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

Тема: Программирование RISC-V.

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

Санкт-Петербург
2022

Оглавление

1. **ТЗ**

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 a2- a6. В a2 – минимальная сумма массива для условия задачи, a3 – единица для работы циклов, a4 – сумма входного массива, a5 – длина входного массива, a6 – входной массив (адрес его первого элемента). Далее следует первый цикл (loop1), в котором мы проходим по элеиентам массива, убавляя счетчик в a5. 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 второй 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 a5 и a6 соответственно.

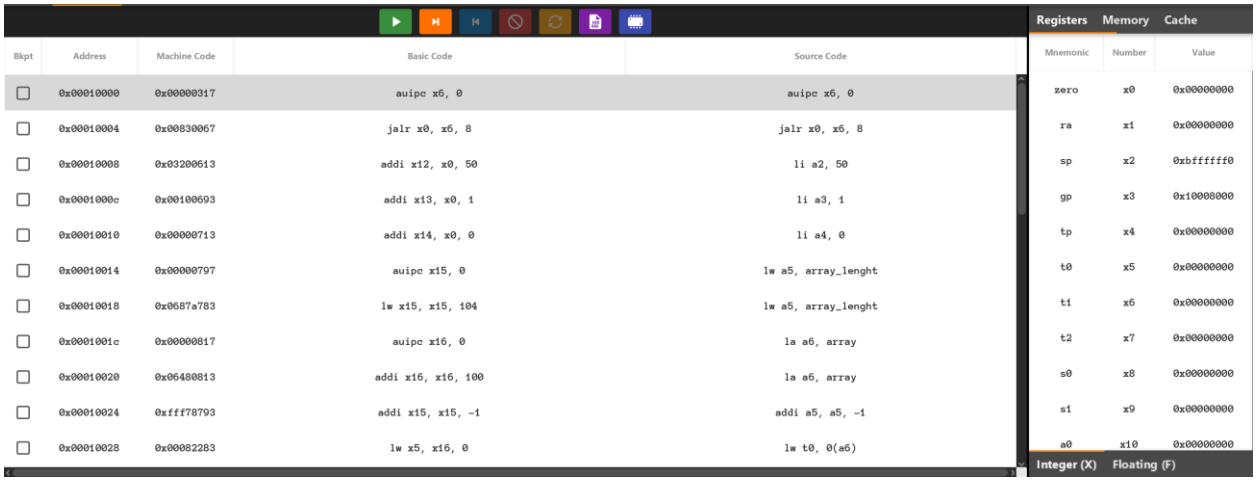
4. **Реализация программы 1**

```
.text
__start:
.globl __start
  li a2, 50 #для проверки суммы
  li a3, 1  #чтобы делать loop
  li a4, 0  #array summ
  lw a5, array_lenght
  la a6, array #addr of array[0]
loop1:
  addi a5, a5, -1 #убавляем счетчик
  lw t0, 0(a6)    #записываем значение элемента
  add a4, a4, t0  #записываем сумму
  addi a6, a6, 4  #переходим к следующему
  beq a5, zero, loop1_end #проверка на 0
  j loop1
loop1_end:
  lw a5, array_lenght #перезаписываем для loop2
  la a6, array        #перезаписываем для loop2
  bge a2, a4, loop2   #переходим в loop2
  li a0, 10
  ecall               #остановка
```

```
22 loop2:                #+7
23   addi a5, a5, -1     #убавляем счетчик
24   lw t0, 0(a6)        #записываем значение элемента
25   addi a7, t0, 7      #+7
26   sw a7, 0(a6)        #записываем по изначальному адресу
27   addi a6, a6, 4      #переходим к следующему
28   beq a5, zero, loop2_end #проверка на 0
29   j loop2
30 loop2_end:
31   li a0, 10
32   ecall
33 .rodata
34 array_lenght:
35   .word 5
36 .data
37 array:
38   .word 1, 20, 5, 10, 14
```

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 по-шагам (debug).

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

```
0x00010090      00    00    00    0e

0x0001008c      00    00    00    0a

0x00010088      00    00    00    05

0x00010084      00    00    00    14

0x00010080      00    00    00    01
```

6. Реализация программы 2 (с подпрограммой)

```
.text
__start:
.globl __start
  call main
  li a0, 10
  ecall
```

```
.text
main:
  li a2, 50 #для проверки суммы
  li a3, 1  #для loops
  li a4, 0  #array summ
  lw a5, array_lenght
  la a6, array #addr of array[0]
  addi sp, sp, -16#выделение памяти в стеке для ra
  sw ra, 12(sp)    #сохраняем ra для возврата
  call subroutine #здесь ra перезаписывается, поэтому для возврата нам надо его сохранить
  lw ra, 12(sp)    #перезаписываем ra
  addi sp, sp, 16 # освобождение памяти в стеке
  li a0, 0
  ret
.rodata    #исходные данные
array_lenght:
  .word 5
.data
array:
  .word 1, 20, 5, 10, 14
```

```
.text
subroutine:       #подпрограмма
.globl subroutine
loop1:
  addi a5, a5, -1 #убавляем счетчик
  lw t0, 0(a6)    #записываем значение элемента
  add a4, a4, t0  #записываем сумму
  addi a6, a6, 4  #переходим к следующему
  beq a5, zero, loop1_end #проверка на 0
  j loop1
loop1_end:
  lw a5, array_lenght #перезаписываем для loop2
  la a6, array        #перезаписываем для loop2
  bge a2, a4, loop2   #переходим в loop2
  ret                 #остановка
loop2:                #+7
  addi a5, a5, -1     #убавляем счетчик
  lw t0, 0(a6)        #записываем значение элемента
  addi a7, t0, 7      #+7
  sw a7, 0(a6)        #записываем по изначальному адресу
  addi a6, a6, 4      #переходим к следующему
  beq a5, zero, loop2_end #проверка на 0
  j loop2
loop2_end:
  ret         #возврат
```

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

будет перезаписано при вызове call и программа зациклится. Поэтому исходное значение ra следует сохранить перед псевдоинструкцией call, и восстановить перед ret. Значение регистра можно сохранить памяти. 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.

Запуск программы 2 происходит аналогично запуску программы 1.