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

# Лабораторная работа № 3

## RISC-V

по дисциплине «Низкоуровневое программирование»

Выполнил
студент гр. 3530901/00004_____ Кручинин К. А.
(подпись)

Руководитель _____ Соболь. В.
(подпись)

«___» _____ 2021 г.

Санкт-Петербург
2021

**Задача**

В соответствии с условием 7 варианта требуется написать программу для RISC-V осуществляющую определение k-й порядковой статистики in-place.

Лабораторная работа делится на две части:

1. Разработать программу на языке ассемблера RISC-V реализующую определенную вариантом задания функциональность, отладить программу в симуляторе 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.

**Алгоритм**

Необходимо смоделировать программу для RISC-V, которая определит такой элемент неупорядоченного массива, если бы он был k-м в упорядоченном. Для реализации сначала отсортируем массив произвольной длины алгоритмом сортировки вставками, опираясь на написанный на языке Kotlin алгоритм (рис. 1). Затем выведем k-й элемент массива.

```kotlin
fun sort(list: MutableList<Int>): List<Int> {
    for (i in 1 until list.size) {
        val insertElement = list[i]
        var j = i - 1
        while (j >= 0 && list[j] > insertElement) {
            list[j + 1] = list[j]
            j--
        }
        list[j + 1] = insertElement
    }
    return list
}
```

Рис.1 Сортировка вставками на языке Kotlin.

# Реализация программы.

```
.text
__start:
.globl __start
  lw a3, array_length # a3 = 6 | array_lenght
  la a4, array # a4 = <адрес 0-го элемента массива>
  li a2, 1 # a2 = 1 | i
  lw a6, k # a6 = k

loop_i_start:
  bgeu a2, a3, loops_exit # if(a2 >= a3) goto loops_exit | if(i >= arr_len)

  slli t0, a2, 2 # t0 = a2(i) << 2 = a2(i) * 4
  add t0, a4, t0 # t0 = a4 + t0 = a4 + a2(i) * 4 | [i]
  lw a7, 0(t0) # a7(insertElement) = arr[i]

  addi a5, a2, -1 # a5 = a2 - 1 | j = i - 1

loop_j:
  bltz a5, loop_i_end # if(j < 0) goto loop_i_end

  slli t1, a5, 2 # t1 = a5(j) << 2 = a5(j) * 4
  add t1, a4, t1 # t1 = a4 + t1 = a4 + a5(j) * 4 | [j]
  lw t3, 0(t1) # t3 = arr[j]

  bgeu a7, t3, loop_i_end # if(a7 >= t3) goto loop_i_end | if(insertElement >= arr[j])

  addi t2, t1, 4 # t2 = t1 + 4 | [j+1]
  sw t3, 0(t2) # arr[j+1] = arr[j]

  addi a5, a5, -1 # a5 = a5 - 1 | j--
  j loop_j

loop_i_end:
```

Рис. 2 Программа строки 1–33.

```
33 loop_i_end:
34   slli t1, a5, 2 # t1 = a5(j) << 2 = a5(j) * 4
35   add t1, a4, t1 # t1 = a4 + t1 = a4 + a5(j) * 4 | [j]
36   addi t2, t1, 4 # t2 = t1 + 4 | [j+1]
37
38   sw a7, 0(t2) # arr[j+1] = insertElement
39
40   addi a2, a2, 1  # i++
41   j loop_i_start
42
43 loops_exit:
44   slli t4, a6, 2 # t4 = a6(k) << 2 = a6(k) * 4
45   add t4, a4, t4 # t4 = a4 + t4 = a4 + a6(k) * 4 | [k]
46   addi t4, t4, -4 # t4 = t4 - 4 | [k-1] | т.к. нумерация в массиве с 0
47   lw a6, 0(t4) # a6 = arr[k]
48
49   li a0, 10 # x10 = 10
50   ecall # ecall при значении x10 = 10 => останов симулятора
51
52 .rodata
53 array_length:
54   .word 6
55 k:
56   .word 4
57
58 .data
59 array: #1, 6, 12, 36, 99, 110
60   .word 12, 36, 110, 1, 6, 99
```

Рис. 3 Программа строки 33–60.

Выполним запуск программы.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0x000100a8 | 0 | 0 | 0 | 99 | 0x000100a8 | 0 | 0 | 0 | 110 |
| 0x000100a4 | 0 | 0 | 0 | 6 | 0x000100a4 | 0 | 0 | 0 | 99 |
| 0x000100a0 | 0 | 0 | 0 | 1 | 0x000100a0 | 0 | 0 | 0 | 36 |
| 0x0001009c | 0 | 0 | 0 | 110 | 0x0001009c | 0 | 0 | 0 | 12 |
| 0x00010098 | 0 | 0 | 0 | 36 | 0x00010098 | 0 | 0 | 0 | 6 |
| 0x00010094 | 0 | 0 | 0 | 12 | 0x00010094 | 0 | 0 | 0 | 1 |

| Рис. 4 Массив данных до изменения. | Рис. 5 Массив данных после выполнения программы. |
|---|---|

В регистр a6 выводится результат – k-й элемент массива:

```
a6          x16          36
```

Рис. 6 k-й элемент отсортированного массива в регистре a6.

