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 А. А.
Группа: 3530901/10003
Преподаватель: Коренев Д. А.

Санкт-Петербург
2022

1. Техническое задание

Вариант 9: Реверс массива чисел.

2. Метод решения

Проход по элементам устроен методом циклического перехода. Элементы меняются местами с помощью «обменных» ячеек в цикле. Каждый проход цикла «arr_length + 1» уменьшается на 2, Тем самым мы получаем нужное количество итераций. Адрес второго операнда вычисляется каждый раз в цикле. Значение адреса первого операнда переходит на следующий в конце цикла.

3. Руководство программисту

Вход: массив (адрес его первого элемента), длина массива.
В реализации без подпрограммы адрес хранится в регистре a2 и адрес в a3.
В реализации с подпрограммой: нулевой элемент массива хранится в a0, а длина в a1.

4. Реализация программы 1

```
#Вариант 9. Реверс массива чисел
#Входные данные(массив) на строчке 40
.text
__start:
.globl __start

    la a2, array #a2 - адрес массива
    la a3, array_length
    lw a3, 0(a3) #a3 - длина массива(для определения второго
операнда)
    la a4, array_length
    lw a4, 0(a4) #a4 - длина массива(счетчик)
    addi a4, a4, 1 #a4 += 1
    add a6, a2, zero #a6 - адресс первого операнда
loop:
    addi a4, a4, -2 #счетчик - 2
    addi a3, a3, -1 #a3 -= 1
    bge zero, a4, loop_exit #если счетчик кончился
    slli a5, a3, 2 #a5 = a3 << 2 = a3 * 4
    add a5, a2, a5 #a5 = a5 + a2 -второй операнд

    lw t1, 0(a6) #t1 = array[1st]
    lw t0, 0(a5) #t0 = array[2nd]
    sw t1, 0(a5) #array[2nd] = t1
    sw t0, 0(a6) #array[1st] = t0

    addi a6, a6, 4 #след. эл-т

    j loop
```

```
loop_exit:
    li a0, 10 #x10 = 10
    ecall #x10 = 10, останов
#входные данные
.rodata #read-only
array_length:
    .word 8

.data
array:
    .word 1, 2, 3, 4, 5, 6, 7, 8
```

Начальное положение:

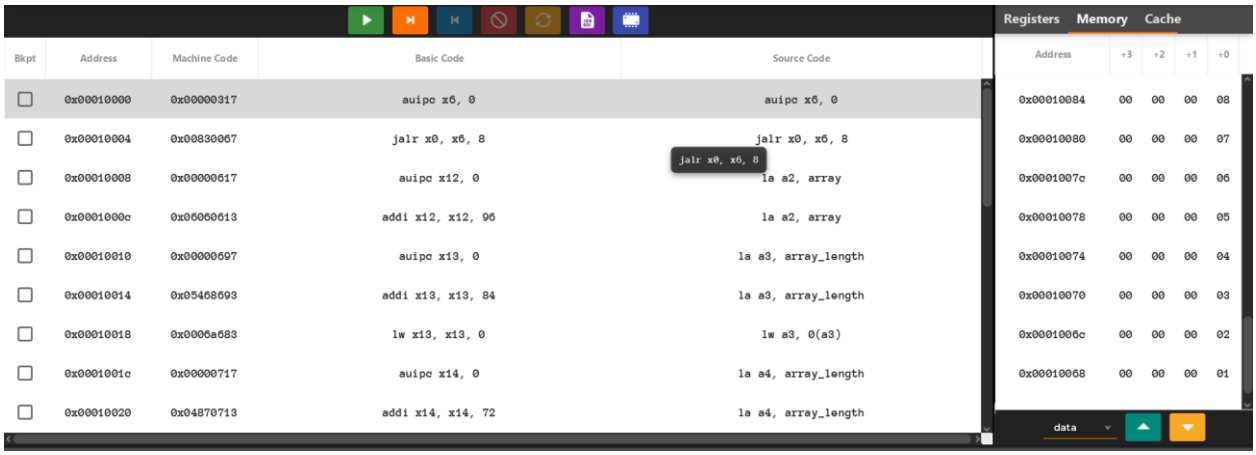


Рис. 1. Перед началом работы

Результат:

