

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

Отчёт по лабораторной работе № 2

Дисциплина: Низкоуровневое программирование

Тема: Программирование RISC-V

Вариант: 4

Выполнил студент гр. 3530901/90002 _____ Е.К. Борисов
(подпись)

Принял старший преподаватель _____ Д.С. Степанов
(подпись)

“ _____ ” _____ 2021 г.

Санкт-Петербург
2021

Цель работы:

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

Вариант 4:

Реализовать алгоритм сортировки вставкой.

Задание:

Необходимо смоделировать программу для EDSAC, которая реализует алгоритм сортировки массива вставкой. Отсортированный массив должен находиться в тех же ячейках, где находился изначально.

Решение:

Для написания программы для RISC-V, был изначально написан алгоритм на языке python, комментарии в программе опираются на написанный на python-е алгоритм (рис.1).

Алгоритм сортировки вставками - алгоритм, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

```
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)

Алгоритм:

1. Получаем значения `arr[i]`, `(item_to_insert)` , `arr[j]`, устанавливаем `j = i`.
2. Проверяем условие `item_to_insert > arr[j]` , если больше увеличиваем значение `i` и переходим к следующей итерации. Иначе идем дальше.
3. Получаем значение элемента `arr[j]`.
4. Проверяем `j = 0`, если `true`, то переходим к пункту 7, иначе идем дальше.
5. Проверяем `item_to_insert > arr[j]`, если `true`, то переходим к пункту 7, иначе заходим в цикл `while`.
6. Устанавливаем в `arr[j + 1]` значение элемента `arr[j]`, уменьшаем `j` на единицу и прыгаем обратно в пункт 3.
7. Устанавливаем в `arr[j + 1]` значение элемента для вставки, увеличиваем `i` на единицу.
8. Проверяем `i >= arr.lenght`, если `true`, завершаем работу программы, иначе переходим к следующей итерации.

Решение.

Для написания кода программы был выбран редактор кода `atom` и добавлен синтаксис RISC-V с помощью библиотеки “`language-riscv`”. Ниже приведен код написанной программы.

