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

Тема: программирование RISC-V

Вариант: 17

Выполнил студент гр. 3530901/90002 _____А.И. Юрченко
(подпись)

Принял старший преподаватель _____Д. С. Степанов
(подпись)

“___” _____ 2021 г.

Санкт-Петербург
2021

Формулировка задачи

1. Разработать программу на языке ассемблера RISC-V, реализующую определенную вариантом задания функциональность, отладить программу в симуляторе VSim/Jupiter. Массив 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 17 необходимо реализовать реверс массива. Алгоритм состоит из одного прохода по массиву. За этот проход элемент с индексом [0] меняется местами с элементом с индексом [n-1], элемент с индексом [1] меняется местами с элементом с индексом [n-2] и так далее. Где n – длина массива.

Первая часть задания

Должна быть написана программа, осуществляющая реверс массива. Код представлен в приложении 1. Приведу описание работы кода.

Для данной части задания начальной меткой (Start label) выбрана start. Массив данных расположен в секции data, а числа массива представлены смежными 4-байтными словами (word). Так же в секции read only data расположен размер массива в word.

```
lw a3, array_length
la a4, array

mv t0, a4
slli t1, a3, 2
add t1, t0, t1
addi t1, t1, -4
li t2, 1

bgeu t2, a3, finish
```

В первой строке мы записываем в регистр a3 длину массива. Во второй – в регистр a4 адрес ячейки, где находится нулевой элемент массива. В следующей копируем значение из регистра a4 в t0 (в нём будут храниться адреса элементов слева, которые будут меняться с элементами справа).

Далее запишем в регистр t1 адрес последнего элемента в массиве. Чтобы получить адрес последнего элемента надо к начальному адресу прибавить длину массива, умноженную на 4, и вычесть 4. В регистр t1 записываем длину массива из a3, сдвинутую на 2 разряда влево (умножение на 4) – это сделано из-за того, что числа хранятся в word, значит для их записи нужно 4 байта. Далее к t1 прибавляем t0 и вычитаем 4. Таким образом, $t1 = a3 * 4 + t0 - 4$.

В регистр t2 записываем 1 для того, чтобы проверять, поменяли ли мы все элементы массива местами.

Последняя строка проверяет перед входом в цикл, больше ли элементов в массиве, чем 1. Иначе программа завершается.

```
loop:
  lw t3, 0(t0)
  lw t4, 0(t1)
  sw t3, 0(t1)
  sw t4, 0(t0)
  addi t0, t0, 4
  addi t1, t1, -4
  addi t2, t2, 2
  bltu t2, a3, loop
```

Первые две строчки в цикле — это запись в регистры t3 и t4 адреса элементов a[i] и a[array_length-1-i]. Следующие две строчки перезаписывают массив, то есть меняют местами значения a[i] и a[array_length-1-i]. Далее мы увеличиваем адрес левого элемента массива на 4, уменьшаем адрес правого на 4 и увеличиваем счётчик в регистре t2 на 2. Последняя строка – проверка, если t2 < a3, то возвращаемся в начало цикла.

```
finish:
  li a0, 10
  li a1, 0
  ecall
```

В конце программы, на метке “finish” выходим из программы при помощи ecall.

