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

Отчёт по лабораторной работе № 3

Дисциплина: Низкоуровневое программирование

Вариант 6.

Выполнил студент гр. 3530901/90002 _____ М.В. Дергачев
(подпись)

Принял преподаватель _____ Д.С. Степанов
(подпись)

“_” _____ 2021 г.

Санкт-Петербург
2021

Задача

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 задания

По варианту номер 6 необходимо реализовать нахождение медианы массива in-place.

Решение

Медиана – элемент массива, который находится ровно посередине после сортировки. Для выполнения поставленной задачи программа проходится по массиву, сравнивая каждое число с каждым, прибавляя к счетчику 1, если второе число больше текущего; вычитая 1, если число меньше текущего и не делая ничего, если они равны. В конце каждого прохода проверяется значение счетчика, если он меньше 0 или больше 1 (количество чисел больше текущего меньше, чем количество чисел меньше текущего и наоборот, соответственно), то программа за место текущего числа берет следующее из массива и начинает проход заново, если значение счетчика равно 0 или 1 (0, когда в массиве нечетное количество элементов и 1, когда - четное), то медиана найдена и программа, записав ее, завершает работу.

Реализация программы.

Текст программы

```
# median search
.text
__start:
.globl __start
  li a2, 0 # i = 0
  li a3, 0 # j = 0
  li a4, 0 # count = 0

  lw a5, array_length # запись в регистр a5 длины массива

  la a6, array # запись адреса первого элемента массива в регистр a6
  la a7, array # запись адреса первого элемента массива в регистр a7

loop:
    bgeu a2, a5, finish # если i >= length выход в loop_exit
    lw t0, 0(a6) # запись в t0 значения a[i]
loop2:
      bgeu a3, a5, check # если j >= length выход в loop2_exit
      lw t1, 0(a7) # запись в t1 значения a[j]
      blt t0, t1, plusOne  # если a[i] < a[j] , то count++
      blt t1, t0, minusOne # если a[j] < a[i] , то count--
      jal zero, skip # безусловный переход в skip если a[i] == a[j]
plusOne:
      addi a4, a4, 1  # count++
      jal zero, skip
minusOne:
      addi a4, a4, -1 # count--
      jal zero, skip
skip:
      addi a7, a7, 4 # добавляем к адресу a[j] 4, для перехода к следующему элементу массива
      addi a3, a3, 1 # j += 1
      jal zero, loop2 # переход в loop2
check:
    beqz a4, finish    # если count == 0, то выходим из цикла
    li a1, 1
    beq a4, a1, finish # если count == 1, то выходим из цикла

    la a7, array # снова записываем адрес a[0] в a7
    addi a6, a6, 4 # добавляем к адресу a[i] 4, для перехода к следующему элементу массива
    li a3, 0 # j = 0
    li a4, 0 # count = 0
    addi a2, a2, 1 # i += 1
    jal zero, loop # переход в loop
finish:
  la a0, result # загружаем адрес result в a0
  lw a6, 0(a6)  # загржаем значение медианы в a6
  sw a6, 0(a0)  # записываем его в result
  li a0, 10 # x10 = 10
  ecall # ecall при значении x10 = 10 => останов симулятора

 .data   # секция изменяемых данных
result:
 .word 65535

 .rodata # секция неизменяемых данных
array_length:
 .word 13
array:
 .word 2, 0, 1, 8, 9, 4, 6, 7, 3, 10, 5, 5, 5
```

Руководство

.text - указание ассемблеру размещать последующие инструкции в секции кода.

Метка **__start:** - точка начала выполнения программы.

В строках 5-12 написаны псевдоинструкции установки значений регистров **a2-a7**. В регистре **a2** хранится счетчик i, в **a3** – счетчик j, в **a4** – счетчик count. В **a5** хранится длина массива. В регистрах **a6-a7** хранится адрес нулевого элемента массива.

В строках 14-49 записана основная часть программы.

Под меткой **loop** происходит проверка «не пройден ли весь массив», далее записывается значение текущего элемента массива в регистр **t0**.

