Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №3

по дисциплине «Низкоуровневое программирование»

Программирование RISC-V

Работу выполнил:
Ильин В.П.
Группа:
35300901/10005
Преподаватель:
Коренев Д.А.

Санкт-Петербург
2022

Содержание

1. Техническое задание

Написать программу поиска k-й порядковой статистики массива in-place при помощи системы команд RISC-V.

2. Метод решения

По определению, *k-й порядковой статистикой* данного массива называется k-й элемент этого массива, если бы он был отсортирован. Будем искать ее по определению: отсортируем массив синтаксически простейшей сортировкой и возьмем элемент с индексом $(k-1)$.

3. Программа 1

В первом варианте программы (файл `first.s`) все вычисления происходят в одной функции. В режиме отладки программы вызывается при помощи команды

```
jupiter -debug first.s
```

3.1. Реализация программы 1

```
.text
__start:
.globl __start
    la a3 array_length
    lw a3, 0(a3) # a3 = array_length

    la a6, array # a6 = address of first element of array

    li a4, 0 # a4 = j - outer loop counter
outer_loop:
    bgeu a4, a3, outer_loop_exit # outer loop exit condition (!(j < array_length))

    li a5, 1 # a5 = i - inner loop counter
inner_loop:
    bgeu a5, a3, inner_loop_exit # inner loop exit condition (!(i < array_length))

    slli a7, a5, 2 # a7 = i * 4 bc sizeof(int) = 4
    add a7, a7, a6 # a7 = address of array[i]
    addi t0, a7, -4 # t0 = address of array[i - 1]

    lw t1, 0(t0) # t1 = array[i - 1]
    lw t2, 0(a7) # t2 = array[i]
    bgeu t2, t1, skip_swap # if (array[i] >= array[i - 1]) goto skip_swap;
    sw t1, 0(a7) # Mem[a7] = t1 = array[i - 1]
    sw t2, 0(t0) # Mem[t0] = t2 = array[i]
skip_swap:
    addi a5, a5, 1 # i++
    jal zero, inner_loop # next inner loop iteration
inner_loop_exit:
    addi a4, a4, 1 # j++
    jal zero, outer_loop # next outer loop iteration
outer_loop_exit:
    la t0 answer # t0 = address of answer cell
    la a3, k
    lw a3, 0(a3) # a3 = k

    addi a3, a3, -1 # a3 = k - 1
    slli a3, a3, 2 # a3 = (k - 1) * 4
    add a3, a3, a6 # a3 = (k - 1) * 4 + address of first element of array = address of (k-1)-th element
    lw t1, 0(a3) # t1 = (k-1)-th element
    sw t1, 0(t0) # Mem[t0] = t1 = (k-1)-th element

finish:
    li a0, 10 # finish program
    li a1, 0
    ecall

.rodata
array_length:
    .word 6
k:
    .word 3
.data
array:
    .word 4, 3, 5, 7, 2, 8
answer:
    .word 0
```

Рис. 3.1: Реализация программы 1

3.2. Работа программы 1

```
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin>jupiter -debug first.s
>>> locals
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin\first.s:
         answer -> 0x000100b0
        __start -> 0x00010008 (global)
inner_loop_exit -> 0x00010054
          array -> 0x00010098
outer_loop_exit -> 0x0001005c
     outer_loop -> 0x00010020
      skip_swap -> 0x0001004c
         finish -> 0x00010084
   array_length -> 0x00010090
              k -> 0x00010094
     inner_loop -> 0x00010028
>>> break 0x00010084
>>> memory 0x000100b0 1
             Value (+0) Value (+4) Value (+8) Value (+c)
[0x000100b0] 0x00000000 0x00000000 0x00000000 0x00000000
>>> c
>>> memory 0x000100b0 1
             Value (+0) Value (+4) Value (+8) Value (+c)
[0x000100b0] 0x00000004 0x00000000 0x00000000 0x00000000
>>> exit
```

Рис. 3.2: Работа программы 1

4. Программа 2

Во втором варианте программа реализована при помощи двух подпрограмм: в одной содержится функция `main` (файл `main.s`), вызывающая подпрограмму, подсчитывающую ответ (ее реализация представлена в файле `find_order_statistic.s`). Стартовая метка, на которой происходит вызов главной подпрограммы, находится в файле `second.s`. В режиме отладки программа вызывается при помощи команды:

```
jupiter -debug -start __start second.s main.s find_order_statistic.s
```