Тестирование программы

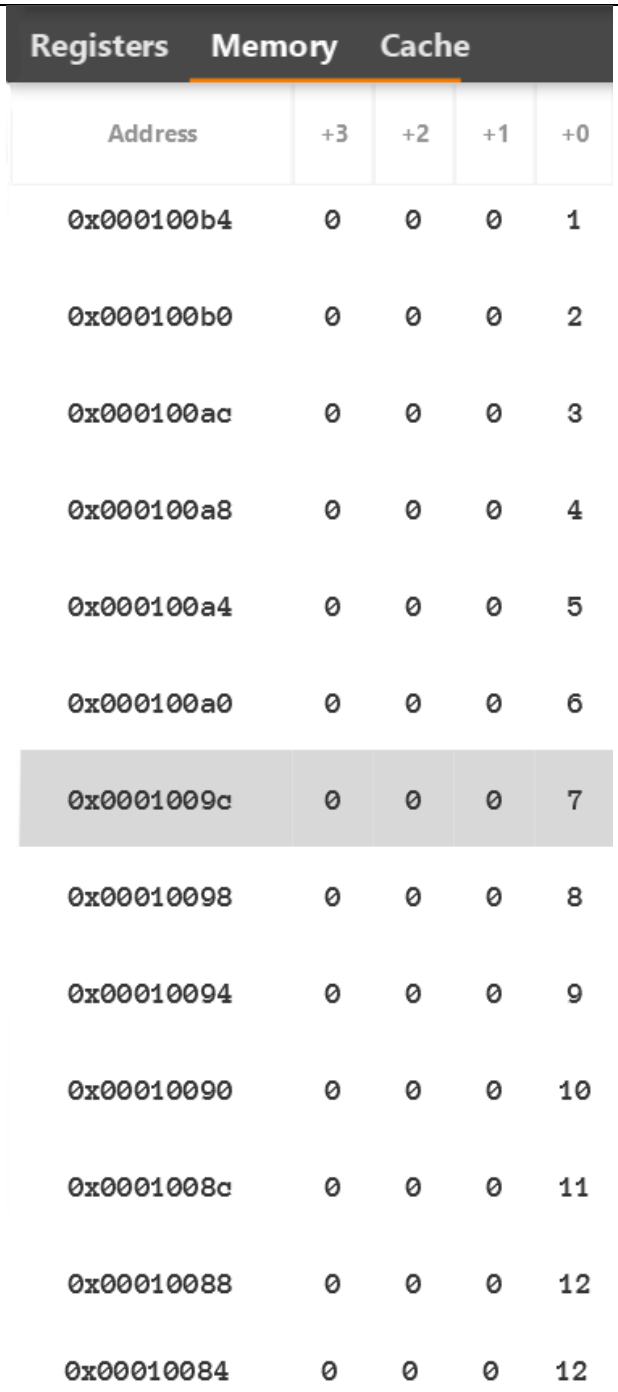
Тестирование написанной программы выполнялось при помощи симулятора Jupiter с графической оболочкой.

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100b4	0	0	0	12
0x000100b0	0	0	0	11
0x000100ac	0	0	0	10
0x000100a8	0	0	0	9
0x000100a4	0	0	0	8
0x000100a0	0	0	0	7
0x0001009c	0	0	0	6
0x00010098	0	0	0	5
0x00010094	0	0	0	4
0x00010090	0	0	0	3
0x0001008c	0	0	0	2
0x00010088	0	0	0	1
0x00010084	0	0	0	12

Рис. 1 Исходный массив.

Видно, что длина массива записана в 0x00010084, а в следующих адресах записаны наши четырёхбайтные слова в числе в младшем байте.



Registers | Memory | Cache

Address	+3	+2	+1	+0
0x000100b4	0	0	0	1
0x000100b0	0	0	0	2
0x000100ac	0	0	0	3
0x000100a8	0	0	0	4
0x000100a4	0	0	0	5
0x000100a0	0	0	0	6
0x0001009c	0	0	0	7
0x00010098	0	0	0	8
0x00010094	0	0	0	9
0x00010090	0	0	0	10
0x0001008c	0	0	0	11
0x00010088	0	0	0	12
0x00010084	0	0	0	12

Рис. 2 Массив после выполнения программы.

Реверс массива выполнен и был совершён выход из программы.

Выделение реверса в подпрограмму

Согласно заданию, была написана программа, листинг которой находится в приложении 2. Была составлена основная программа setup, которая вызывает тестовую программу main, а та в свою очередь вызывает revers. Так как при исполнении псевдоинструкции call в регистр ra записывается адрес возврата для инструкции ret, а мы вызываем call 2 раза и возврат в setup не выполняется.

Решение этой 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.

Адрес 0-ого элемента массива и длина передаются через регистры a0, a1. Сами значения записываются в тестовой программе. Программа организована в соответствии с ABI.

Тестирование программы

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100cc	0	0	0	12
0x000100c8	0	0	0	11
0x000100c4	0	0	0	10
0x000100c0	0	0	0	9
0x000100bc	0	0	0	8
0x000100b8	0	0	0	7
0x000100b4	0	0	0	6
0x000100b0	0	0	0	5
0x000100ac	0	0	0	4
0x000100a8	0	0	0	3
0x000100a4	0	0	0	2
0x000100a0	0	0	0	1

Рис. 3 Первый исходный массив.