```

1  .text
2  start:
3  .globl start
4  la a3, array_length
5  lw a3, 0(a3) # a3 = arr.length
6  la a4, array # arr[0] adress
7  li a2, 1 # i
8  loop1:
9  bgeu a2, a3, loop_exit # if( i >= arr.length ) goto loop_exit
10 slli a5, a2, 2 # a5 = a2 << 2 = a2 * 4
11 add a5, a4, a5 # a5 = a4 + a5 = a4 + a2 * 4
12 lw t1, -4(a5) # t1 = array[i-1]
13 lw t0, 0(a5) # item_to_insert
14 mv t2, a2 #j_for_while
15 bgeu t0, t1, loop_exit3 # item_to_insert > arr[j]
16 loop2:
17 slli t3, t2, 2 # a5 = a2 << 2 = a2 * 4
18 add t3, a4, t3 # a5 = a4 + a5 = a4 + a2 * 4
19 lw t6, -4(t3) # arr[j]
20 beqz t2, loop_exit2 # j == 0 => end while
21 bgeu t0, t6, loop_exit2 # item_to_insert > arr[j]
22 sw t6, 0(t3) # array[j + 1] = t6
23 addi t2, t2, -1 # j -= 1
24 mv t5, t3 # arr[j+1] to insertion at the end of while
25 jal zero, loop2 # back to while
26 loop_exit2:
27 sw t0, -4(t5)
28 loop_exit3:
29 addi a2, a2, 1 # i += 1
30 jal zero, loop1 # goto loop1
31 loop_exit:
32 finish:
33 li a0, 10 # x10 = 10
34 li a1, 0 # x11 = 0
35 ecall # end of programm
36 .rodata
37 array_length:
38 .word 13
39 .data
40 array:
41 .word 10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4

```

Рис. 2 Код первой программы.

“text” – указатель на основную часть кода, **“start”** – точка начала выполнения программы. В строках 4-7 написаны псевдоинструкции установки значений регистров.

a2 = *i*.

a3 = длина массива.

a4 = адрес первого элемента массива.

Разберем далее программу построчно:

8-35 - Основная часть программы.

8 - Вход в цикл `for`. Инструкция в строке служит условием выхода из цикла `for` (*i* >= длина массива), если `true`, переходим к 33 строчке. Иначе идем далее по программе.

9-11 - Инструкции, отвечающие за вычисление адреса `arr[i]` при помощи сдвига влево *i*-ого числа и прибавления этого выражения к нулевому элементу массива. Значение заносится в регистр **a5**.

12-14 - В регистры *t1*, *t0*, *t2* записывается `arr[i - 1]`, `arr[i]` и *j* = *i* соответственно. Далее в строке 15 проверяется условие (`item_to_insert > arr[j]`), если `true` – мы переходим к строке 29, где увеличиваем значение *i* и переходим к следующей итерации. Иначе заходим в цикл `While`.

17-19 - получаем значение `arr[j]` и записываем его в регистр *t6*

20-21 – Условия цикла `while`: *j* = 0, `item_to_insert > arr[j]`, если хоть одно из условий `true` – выходим из `while` в строку 27.

22-25 – Записываем в `arr[j + 1]` значение элемента `arr[j]`, уменьшает *j* на единицу и записываем в регистр *t5* ссылку на элемент `arr[j + 1]` для последующей вставки числа в это значение после выхода из цикла. Далее прыгаем в строку 16.

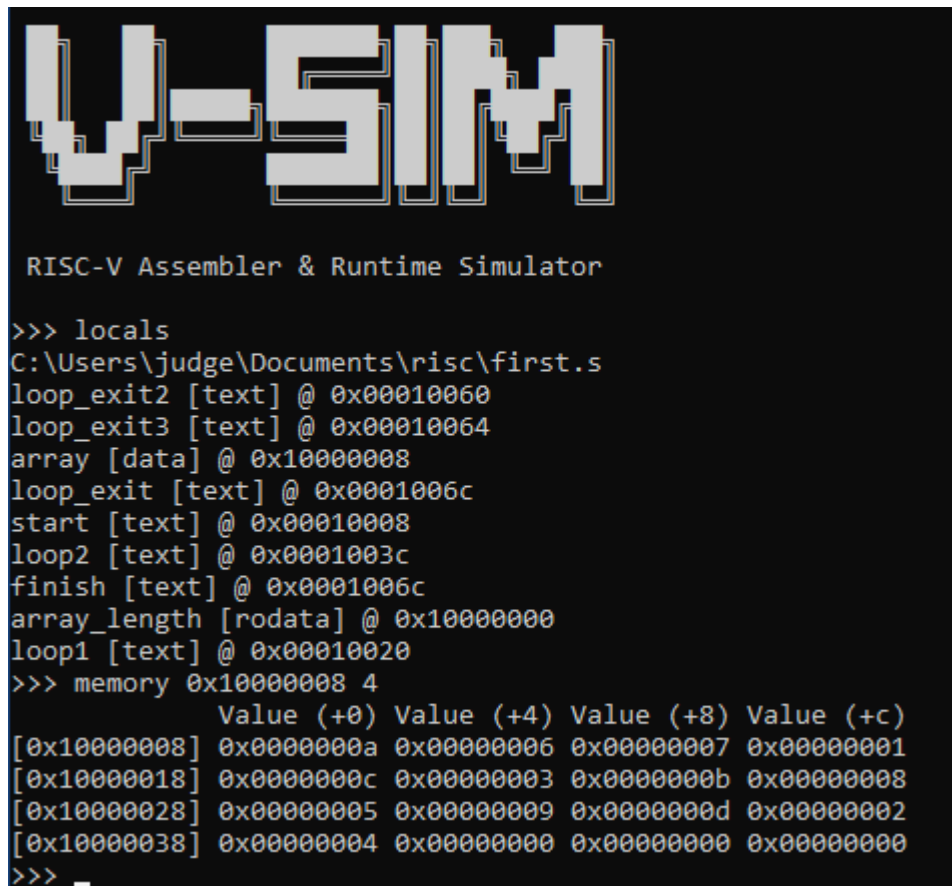
26-30 – Вставляем элемент для вставки (`item_to_insert`) в ячейку `arr[j + 1]`, увеличиваем *i* на единицу и переходим к следующей итерации обратно в строку 8.

33-35 – Завершение работы программы

36-41 – Данные программы. *.word* - означает, что мы используем 32-битные слова (4 байта). То есть они занимают 4 восьмибитных секции. *.rodata* – константы, *.data* – переменные.

Примеры работы программы

С помощью команд `locals`, `breakpoint` и с посмотрим заданный массив в начале и конце работы программы.



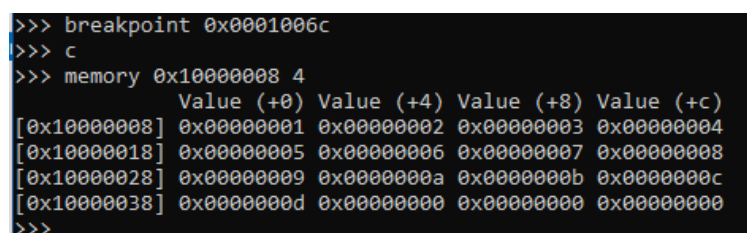
```
U-SIM
RISC-V Assembler & Runtime Simulator

