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. 3530901/90004

_____ Сергеев И.А.
(подпись)

Руководитель

_____ Алексюк А.О.
(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. Выделить определенную вариантом задания функциональность в подпрограмму, организованную в соответствии с, 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-ым в упорядоченном. Для реализации сначала отсортируем массив произвольной длины алгоритмом сортировки вставками, опираясь на написанный на языке Python алгоритм (рис.1). Затем выведем k-ый элемент массива.

```python
def insertion_sort(nums):
    # Сортировку начинаем со второго элемента, т.к. считается, что первый элемент уже отсортирован
    for i in range(1, len(nums)):
        item_to_insert = nums[i]
        # Сохраняем ссылку на индекс предыдущего элемента
        j = i - 1
        # Элементы отсортированного сегмента перемещаем вперёд, если они больше
        # элемента для вставки
        while j >= 0 and nums[j] > item_to_insert:
            nums[j + 1] = nums[j]
            j -= 1
        # Вставляем элемент
        nums[j + 1] = item_to_insert
```

Рис. 1 Алгоритм сортировки вставкой (python)

1. Листинг программы sort.s:

```
.text
start:
.globl start
la a3, arr # указатель на 0й элем массива !load adress -> загружает адресс
lw a4, arr_length # счетчик !load word -> загружает значение
lw a7, k # a7 = k

li t1, 1 # i = 1

loop_i:
slli t3, t1, 2 # смещение по i ! t3 = i4
add t3, t3, a3 # адресс arr[i] ! t3 = указатель на arr[0] + смещение по i
lw a5, 0(t3) # a5 = arr[i] ! item_to_insert
mv t3, zero # t3 clk
addi t2, t1, -1 # j = i - 1

loop_j:
bltz t2, loop_i_end # if(j < 0) -> loop_i_end
slli t3, t2, 2 # смещение по j ! t3 = j4
add t3, t3, a3 # адресс arr[j] ! t3 = указатель на arr[0] + смещение по j
lw a6, 0(t3) # a6 = arr[j]
ble a6, a5, loop_i_end # if(arr[j] <= arr[i]) -> loop_i_end
addi t3, t3, 4 # в t3 был адресс на arr[j], стал arr[j+1]
sw a6, 0(t3) # arr[j+1] = arr[j]
addi t2, t2, -1 # j = j - 1
j loop_j

loop_i_end:
mv t3, zero # t3 clk
slli t3, t2, 2 # смещение по j ! t3 = j4
add t3, t3, a3 # адресс arr[j] ! t3 = указатель на arr[0] + смещение по j
addi t3, t3, 4 # в t3 был адресс на arr[j], стал arr[j+1]
sw a5, 0(t3) # arr[j+1] = a5 ! arr[j+1] = item_to_insert
mv t3, zero # t3 clk
addi t1, t1, 1 # i = i + 1
bgt a4, t1, loop_i # if (arr_length > i) -> loop_i
```

Рисунок 1.1. Программа sort (1)

```
loop_exit:
slli t3, a7, 2 # смещение по k ! t3 = k4
add t3, t3, a3 # адресс arr[k] ! t3 = указатель на arr[0] + смещение по k
addi t3, t3, -4 # уменьшили адресс на 1, т.к. к по порядковому номеру
lw a5, 0(t3) # a5 = arr[k]
li a0, 10
ecall

.rodata
arr_length:
.word 6
k:
.word 3
.data
arr: # 27  45  53  67  90  108
.word 108, 53, 45, 27, 67, 90
```

Рисунок 1.2. Программа sort (2)

Проведём анализ изменения памяти после выполнения работы. Рисунком 1.3 представлено состояние памяти до выполнения алгоритма, а рисунком 1.4 после. Таким образом, проведя анализ можно сделать вывод о корректности работы программы.