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100cc	0	0	0	1
0x000100c8	0	0	0	2
0x000100c4	0	0	0	3
0x000100c0	0	0	0	4
0x000100bc	0	0	0	5
0x000100b8	0	0	0	6
0x000100b4	0	0	0	7
0x000100b0	0	0	0	8
0x000100ac	0	0	0	9
0x000100a8	0	0	0	10
0x000100a4	0	0	0	11
0x000100a0	0	0	0	12

Рис. 4 Первый массив после выполнения программы.

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100e0	0	0	0	90
0x000100dc	0	0	0	0
0x000100d8	0	0	0	10
0x000100d4	0	0	0	5
0x000100d0	0	0	0	65

Рис. 5 Второй исходный массив.

Registers Memory Cache

Address	+3	+2	+1	+0
0x000100e0	0	0	0	65
0x000100dc	0	0	0	5
0x000100d8	0	0	0	10
0x000100d4	0	0	0	0
0x000100d0	0	0	0	90

Рис. 6 Второй массив после выполнения программы.

Тестирование программы

В ходе выполнения данной лабораторной работы был получен опыт программирования на RISC-V. Была написана программа реверса массива. Так же данная программа была выделена как подпрограмма и вызвана несколько раз в тестовой программе. Вызов подпрограммы выполняется согласно соглашениям ABI. Весь исходный код предоставлен в репозитории на GitHub.

Приложение 1

```
.text
start:
.globl start
  lw a3, array_length # a3 = <длина массива>
  la a4, array # a4 = <адрес 0-го элемента массива>

  mv t0, a4 # t0 = a4
  slli t1, a3, 2  # t1 = a3 << 2 = a3 * 4
  add t1, t0, t1  # t1 = t0 + t1 = t0 + a3 * 4
  addi t1, t1, -4  # t1 = t1 - 4
  li t2, 1 # t2 = 1

  bgeu t2, a3, finish # if( t2 >= a3 ) goto finish
loop:
  lw t3, 0(t0) # \
  lw t4, 0(t1) #  | swapping
  sw t3, 0(t1) #  |
  sw t4, 0(t0) # /
  addi t0, t0, 4  # t0 += 4
  addi t1, t1, -4  # t1 -= 4
  addi t2, t2, 2  # t2 += 2
  bltu t2, a3, loop # if( t2 < a3 ) goto loop

finish:
  li a0, 10 # x10 = 10
  li a1, 0
  ecall # ecall при значении x10 = 10 => останов симулятора

.rodata
array_length:
 .word 12
.data
array:
 .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

Приложение 2

Main.s:

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

  la a0, array1  # адрес 0-го элемента первого массива
  lw a1, array_length1  # длина второго массива
  call revers

  la a0, array2  # адрес 0-го элемента второго массива
  lw a1, array_length2  # длина массива второго массива
  call revers

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

.rodata
array_length1:
 .word 12
array_length2:
 .word 5

.data
array1:
 .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
array2:
 .word 65, 5, 10, 0, 90
```

Setup.s:

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

finish:
  li a0, 10
  ecall
```

Revers.s:

```
# revers.s
.text
revers:
.globl revers
  # в a0 - адрес 0-го элемента массива чисел типа unsigned
  # в a1 - длина массива
  mv t0, a0

  slli t1, a1, 2  # t1 = a1 << 2 = a1 * 4
  add t1, a0, t1  # t1 = a0 + t1 = a0 + a3 * 4
  addi t1, t1, -4  # t1 = t1 - 4
  li t2, 1 # t2 = 1

  bgeu t2, a1, finish # if( t2 >= a1 ) goto loop_exit
loop:
  lw t3, 0(t0) # \
  lw t4, 0(t1) #  | swapping
  sw t3, 0(t1) #  |
  sw t4, 0(t0) # /
  addi t0, t0, 4  # t0 += 4
  addi t1, t1, -4  # t1 -= 4
  addi t2, t2, 2  # t2 += 2
  bltu t2, a1, loop # if( t2 < a1 ) goto loop
finish:

  ret
```