## Реализация программы с подпрограммами.

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

Рис. 7 Setup-программа.

```
1  .text
2  kStatMy_sub_sortMain:
3  .globl kStatMy_sub_sortMain
4    lw a3, array_length # a3 = 6 | array_lenght
5    la a4, array # a4 = <адрес 0-го элемента массива>
6    lw a6, k # a6 = k
7
8    add s1, s1, ra # s1 = <возвращаемый адрес на kStatMy_Sub_Setup>
9
10   call kStatMy_sub_sortSub  # call fun
11
12   mv ra, s1 # устанавливаем в возвращаемый регистр сохраненный адрес
13   ret # return
14
15 .rodata
16 array_length:
17   .word 6
18 k:
19   .word 4
20
21 .data
22 array: #1, 6, 12, 36, 99, 110
23   .word 12, 36, 110, 1, 6, 99
```

Рис. 8 Основная программа.

```
.text
kStatMy_sub_sortSub:
.globl kStatMy_sub_sortSub

  li a2, 1 # a2 = 1 | i

loop_i_start:
  bgeu a2, a3, loops_exit # if(a2 >= a3) goto loops_exit | if(i >= arr_len)

  slli t0, a2, 2 # t0 = a2(i) << 2 = a2(i) * 4
  add t0, a4, t0 # t0 = a4 + t0 = a4 + a2(i) * 4 | [i]
  lw a7, 0(t0) # a7(insertElement) = arr[i]

  addi a5, a2, -1 # a5 = a2 - 1 | j = i - 1

loop_j:
  bltz a5, loop_i_end # if(j < 0) goto loop_i_end

  slli t1, a5, 2 # t1 = a5(j) << 2 = a5(j) * 4
  add t1, a4, t1 # t1 = a4 + t1 = a4 + a5(j) * 4 | [j]
  lw t3, 0(t1) # t3 = arr[j]

  bgeu a7, t3, loop_i_end # if(a7 >= t3) goto loop_i_end | if(insertElement >= arr[j])

  addi t2, t1, 4 # t2 = t1 + 4 | [j+1]
  sw t3, 0(t2) # arr[j+1] = arr[j]

  addi a5, a5, -1 # a5 = a5 - 1 | j--
  j loop_j

loop_i_end:
  slli t1, a5, 2 # t1 = a5(j) << 2 = a5(j) * 4
  add t1, a4, t1 # t1 = a4 + t1 = a4 + a5(j) * 4 | [j]
  addi t2, t1, 4 # t2 = t1 + 4 | [j+1]

  sw a7, 0(t2) # arr[j+1] = insertElement

  addi a2, a2, 1  # i++
  j loop_i_start

loops_exit:
  slli t4, a6, 2 # t4 = a6(k) << 2 = a6(k) * 4
  add t4, a4, t4 # t4 = a4 + t4 = a4 + a6(k) * 4 | [k]
  addi t4, t4, -4 # t4 = t4 - 4 | [k-1] | т.к. нумерация в массиве с 0
  lw a6, 0(t4) # a6 = arr[k]

  ret
```

Рис. 9 Подпрограмма.

Выполним запуск программ.

| | | | | |
|---|---|---|---|---|
| 0x000100c8 | 0 | 0 | 0 | 99 |
| 0x000100c4 | 0 | 0 | 0 | 6 |
| 0x000100c0 | 0 | 0 | 0 | 1 |
| 0x000100bc | 0 | 0 | 0 | 110 |
| 0x000100b8 | 0 | 0 | 0 | 36 |
| 0x000100b4 | 0 | 0 | 0 | 12 |

Рис. 10 Массив данных до изменения.

| | | | | |
|---|---|---|---|---|
| 0x000100c8 | 0 | 0 | 0 | 110 |
| 0x000100c4 | 0 | 0 | 0 | 99 |
| 0x000100c0 | 0 | 0 | 0 | 36 |
| 0x000100bc | 0 | 0 | 0 | 12 |
| 0x000100b8 | 0 | 0 | 0 | 6 |
| 0x000100b4 | 0 | 0 | 0 | 1 |

Рис. 11 Массив данных после выполнения программы.

В регистр a6 выводится результат – k-й элемент массива:

```
a6          x16          36
```

Рис. 12 k-й элемент отсортированного массива в регистре a6.

**Вывод**

В ходе данной работы был реализован алгоритм сортировки вставками и вывод k-ого элемента массива на процессоре архитектуры RISC-V. Была создана как версия самостоятельной программы, так и версия подпрограммы с использующей её программой. Результаты полностью соответствуют ожидаемым.