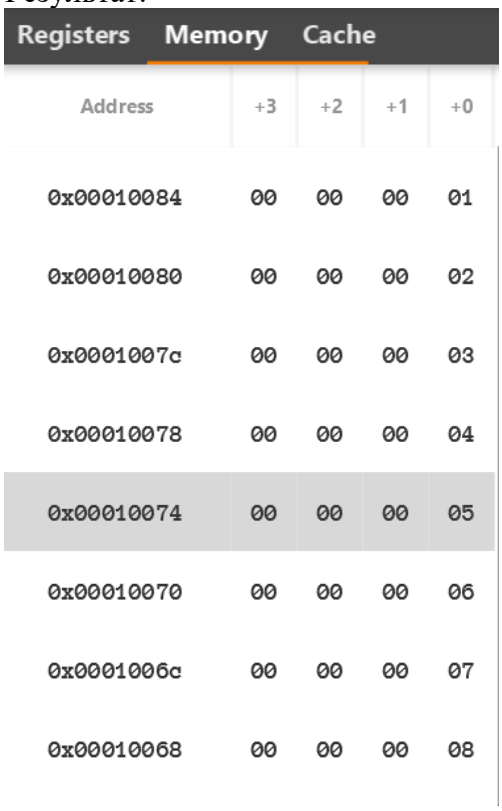


Рис. 2. Результат работы

Как видно из изображений, результат работы совпал с ожидаемым. Реверс массива получен верно.

5. Реализация программы 2 с подпрограммой

#setup

```
.text
__start:
.globl __start
    call main

finish:
    mv a1, a0 #a1 = a0 копирование
    li a0, 10 #a0 = 10
    ecall #exit
```

#main

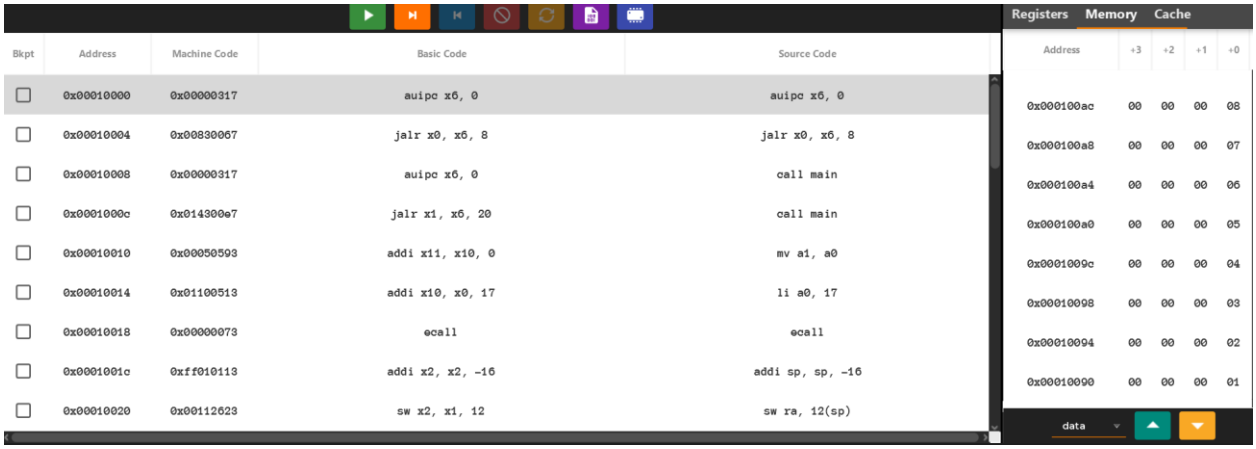
```
.text
main:
.globl main
    addi sp, sp, -16 #выделение памяти в стеке
    sw ra, 12(sp) #сохранение ra
    la a0, array
    lw a1, array_len
    lw a2, array_len
    call sub_fun #sub_fun(array, array_len) => auipc + jalr
    li a0, 0
    lw ra, 12(sp) #восстановим ra
    addi sp, sp, 16 #освобождение памяти в стеке
    ret # return 0

.rodata #read-only
array_len:
    .word 8
.data
array:
    .word 1, 2, 3, 4, 5, 6, 7, 8
```