Под меткой **loop2** выполняется та же проверка, записывается другой текущий элемент массива в регистр **t1** и сравниваются два элемента массива (**t0** и **t1**). Если **t0** < t**1**, то происходит переход в **plusOne**, если меньше, то в **minusOne**, если они равны, то в **skip**.

В **plusOne** счетчик count увеличивается на 1, в **minusOne** – уменьшается на 1, а после из обеих меток программа переходит в skip. Далее к адресу массива **a7** добавляется 4, для перехода к следующему элементу. К **a3** (j) прибавляется 1, и программа возвращается в начало второго цикла.

Когда будет пройден весь массив, программа перейдет к метке **check**, в которой проверит «была ли найдена медиана» (**count** равен 1 или 0), если проверка успешна, то программа перейдет к метке **finish**, иначе запишет в **a7** адрес нулевого элемента массива, обнулит счетчики **count** и **j**, добавит к счетчику **i** 1 и вернется в начало первого цикла.

В метке **finish** программа загрузит медиану в **result** и закончит свою работу.

Пример выполнения программы:

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100d8	00	00	00	05
0x000100d4	00	00	00	05
0x000100d0	00	00	00	05
0x000100cc	00	00	00	0a
0x000100c8	00	00	00	03
0x000100c4	00	00	00	07
0x000100c0	00	00	00	06
0x000100bc	00	00	00	04
0x000100b8	00	00	00	09
0x000100b4	00	00	00	08
0x000100b0	00	00	00	01
0x000100ac	00	00	00	00
0x000100a8	00	00	00	02

Рис. 1. Исходный массив

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100dc	00	00	00	05

Рис. 2 Результат

Таким образом, мы видим, что программа работает корректно: для массива { 2, 0, 1, 8, 9, 4, 6, 7, 3, 10, 5, 5, 5 } медиана находится правильно.

Реализация подпрограммы

Код программы и руководство

Текст программы setup

```
# setup.s
.text
__start:
.globl __start
  call main
finish:
  li a0, 10
  ecall
```

В программе вызывается подпрограмма main с помощью команды call. Эта псевдоинструкция обеспечивает безусловный переход (jump) на метку main с сохранением адреса следующей за jalr инструкции в регистре ra. Когда выполнение подпрограммы завершится, исполнение кода тестирующей программы перейдет к метке finish, в которой работа программы завершается.

Текст программы main

```
# main.s
.text
main:
.globl main
  la t2, array
  lw t3, array_length
  la t4, result

  addi sp, sp, -16
  sw ra, 12(sp)

  call median

  lw ra, 12(sp)
  addi sp, sp, 16

  li a0, 0
  ret

 .data   # секция изменяемых данных
result:
 .word 65535

 .rodata # секция неизменяемых данных
array_length:
 .word 14
array:
 .word 2, 0, 1, 8, 9, 4, 6, 7, 3, 10, 5, 7, 10, 10
```

Здесь задаются исходный массив (27-28 строки), его длина (24-25 строки) и результат (21-22). В строках 5-7 в регистры **t2-t4** записываются необходимые данные. При входе в **main** адрес возврата находится в регистре **ra**, и возврат из подпрограммы осуществляется переходом на адрес, содержащийся в этом регистре. Однако прежде, чем это произойдет, значение **ra** изменяется в результате «выполнения» псевдоинструкции **call**: в **ra** будет записан адрес возврата для вызываемой подпрограммы. Таким образом, результатом выполнения инструкции возврата, соответствующе псевдоинструкции **ret**, будет переход на эту же инструкцию – программа зациклится! Решение указанной проблемы состоит в следующем: исходное значение **ra** следует сохранить перед псевдоинструкцией **call**, и восстановить перед псевдоинструкцией **ret**. Значение регистра можно сохранить либо в другом регистре, либо в памяти.

В случае 32-разрядной версии RISC-V для сохранения значения **ra** в стеке требуется только 4 байта, однако ABI RISC-V требует выравнивания указателя стека на границу 128 разрядов (16 байт), следовательно, величина изменения указателя стека должна быть кратна 16. Кроме того, в RISC-V (как и в большинстве архитектур) стек растет вниз (grows downwards), то есть выделению памяти в стеке (stack allocation) соответствует уменьшение значения указателя стека. Отметим, что начальное значение **sp** устанавливается симулятором.