>>> locals
C:\Users\judge\Documents\risc\first.s
loop_exit2 [text] @ 0x00010060
loop_exit3 [text] @ 0x00010064
array [data] @ 0x10000008
loop_exit [text] @ 0x0001006c
start [text] @ 0x00010008
loop2 [text] @ 0x0001003c
finish [text] @ 0x0001006c
array_length [rodata] @ 0x10000000
loop1 [text] @ 0x00010020
>>> memory 0x10000008 4
      Value (+0) Value (+4) Value (+8) Value (+c)
[0x10000008] 0x0000000a 0x00000006 0x00000007 0x00000001
[0x10000018] 0x0000000c 0x00000003 0x0000000b 0x00000008
[0x10000028] 0x00000005 0x00000009 0x0000000d 0x00000002
[0x10000038] 0x00000004 0x00000000 0x00000000 0x00000000
>>> _
```

Рис. 3 Значения элементов массива в начале работы программы.

$Arr = [10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4]$ – значение элементов в десятичном представлении

Перейдем с помощью команды `breakpoint` к метке `finish`, чтобы посмотреть значения массива в конце работы программы.



```
>>> breakpoint 0x0001006c
>>> c
>>> memory 0x10000008 4
      Value (+0) Value (+4) Value (+8) Value (+c)
[0x10000008] 0x00000001 0x00000002 0x00000003 0x00000004
[0x10000018] 0x00000005 0x00000006 0x00000007 0x00000008
[0x10000028] 0x00000009 0x0000000a 0x0000000b 0x0000000c
[0x10000038] 0x0000000d 0x00000000 0x00000000 0x00000000
>>>
```

Рис. 3 Значения элементов массива в конце работы программы.

Arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] - значение элементов в десятичном представлении.

Как мы видим, массив отсортировался корректно.

Часть 2

```
.text
start:
.globl start
call main
finish:
mv a1, a0 # a1 = a0
li a0, 17 # a0 = 17
ecall # end of program
```

Рис.5 Тестирующая программа.

В программе вызывается подпрограмма main с помощью команды call.

Псевдоинструкция call соответствует следующей паре инструкций:

```
auipc ra, %pcrel_hi(main)
jalr ra, ra, %pcrel_lo(main)
```

Исполненные одна за другой, эти инструкции обеспечивают безусловный переход (jump) на метку main с сохранением адреса следующей за jalr инструкции в регистре ra (синоним x1).

Когда выполнение подпрограммы завершится, исполнение кода тестирующей программы перейдет к метке finish, в которой работа программы завершается.


```

1  # main.s
2  .text
3  main:
4  .globl main
5      la a0, array # adress arr[0]
6      la a1, array_length
7      lw a1, 0(a1) # arr.length
8
9      addi sp, sp, -16 # allocating memory on the stack
10     sw ra, 12(sp) # saving ra
11
12     call insertion
13
14     lw ra, 12(sp) # recovery ra
15     addi sp, sp, 16 #freeing memory on the stack
16
17     li a0, 0
18     ret
19
20 .rodata
21 array_length:
22     .word 13
23 .data
24 array:
25     .word 10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4

```

Рис. 4 Подпрограмма main.

Здесь задаются данные работы программы: адрес первого элемента массива и длинна массива в соответствующие регистры *a0*, *a1*. После этого происходит вызов подпрограммы *insertion*. Однако ее вызов отличается от вызова *main* в тестирующей подпрограмме. Это связано с тем, что мы уже находимся в подпрограмме, а регистр с кодом возврата (*ra*) один, поэтому перед тем, как мы вызовем еще одну подпрограмму, нам нужно где-то сохранить этот код. *ret* - возврат из подпрограммы.