#sub_fun

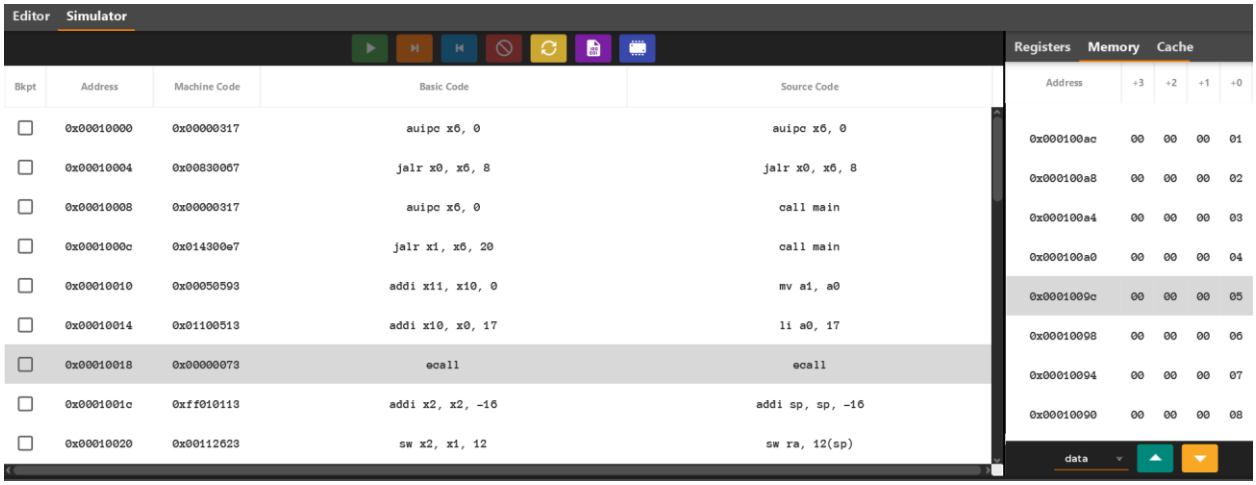
```
.text
sub_fun:
.globl sub_fun
    addi a2, a2, 1
    mv a6, a0
loop:
    addi a2, a2, -2 #счетчик - 2
    addi a1, a1, -1 #a3 -= 1
    bge zero, a2, loop_exit #
    slli a5, a1, 2 #a5 = a3 << 2 = a3 * 4
    add a5, a0, a5 #a5 = a5 + a2 (второй операнд)

    lw t1, 0(a6) #t1 = array[1st]
    lw t0, 0(a5) #t0 = array[2nd]
    sw t1, 0(a5) #array[2nd] = t1
```

```
    sw t0, 0(a6) #array[1st] = t0

    addi a6, a6, 4 #след. эл-т

    j loop #jump (jal x0) - goto loop
loop_exit:
    ret  #jalr zero, ra, 0
```

Начало работы программы:



Результат работы программы:



Как видно из изображений, результат работы совпал с ожидаемым. Реверс массива получен верно.

Вывод

В ходе данной работы был реализован алгоритм реверса на языке ассемблера RISC-V. Была написана программа (пункт 4), и ее представление в виде подпрограмм (пункт 5). Результаты работы программ совпали с ожидаемыми.