Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчет по лабораторной работе №3**

Тема: Программирование на Risc-V

Вариант 16

Выполнил студент гр. 3530901/90002 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_П. В. Рубинова

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. С. Степанов

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Постановка задачи:**

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.

**Формулировка задачи:**

Определение максимального простого числа, не превышающего заданное число N, методом решета Эратосфена (используется рабочий массив длины не более N).

**Теория:**

Решето Эратосфена — алгоритм нахождения всех [простых чисел](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) до некоторого целого числа N, который приписывают древнегреческому математику Эратосфену Киренскому.

В данном случае решето подразумевает фильтрацию всех чисел за исключением простых. По мере обработки массива чисел нужные числа (простые) остаются, а ненужные (составные) исключаются.

**1 часть**

**Текст программы:**

.text # Обозначение того, что ниже идет код программы

start: # Метка start

.globl start # Обозначение того, что метка start - глобальная

la a3, array\_lenght # Загружаем адрес, где находится длина массива

lw a3, 0(a3) # a3 = длина массива

li a2, 2 # Псевдоинструкция: в регистр a2 записываем 2

la a4, array # В регистре a4 - адрес первого элемента массива array[0]

sw x0, 0(a4) # Меняем значение первого элемента массива на 0 (array[0] = 0)

j loop\_1\_check # Псевдоинструкция: безусловный переход на метку loop\_1\_check

loop\_1: # Начало первого цикла --------------------------------------------------

addi a4, a4, 4 # Переход к следующему элементу масива - увеличиваем адрес на 4 (a4 = a4 + 4)

lw t0, 0(a4) # Записываем в регистр t0 значение этого элемента массива

beqz t0, loop\_2\_exit # Условный переход: если t0 == 0, то идем в loop\_2\_exit -> Цикл закончен, все элеменеты обработаны

mv a1, a2 # Псевдоинструкция: в регистр a1 записываем значение регистра a2

mv t1, a2 # t1 = a2

mv a5, a4 # a5 = a4

j loop\_2\_check # Безусловный переход на метку loop\_2\_check

loop\_2: # Начало второго цикла --------------------------------------------------

slli a7, t1, 2 # В регистре a7 записываем сдвинутый на 2 разряда влево регистр t1 (= t1 \* 4)

add a6, a5, a7 # В регистре a6 записываем сумму значений регистров a7 и a5 (= a5 + t1 \* 4)

sw x0, 0(a6) # Записываем в i-тый элемент массива значение 0

add t1, t1, a2 # Переход к следующему элементу масива (t1 = t1 + a2)

loop\_2\_check:

bltu t1, a3, loop\_2 # Условный переход: если t1 < a3, то переходим к метке loop\_2

loop\_2\_exit: # Конец второго цикла ----------------------------------------------

addi a2, a2, 1 # Переход к следующему элементу масива (a2 = a2 + 1)

loop\_1\_check:

bgeu a3, a2, loop\_1 # Условный переход: если a2 меньше a3 переходим на метку loop\_1

loop\_1\_exit: # Конец первого цикла ----------------------------------------------

li a0, 1

ecall # Печатаем результат

li a0, 10

ecall # Завершение работы программы

result:

.rodata # Обозначение, что дальше идут НЕизменяемые переменные

array\_lenght: # Длина массива (= 15)

.word 15

.data # Обозначение, что дальше идут изменяемые переменные

array: # Массив, заполненный единицами

.word 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

**Руководство:**

.text - указание ассемблеру размещать последующие инструкции в секции кода. Метка start: - точка начала выполнения программы.

В строках 4–9 написаны инструкции установки значений регистров a2-a4:

1. В a3 хранится длина массива array – array\_lenght.
2. В a2 записываем значение счетчика для внешнего цикла (= 2).
3. В a4 хранится адрес первого элемента массива array.
4. Устанавливаем значение массива для первого элемента равным 0 (так как 1 не простое число).

Далее совершаем безусловный переход на метку loop\_1\_check, где далее проверяем, равны ли значения внешнего счетчика и длины массива. Если да- заканчиваем работу программы. Если нет- заходим во внешний цикл loop\_1.