```

.text
insertion:
.global insertion
    li a2, 1 # a2 = 1
loop1:
    bgeu a2, a1, loop_exit # if( i > arr.lenght ) goto loop_exit
    slli a5, a2, 2 # a5 = a2 << 2 = a2 * 4
    add a5, a0, a5 # a5 = a4 + a5 = a4 + a2 * 4
    lw t1, -4(a5) # t1 = array[i-1]
    lw t0, 0(a5) # item_to_insert
    mv t2, a2 #j_for_while
    bgeu t0, t1, loop_exit3 # item_to_insert > arr[j]
loop2:
    slli t3, t2, 2 # a5 = a2 << 2 = a2 * 4
    add t3, a0, t3 # a5 = a4 + a5 = a4 + a2 * 4
    lw t6, -4(t3) # arr[j]
    beqz t2, loop_exit2 # j == 0 => end while
    bgeu t0, t6, loop_exit2 # item_to_insert > arr[j]
    sw t6, 0(t3) # array[j + 1] = t6
    addi t2, t2, -1 # j -= 1
    mv t5, t3 # save arr[j+1] to insertion at the end of while
    jal zero, loop2 # back to while
loop_exit2:
    sw t0, -4(t5)
loop_exit3:
    addi a2, a2, 1 # i += 1
    jal zero, loop1 # goto loop1
loop_exit:
    ret

```

Рис. 5 Подпрограмма insertion

Программа сортировки вставкой , оформленная в подпрограмму. В конце программы, вместо завершения – выход из подпрограммы инструкцией *ret*.

Результаты работы программы:

```

>>> locals
C:\Users\judge\Documents\risc\insertion.s
loop_exit2 [text] @ 0x00010094
loop_exit3 [text] @ 0x00010098
loop_exit [text] @ 0x000100a0
loop2 [text] @ 0x00010070
insertion [text] @ 0x00010050
loop1 [text] @ 0x00010054
C:\Users\judge\Documents\risc\setup.s
start [text] @ 0x00010008
finish [text] @ 0x00010010
C:\Users\judge\Documents\risc\main.s
array [data] @ 0x10000008
main [text] @ 0x0001001c
array_length [rodata] @ 0x10000000
>>> memory 0x10000008 4
      Value (+0) Value (+4) Value (+8) Value (+c)
[0x10000008] 0x0000000a 0x00000006 0x00000007 0x00000001
[0x10000018] 0x0000000c 0x00000003 0x0000000b 0x00000008
[0x10000028] 0x00000005 0x00000009 0x0000000d 0x00000002
[0x10000038] 0x00000004 0x00000000 0x00000000 0x00000000
>>>

```

Рис. 6 Значения элементов массива в начале работы программы.

$Arr = [10, 6, 7, 1, 12, 3, 11, 8, 5, 9, 13, 2, 4]$ – значение элементов в десятичном представлении в начале работы программы.

```

>>> breakpoint 0x00010010
>>> c
>>> memory 0x10000008 4
      Value (+0) Value (+4) Value (+8) Value (+c)
[0x10000008] 0x00000001 0x00000002 0x00000003 0x00000004
[0x10000018] 0x00000005 0x00000006 0x00000007 0x00000008
[0x10000028] 0x00000009 0x0000000a 0x0000000b 0x0000000c
[0x10000038] 0x0000000d 0x00000000 0x00000000 0x00000000
>>>

```

Рис. 7 Значения элементов массива в конце работы программы.

$Arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]$ - значение элементов в десятичном представлении в конце работы.

Таким образом, мы видим, что программа работает корректно.

Вывод

В ходе выполнения лабораторной работы была реализована программа на RISC-V, реализующая алгоритм сортировки вставкой. Были изучены основные аспекты RISC-V и изучена отладка программ на этом языке.