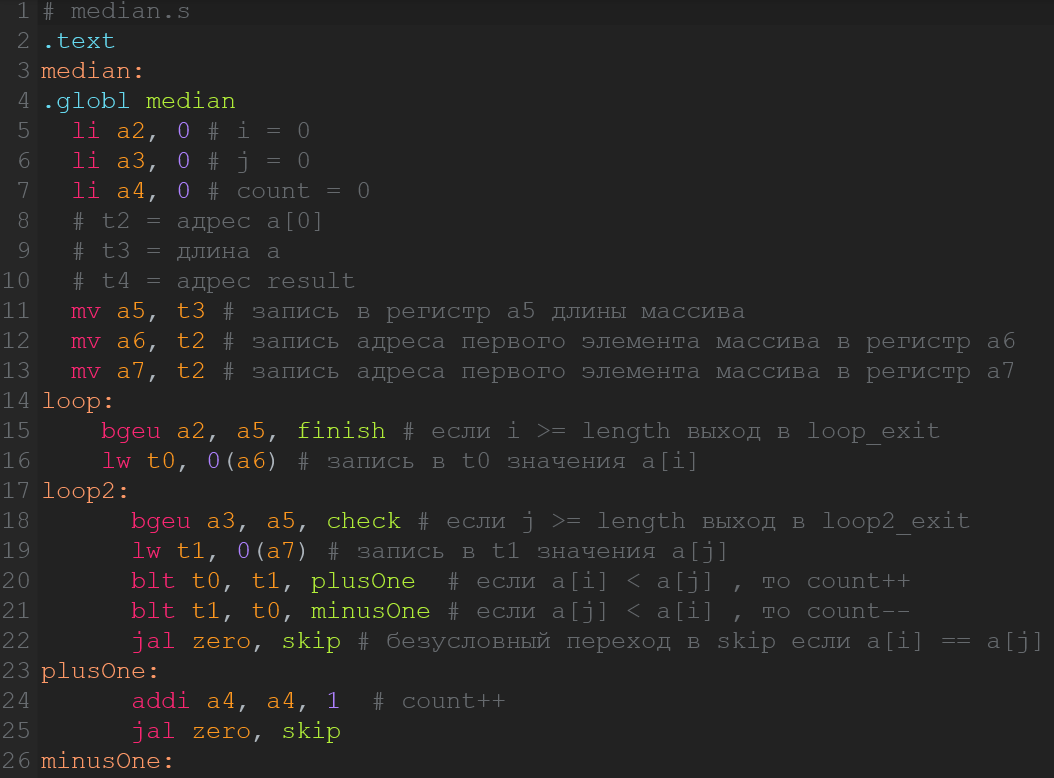
Исходное значение ra следует сохранить перед псевдоинструкцией call, и восстановить перед псевдоинструкцией ret. Значение регистра можно сохранить либо в другом регистре, либо в памяти. Как известно, регистры делятся на две группы: рабочие (temporary, caller-saved) и сохраняемые (callee-saved). Значение ra нельзя сохранить в рабочем регистре, так как значение этого регистра может быть изменено вызываемой подпрограммой, а чтобы сохранить значение ra в сохраняемом регистре, значение самого этого регистра необходимо сохранить и восстановить перед возвратом из main . Таким образом, значение ra следует сохранить в памяти, и естественно использовать для этого стек