В строках 13–33 располагается внешний цикл loop\_1:

1. Переходим к следующему элементу массива, увеличивая адрес первого элемента массива на 4 (длина 1 слова).
2. Записываем в регистр t0 значение того элемента массива, адрес которого мы получили на предыдущем шаге.
3. Проверяем: если значение t0 == 0, то есть рассматриваемый элемент массива не является простым числом, переходим на метку выхода из внутреннего цикла loop\_2\_exit.
4. Иначе записываем значения: в регистре a1 будет храниться результат работы программы: a1 = a2; во внутренний счетчик записываем значение внешнего счетчика t1 = a2; записываем адрес i-того элемента массива a5 = a4.
5. Совершаем безусловный переход на метку loop\_2\_check, где далее проверяем: если значение внутреннего счетчика меньше размера массива, то переходим к метке loop\_2. Иначе заканчиваем внутренний цикл.

В строках 22–31 располагается внутренний цикл loop\_2:

1. В регистр a7 записываем сдвинутый на 2 разряда влево регистр t1 (= t1 \* 4).
2. В регистр a6 записываем сумму значений регистров a5 и a7: a6 = a5 + a7 = a5 + t1 \* 4. Таким образом мы к переданному адресу элемента массива прибавили умноженное на 4 значение внешнего счетчика и теперь в регистре a6 находится адрес следующего элемента массива.
3. В значение массива, адрес которого мы нашли на предыдущем шаге, записываем 0, так как индекс этого элемента массива не является простым числом.
4. Увеличиваем значение внутреннего счетчика на значение внешнего счетчика: t1 += a2.

В строках 38–42 заканчиваем работу программы:

1. Печатаем результат с помощью ecall с кодом 1 в a0.
2. Выходим из программы с помощью ecall с кодом 10 в a0.

В строках 44–50 определяем данные:

1. .rodata – Обозначение, что дальше идут неизменяемые данные: длина массива array\_lenght = 15 (.word означает, что мы используем 32-битные слова).
2. .data – Обозначение, что дальше идут изменяемые данные: массив длины array\_lenght, заполненный единицами.

**2 часть**

Функциональность программы, приведённой в первой части задания, была выделена в подпрограмму. Алгоритм не претерпел изменений.

**Текст программы:**

**main**

.text

start:

.globl start

call sub1 # Вызов подпрограммы sub1

result:

li a0, 10

ecall # Завершение работы программы

**sub1**

.text

sub1:

.globl sub1

addi sp, sp, -4 # Выделяем пространство для одного регистра

sw ra, 0(sp) # Сохраняем адрес возврата ra на стек

la a0, array

lw a1, array\_lenght

call sub2 # Вызов подпрограммы sub2

lw ra, 0(sp) # Восстанавливаем значение из стека

addi sp, sp, 4 # Освобождаем стек

li a0, 0 # a0 = 0

ret # return

.rodata

array\_lenght:

.word 15

.data

array:

.word 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

**sub2**

.text

sub2:

.globl sub2

mv a3, a1 # a3 = a1

li a2, 2 # a2 = 2

sw x0, 0(a0) # array[0] = 0

j loop\_1\_check # goto loop\_1\_check

loop\_1:

addi a0, a0, 4 # a4 += 4

lw t0, 0(a0) # t0 = array[i]

beqz t0, loop\_2\_exit # if(t0 == 0) goto loop\_2\_exit

mv a1, a2 # a1 = a2

mv t1, a2 # t1 = a2

mv a5, a0 # a5 = a4

j loop\_2\_check # goto loop\_2\_check

loop\_2:

slli a7, t1, 2 # a7 = t1 << 2 = t1 \* 4

add a6, a5, a7 # a6 = a7 + a5 = a5 + t1 \* 4

sw x0, 0(a6) # array[i] = 0

add t1, t1, a2 # t1 += a2

loop\_2\_check:

bltu t1, a3, loop\_2 # if( t1 < a3 ) goto loop\_2

loop\_2\_exit:

addi a2, a2, 1 # a2 += 1

loop\_1\_check:

bgeu a3, a2, loop\_1 # if( a2 <= a3 ) goto loop\_1

loop\_1\_exit:

li a0, 1

ecall # Печатаем результат

li a0, 0

ret # return

**Руководство:**

Тестовая программа main вызывает программу sub1, которая вызывает sub2 и передает необходимые параметры (длину массива и начальный адрес) через регистры a0 и a1.

Перед вызовом sub2, sub1 сохраняет значение регистра ra в стеке.

После вызова достает его оттуда и восстанавливает память в стеке.

**Вывод:**

В ходе данной лабораторной работы я познакомилась с принципом работы Risc-V и общими правилами реализации алгоритмов на ней на примере определения максимального простого числа, не превышающего заданное число N, методом решета Эратосфена.