Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Лабораторная работа № 3

Программирование RISC-V

по дисциплине «Низкоуровневое программирование»

Выполнил
студент гр. 3530901/10002

_____ Котов К.А.
(подпись)

Руководитель

_____ Максименко С.Л.
(подпись)

«___» _____ 2022 г.

Санкт-Петербург
2022

Задача

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 задания

Вариант: 9 - Расчет биномиальных коэффициентов для данного показателя по треугольнику Паскаля.

Выполнение работы

Массив – под меткой «array»

Показатель – под меткой «pokaz»

Длина массива – под меткой «array_length»

Изначально мы имеем массив из 2 элементов, равных 1 и 0. Суть алгоритма заключается в том, чтобы проходить по массиву начиная с конца складывать значение текущего и предыдущего элемента и записывать результат в текущий элемент. Результат начинает формироваться с ячейки памяти 0x00010078.

Код программы для задачи 1 с комментариями:

```
.text
.globl __start
__start:
  la a3, array_length # загрузили длину массива в регистр a3
  lw a3, 0(a3)

  la a2, pokaz # загрузили показатель в регистр a2
  lw a2, 0(a2)
```

```
  la a4, array # загрузили адрес первого элемента массива в регистр a4

  li a5, 1 # загрузили 1 в регистр

  jal zero, loop_check1
loop1:
  addi a6, a3, -1

  jal zero, loop_check2
  loop2:
    slli a7, a6, 2 # получаем адрес i-ого элемента
    add a7, a4, a7

    lw t0, 0(a7) # загружаем значение по адресу
    lw t1, -4(a7)

    add t2, t1, t0

    sw t2, 0(a7)

    addi a6, a6, -1 # i++

  loop_check2:
     bgeu a6, a5, loop2 # for(int i = list.size - 1; i >= 1; i--)
  loop_exit2:

  addi a3, a3, 1

  addi a2, a2, -1
loop_check1:
  blt x0, a2, loop1 # while(pokaz > 0)
loop_exit1:

finish:
  li a0, 10
  li a1, 0
  ecall

.data
array_length:
  .word 2
pokaz:
  .word 4
array:
  .word 1, 0
```

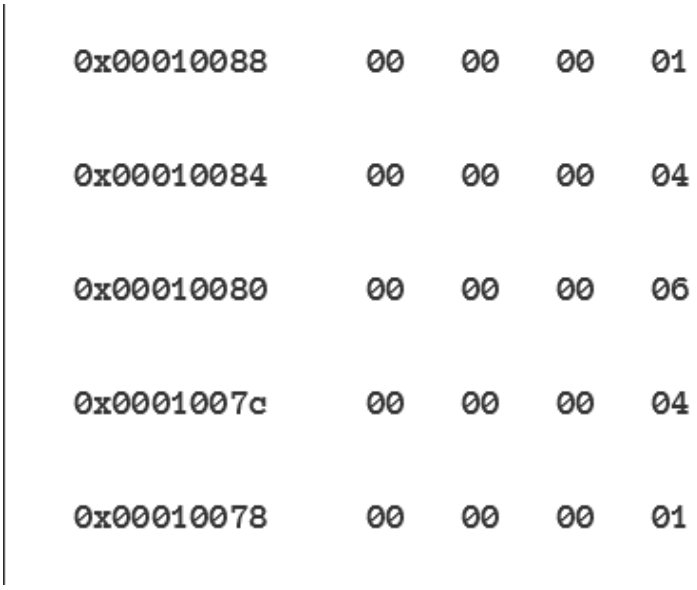

Рис.1 – Результат работы алгоритма с входным показателем равным 4

Аналогичный алгоритм используется для задачи 2 и в целом часть подпрограмма базируется на изменённой версии кода для задачи 1. Так же разработана вызывающая ее тестовую программа. Показатель передается через регистр a1, а ячейка памяти, в которой должен начать формироваться ответ передается через регистр a0.

```
# setup.s
      .text
      __start:
      .globl __start
       call main
      finish:
       mv a1, a0 # a1 = a0
       li a0, 17 # a0 = 17
       ecall # выход с кодом завершения

      # main.s
      .text
      main:
      .globl main
       addi sp, sp, -16 # выделение памяти в стеке
       sw ra, 12(sp) # сохранение ra

       lw a0, array_pointer #}
       lw a1, param         #} find_koeffs( *array, param );
       call find_koeffs     #}

       lw a0, array_pointer1 #}
       lw a1, param1         #} find_koeffs( *array1, param1 );
       call find_koeffs      #}
```

```
 li a0, 0 # }
 lw ra, 12(sp) # восстановление ra
 addi sp, sp, 16 # освобождение памяти в стеке

 ret # } return 0;
.data
param:
  .word 5
param1:
  .word 6
array_pointer:
  .word 0x000100f0
array_pointer1:
  .word 0x0001010c


#pascal_tri.s
# a0 - *array, a1 - pokaz
# find_koeffs
.text
find_koeffs:
.globl find_koeffs

  li a2, 2 # инициализация переменных
  li a3, 1 # загрузили 1 в регистр
  sw a3, 0(a0) # инициализировали массив [1] по адресу переданному из
главной программы

  jal zero, loop_check1
loop1:
  addi a4, a2, -1

  jal zero, loop_check2
  loop2:
    slli a5, a4, 2 # получаем адрес i-ого элемента
    add a5, a0, a5

    lw a6, 0(a5) # загружаем значение по адресу
    lw a7, -4(a5)

    add t0, a6, a7

    sw t0, 0(a5)

    addi a4, a4, -1 # i--

  loop_check2:
     bgeu a4, a3, loop2 # for(int i = list.size - 1; i >= 1; i--)
  loop_exit2:
```

```
  addi a2, a2, 1

  addi a1, a1, -1
loop_check1:
  blt x0, a1, loop1 # while(pokaz > 0)
loop_exit1:

ret
```

0x00010104	00	00	00	01	0x00010124	00	00	00	01
0x00010100	00	00	00	05	0x00010120	00	00	00	06
0x000100fc	00	00	00	0a	0x0001011c	00	00	00	0f
0x000100f8	00	00	00	0a	0x00010118	00	00	00	14
0x000100f4	00	00	00	05	0x00010114	00	00	00	0f
0x000100f0	00	00	00	01	0x00010110	00	00	00	06
					0x0001010c	00	00	00	01

Убедимся в правильности работы программы, с ячейки памяти 0x000100f0 формируются коэффициенты при показателе 5, а с ячейки памяти 0x0001010c коэффициенты при показателе 6.

Вывод

В ходе выполнения лабораторной работы была разработана программа на языке ассемблера RISC-V, выполняющая расчет биномиальных коэффициентов для данного показателя по треугольнику Паскаля и выполняющая запуск как цельной программы, так и подпрограммы, организованную в соответствии с ABI, разработана использующая её тестовая программа.