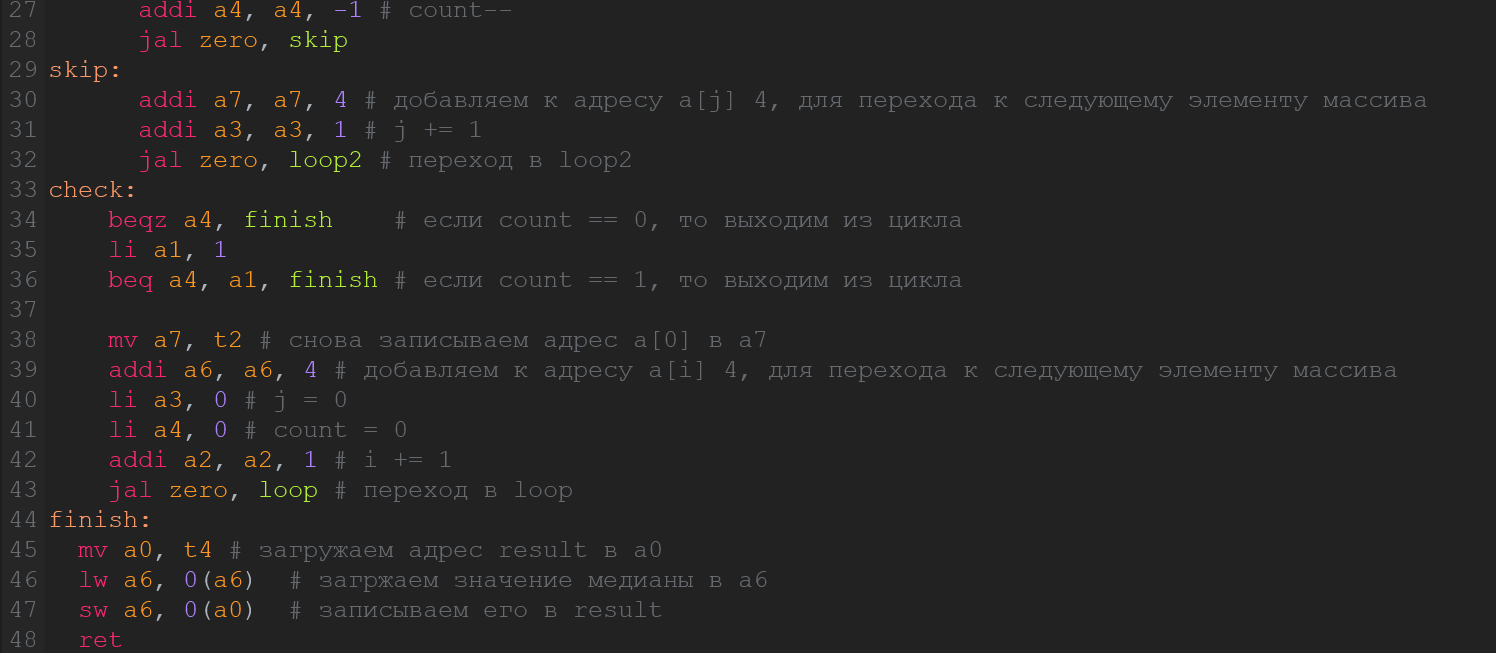
В случае 32-разрядной версии RISC-V для сохранения значения ra в стеке требуется только 4 байта, однако ABI RISC-V требует выравнивания указателя стека на границу 128 разрядов (16 байт), следовательно, величина изменения указателя стека должна быть кратна 16. Кроме того, в RISC-V (как и в большинстве архитектур) *стек растет вниз* (grows downwards), то *есть выделению памяти в стеке (stack allocation) соответствует уменьшение значения указателя стека*. Отметим, что начальное значение sp устанавливается симулятором. В ABI RISC-V регистр sp является сохраняемым, то есть при возврате из подпрограммы он должен иметь исходное значение. Поскольку для выделения памяти в стеке значение sp уменьшается (в данном случае на 16), перед возвратом из подпрограммы достаточно увеличить sp на ту же величину.











Здесь задаются исходный массив (27-28 строки), его длина (24-25 строки) и результат (21-22). В строках 5-7 в регистры **t2-t4** записываются необходимые данные.

**Результат:**

|  |
| --- |
| Array = [12,7,17,4,3,6,5] mediana = 6 |
|  |

**Вывод:** В ходе лабораторной работы были получены навыки работы в симуляторе VSim/Jupiter и разработана программа на языке ассемблера RISC-V, реализующая нахождение медианы массива. Также, выявлены особенности RISC-V: существуют специализированные команды для операций с памятью — чтения или записи. Операции вида Read-Modify-Write («прочитать-изменить-записать») отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (т. н. архитектура load-and-store); Архитектура использует только модель [little-endian](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2#%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BE%D1%82_%D0%BC%D0%BB%D0%B0%D0%B4%D1%88%D0%B5%D0%B3%D0%BE_%D0%BA_%D1%81%D1%82%D0%B0%D1%80%D1%88%D0%B5%D0%BC%D1%83) — первый байт операнда в памяти соответствует наименее значащим битам значений регистрового операнда.