0x0001016c	0	0	0	90
0x00010168	0	0	0	67
0x00010164	0	0	0	27
0x00010160	0	0	0	45
0x0001015c	0	0	0	53
0x00010158	0	0	0	108

Рис. 1.3 До выполнения программы

0x0001016c	0	0	0	108
0x00010168	0	0	0	90
0x00010164	0	0	0	67
0x00010160	0	0	0	53
0x0001015c	0	0	0	45
0x00010158	0	0	0	27

Рис. 1.4. После выполнения программы

В регистр a5 выводится k-ое значение массива

a5	x15	53

Рис. 1.5. k-ое значение массива

2. Реализация подпрограммы

Тестовая программа:

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

Рис. 2.1. Программа setup

Подпрограмма:

```
.text
sort_main:
.globl sort_main
la a3, arr # указатель на 0й элем массива !load adress -> загружает адресс
lw a4, arr_length # счетчик !load word -> загружает значение
lw a7, k # a7 = k

addi sp, sp, -16 #делаем для того чтобы не было зацикливания и сохраняем
sw ra, 12(sp) # загружаем в память
call sort_sub
lw ra, 12(sp)
addi sp, sp, 16 # возвращаем все обратно ! для рекурсии?
ret # return

.rodata
arr_length:
.word 6
k:
.word 3
.data
arr:
.word 108, 53, 45, 27, 67, 90
```

Рис. 2.2. Подпрограмма sort_main

sort_sub:

```
.text
sort_sub:
.globl sort_sub

li t1, 1 # i = 1

loop_i:
slli t3, t1, 2  # смещение по i ! t3 = i4
add t3, t3, a3  # адресс arr[i] ! t3 = указатель на arr[0] + смещение по i
lw a5, 0(t3)    # a5 = arr[i] ! item_to_insert
mv t3, zero     # t3 clk
addi t2, t1, -1 # j = i - 1

loop_j:
bltz t2, loop_i_end # if(j < 0) -> loop_i_end
slli t3, t2, 2 # смещение по j ! t3 = j4
add t3, t3, a3 # адресс arr[j] ! t3 = указатель на arr[0] + смещение по j
lw a6, 0(t3) # a6 = arr[j]
ble a6, a5, loop_i_end # if(arr[j] <= arr[i]) -> loop_i_end
addi t3, t3, 4 # в t3 был адресс на arr[j], стал arr[j+1]
sw a6, 0(t3) # arr[j+1] = arr[j]
addi t2, t2, -1 # j = j - 1
j loop_j

loop_i_end:
mv t3, zero # t3 clk
slli t3, t2, 2 # смещение по j ! t3 = j4
add t3, t3, a3 # адресс arr[j] ! t3 = указатель на arr[0] + смещение по j
addi t3, t3, 4 # в t3 был адресс на arr[j], стал arr[j+1]
sw a5, 0(t3) # arr[j+1] = a5 ! arr[j+1] = item_to_insert
mv t3, zero # t3 clk
addi t1, t1, 1 # i = i + 1
bgt a4, t1, loop_i # if (arr_length > i) -> loop_i

loop_exit:
slli t3, a7, 2 # смещение по k ! t3 = k4
add t3, t3, a3 # адресс arr[k] ! t3 = указатель на arr[0] + смещение по k
addi t3, t3, -4 # уменьшили адресс на 1, т.к. k по порядковому номеру
lw a5, 0(t3) # a5 = arr[k]
ret
```

Рис 2.3. Программа sort_sub

Проведём анализ изменения памяти после выполнения работы. Рисунком 2.4 представлено состояние памяти до выполнения алгоритма, а рисунком 2.5 после. Таким образом, проведя анализ можно сделать вывод о корректности работы программы

0x000100d8	0	0	0	90	0x000100d8	0	0	0	108
0x000100d4	0	0	0	67	0x000100d4	0	0	0	90
0x000100d0	0	0	0	27	0x000100d0	0	0	0	67
0x000100cc	0	0	0	45	0x000100cc	0	0	0	53
0x000100c8	0	0	0	53	0x000100c8	0	0	0	45
0x000100c4	0	0	0	108	0x000100c4	0	0	0	27

Рис. 2.4 До выполнения программы

Рис. 2.5. После выполнения программы

a5	x15	53

Рис. 2.6. k-ое значение массива

Вывод

В ходе данной работы был реализован алгоритм сортировки вставками и вывод k-ого элемента массива на процессоре архитектуры RISC-V. Изначально приведен код написанный на языке Python, а затем интерпретирован под RISC-V. При реализации подпрограммы были написаны инициализирующий код, код завершения, подпрограмма main и тестируемая подпрограмма. Результаты выполнения программ полностью соответствуют ожидаемым.