В ABI RISC-V регистр **sp** является сохраняемым, то есть при возврате из подпрограммы он должен иметь исходное значение. Поскольку для выделения памяти в стеке значение **sp** уменьшается (в данном случае на 16), перед возвратом из подпрограммы достаточно увеличить **sp** на ту же величину

Текст программы median

```
# median.s
.text
median:
.globl median
  li a2, 0 # i = 0
  li a3, 0 # j = 0
  li a4, 0 # count = 0
  # t2 = адрес a[0]
  # t3 = длина a
  # t4 = адрес result
  mv a5, t3 # запись в регистр a5 длины массива
  mv a6, t2 # запись адреса первого элемента массива в регистр a6
  mv a7, t2 # запись адреса первого элемента массива в регистр a7
loop:
    bgeu a2, a5, finish # если i >= length выход в loop_exit
    lw t0, 0(a6) # запись в t0 значения a[i]
loop2:
      bgeu a3, a5, check # если j >= length выход в loop2_exit
      lw t1, 0(a7) # запись в t1 значения a[j]
      blt t0, t1, plusOne  # если a[i] < a[j] , то count++
      blt t1, t0, minusOne # если a[j] < a[i] , то count--
      jal zero, skip # безусловный переход в skip если a[i] == a[j]
plusOne:
      addi a4, a4, 1  # count++
      jal zero, skip
minusOne:
      addi a4, a4, -1 # count--
      jal zero, skip
skip:
      addi a7, a7, 4 # добавляем к адресу a[j] 4, для перехода к следующему элементу массива
      addi a3, a3, 1 # j += 1
      jal zero, loop2 # переход в loop2
check:
   beqz a4, finish    # если count == 0, то выходим из цикла
   li a1, 1
   beq a4, a1, finish # если count == 1, то выходим из цикла

   mv a7, t2 # снова записываем адрес a[0] в a7
   addi a6, a6, 4 # добавляем к адресу a[i] 4, для перехода к следующему элементу массива
   li a3, 0 # j = 0
   li a4, 0 # count = 0
   addi a2, a2, 1 # i += 1
   jal zero, loop # переход в loop
finish:
  mv a0, t4 # загружаем адрес result в a0
  lw a6, 0(a6)  # загржаем значение медианы в a6
  sw a6, 0(a0)  # записываем его в result
  ret
```

Копия программы из первого пункта работы, за исключением того, что значение в регистрах устанавливается до вызова подпрограммы, а вместо завершения в конце находится возврат из подпрограммы **ret**

Пример выполнения программы:

Registers | Memory | Cache

Address	+3	+2	+1	+0
0x0001010c	00	00	00	0a
0x00010108	00	00	00	0a
0x00010104	00	00	00	07
0x00010100	00	00	00	05
0x000100fc	00	00	00	0a
0x000100f8	00	00	00	03
0x000100f4	00	00	00	07
0x000100f0	00	00	00	06
0x000100ec	00	00	00	04
0x000100e8	00	00	00	09
0x000100e4	00	00	00	08
0x000100e0	00	00	00	01
0x000100dc	00	00	00	00
0x000100d8	00	00	00	02

Рис. 3. Исходный массив

Registers | Memory | Cache

Address	+3	+2	+1	+0
0x00010110	00	00	00	06

Рис. 4. Результат

Таким образом, мы видим, что программа работает корректно: для массива { 2, 0, 1, 8, 9, 4, 6, 7, 3, 10, 5, 7, 10, 10} медиана находится правильно.

Вывод

В ходе выполнения данной лабораторной работы была реализована программа на RISC-V, реализующая нахождение медианы массива. Также она была представлена в виде подпрограммы.

Список использованных источников

http://kspt.icc.spbstu.ru/media/files/2020/lowlevelprog/riscv_prgc.pdf

http://kspt.icc.spbstu.ru/media/files/2020/lowlevelprog/riscv_subprgc.pdf

https://github.com/riscv/riscv-elf-psabi-doc/blob/master/riscv-elf.md

https://habr.com/ru/post/533272/