4.1. Реализация программы 2

```
# second.s
.text
__start:
.globl __start
    call main
```

Рис. 4.1: Файл тестовой программы

```
# main.s
.text
main:
.globl main
    addi sp, sp, -16 # malloc
    sw ra, 12(sp)    # save ra

    la a0, array
    lw a1, array_length
    lw a2, k
    call find_order_statistic     # a0 = findOrderStatistic(array, array_length, k);

    la t0, answer
    sw a0, 0(t0) # Mem[t0] = a0 = (k-1)-th element
finish:
    lw ra, 12(sp)    # restore ra
    addi sp, sp, 16  # free
    li a0, 0
    ret      # return 0;

.rodata
array_length:
    .word 6
k:
    .word 3
.data
array:
    .word 4, 3, 5, 7, 2, 8
answer:
    .word 0
```

Рис. 4.2: Файл с основной подпрограммой

```
# find_order_statistic.s
.text
find_order_statistic:
.globl find_order_statistic
    # a0 = address of array
    # a1 = array_length
    # a2 = k

    li a3, 0 # a3 = j - outer loop counter
outer_loop:
    bgeu a3, a1, outer_loop_exit # outer loop exit condition (!(j < array_length))

    li a4, 1 # a4 = i - inner loop counter
inner_loop:
    bgeu a4, a1, inner_loop_exit # inner loop exit condition (!(i < array_length))

    slli a5, a4, 2 # a5 = i * 4 bc sizeof(int) = 4, now a5 is byte-shift address of i-th element
    add a5, a5, a0 # a5 = address of array[i]
    addi a6, a5, -4 # a6 = address of array[i - 1]

    lw t1, 0(a6) # t1 = array[i - 1]
    lw t2, 0(a5) # t2 = array[i]
    bgeu t2, t1, skip_swap # if (array[i] >= array[i - 1]) goto skip_swap;
    sw t1, 0(a5) # Mem[a5] = t1 = array[i - 1]
    sw t2, 0(a6) # Mem[a6] = t2 = array[i]
skip_swap:
    addi a4, a4, 1 # i++
    jal zero, inner_loop # next inner loop iteration
inner_loop_exit:
    addi a3, a3, 1 # j++
    jal zero, outer_loop # next outer loop iteration
outer_loop_exit:
    addi a2, a2, -1 # a2 = k - 1
    slli a2, a2, 2 # a2 = (k - 1) * 4
    add a2, a2, a0 # a2 = (k - 1) * 4 + address of first element of array = address of (k-1)-th element
    lw a0, 0(a2) # a0 = (k-1)-th element = answer
    ret # return (k-1)-th element;
```

Рис. 4.3: Файл с функциональной подпрограммой

4.2. Работа программы 2

```
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin>jupiter -debug -start __start riscv/second.s riscv/main.s riscv/find_order_st
>>> locals
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin\riscv\second.s:
            __start -> 0x00010008 (global)
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin\riscv\find_order_statistic.s:
    inner_loop_exit -> 0x0001008c
find_order_statistic -> 0x00010054 (global)
    outer_loop_exit -> 0x00010094
         outer_loop -> 0x00010058
          skip_swap -> 0x00010084
         inner_loop -> 0x00010060
C:\Users\ilyin\Downloads\Jupiter-3.1-win\image\bin\riscv\main.s:
             answer -> 0x000100c8
              array -> 0x000100b0
               main -> 0x00010010 (global)
             finish -> 0x00010044
       array_length -> 0x000100a8
                  k -> 0x000100ac
>>> break 0x00010044
>>> memory 0x000100c8 1
            Value (+0) Value (+4) Value (+8) Value (+c)
[0x000100c8] 0x00000000 0x00000000 0x00000000 0x00000000
>>> c
>>> memory 0x000100c8 1
            Value (+0) Value (+4) Value (+8) Value (+c)
[0x000100c8] 0x00000004 0x00000000 0x00000000 0x00000000
>>> exit
```

Рис. 4.4: Работа программы 2