САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

Дисциплина: Низкоуровневое программирование **Tema:** Программирование RISC-V.

Работу выполнил: Чевычелов Д. А. Группа: 3530901/10003 Преподаватель: Коренев Д. А.

Санкт-Петербург 2022

Оглавление

1.	ТЗ	3
2.	Метод решения	3
3.	Руководство программисту	4
4.	Реализация программы 1	3-4
5.	Запуск программы 1	4
6	Реализация программы 2	5-6

1. T3

Найти сумму всех элементов массива. Если сумма меньше 50 — увеличить значения всех элементов на 7. 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.

Выделить определенную вариантом задания функциональность в подпрограмму, 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. Метод решения

Решение состоит из циклического прохода по элементам массива, во время которого реализуется подсчет суммы всего массива, а затем, в зависимости от выполнения условий задачи продим еще раз, где прибавляем к элементам массива заданное значение. Длина массива определяется из входных данных. Входные данные записываются в ячейки а2- а6. В а2 — минимальная сумма массива для условия задачи, а3 — единица для работы циклов, а4 — сумма входного массива, а5 — длина входного массива, а6 — входной массив (адрес его первого элемента). Далее следует первый цикл (loop1), в котором мы проходим по элеиентам массива, убавляя счетчик в а5. Когда в регистре счетчика появится 0, мы переходим в часть, где перезаписываются данные для loop2 и определяется надо ли заходить во второй массив (loop1_end). В конкретном случае если сумма элементов меньше 50, мы переходим во воторой массив, где добавляем к каждому элементу 7. По окончанию счетчика у второго массива, мы заканчиваем программу (ecall).

Пример: в результате работы программы (подпрограммы) массив [1, 2, 3, 4, 5, 6, 7, 8, 9] преобразуется в массив [8, 9, 10, 11, 12, 13, 14, 15, 16].

3. Руководство программисту

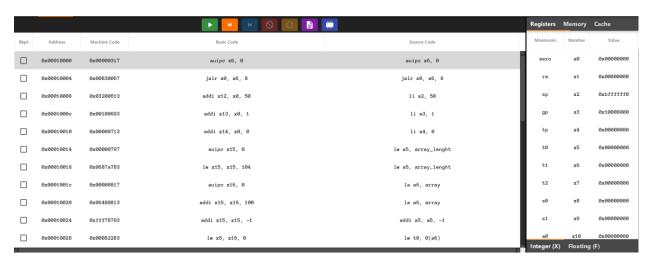
Начальные данные к программе: адрес нулевого элемента массива (и соответственно сам массив) и его длина. В реализации без подпрограммы и с ней адрес и длина хранятся в регистрах а5 и а6 соответственно.

4. Реализация программы 1

```
1 .text
2 __start:
3 .globl __start
4  li a2, 50 #для проверки суммы
5  li a3, 1 #чтобы делать loop
6  li a4, 0 #array summ
7  lw a5, array_lenght
8  la a6, array #addr of array[0]
9 loop1:
10  addi a5, a5, -1 #убавляем счетчик
11  lw t0, 0(a6) #записываем значение элемента
12  add a4, a4, t0 #записываем сумму
13  addi a6, a6, 4 #переходим к следующему
14  beq a5, zero, loop1_end #проверка на 0
15  j loop1
16 loop1_end:
17  lw a5, array_lenght #перезаписываем для loop2
18  la a6, array #перезаписываем для loop2
19  bge a2, a4, loop2 #переходим в loop2
20  li a0, 10
21  ecall #остановка
```

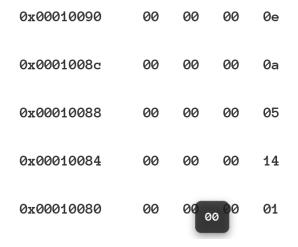
5. Запуск программы 1

Запуск программы через Jupiter осуществляется выбором меню run и нажатием Assemble.



Из этого состояния мы можем либо выполнить всю программу сразу или пройти ее пошагам (debug).

При этом наш исходный массив (в примере 1, 20, 5, 10, 14), распологается в ячейках:



6. Реализация программы 2 (с подпрограммой)

```
1 .text
2 __start:
3 .globl __start
4 call main
5 li a0, 10
6 ecall
```

```
8 .text
9 main:
10 li a2, 50 #для проверки суммы
11 li a3, 1 #для loops
12 li a4, 0 #array summ
13 lw a5, array_lenght
14 la a6, array #addr of array[0]
15 addi sp, sp, -16#выделение памяти в стеке для га
16 sw га, 12(sp) #сохраняем га для возврата
17 call subroutine #здесь га перезаписывается, поэтому для возврата нам надо его сохранить
18 lw га, 12(sp) #перезаписываем га
19 addi sp, sp, 16 # освобождение памяти в стеке
20 li a0, 0
21 ret
22 .rodata #исходные данные
23 array_lenght:
24 .word 5
25 .data
26 array:
27 .word 1, 20, 5, 10, 14
```

```
29 .text
30 subroutine: #подпрограмма
31 .globl subroutine
32 loop1:
33 addi a5, a5, -1 #убавляем счетчик
34 lw t0, 0(a6) #записываем значение элемента
35 add a4, a4, t0 #записываем сумму
36 addi a6, a6, 4 #переходим к следующему
37 beq a5, zero, loop1_end #проверка на 0
38 j loop1
39 loop1_end:
40 lw a5, array_lenght #перезаписываем для loop2
41 la a6, array #перезаписываем для loop2
42 bge a2, a4, loop2 #переходим в loop2
43 ret #остановка
44 loop2: #+7
45 addi a5, a5, -1 #убавляем счетчик
46 lw t0, 0(a6) #записываем значение элемента
47 addi a7, t0, 7 #+7
48 sw a7, 0(a6) #записываем по изначальному адресу
49 addi a6, a6, 4 #переходим к следующему
50 beq a5, zero, loop2_end #проверка на 0
51 j loop2
52 loop2_end:
53 ret #возврат
```

Отдельно стоит отметить, что при входе в main адрес возврата находится в регистре га, и возврат из подпрограммы осуществляется переходом на адрес, содержащийся в этом регистре, с помощью инструкции jalr. Однако прежде, чем это произойдет, значение га

будет перезаписано при вызове call и программа зациклится. Поэтому исходное значение га следует сохранить перед псевдоинструкцией call, и восстановить перед ret. Значение регистра можно сохранить памяти. Значение га нельзя сохранить в рабочем регистре, так как значение этого регистра может быть изменено вызываемой подпрограммой, а чтобы сохранить значение га в сохраняемом регистре, значение самого этого регистра необходимо сохранить и восстановить перед возвратом из main. Таким образом, значение га следует сохранить в памяти, для этого был использован стек.

Запуск программы 2 происходит аналогично запуску программы 1.