Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа информационных систем и суперкомпьютерных технологий

Низкоуровневое программирование

Отчёт по лабораторной работе №3
Программирование RISC-V

Работу выполнил:
Унтила А.А.
Группа:
3530901/90004
Преподаватель:
Алексюк А.О.

Санкт-Петербург
2021

Содержание

1. Цель работы

1. Разработать программу на языке ассемблера RISC-V, реализующую Реверс массива чисел, 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.

2. Программа работы

- Постановка задачи.
- Симулятор RISC-V.
- Программа, реализующая реверс массива.
- Тестовая программа и подпрограмма.

3. Теоретическая информация

RISC (англ. ***R**educed **I**nstruction **S**et **C**omputer* - «компьютер с набором коротких/-простых/быстрых команд») — архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим. Первые RISC-процессоры даже не имели инструкций умножения и деления. Это также облегчает повышение тактовой частоты и делает более эффективной суперскалярность (распараллеливание инструкций между несколькими исполнительными блоками).

RISC-V - открытая и свободная система команд и процессорная архитектура на основе концепции RISC для микропроцессоров и микроконтроллеров. Создана в 2010 году исследователями из отделения информатики Калифорнийского университета в Беркли. Первая версия процессора «RISC-I» появилась на свет в 1982 году и имела всего 32 инструкции и около 40 тыс. транзисторов (меньше, чем у аналогов), однако превосходила по скорости работы любой одночиповый процессор того времени.

4. Ход выполнения работы

4.1. Постановка задачи

Реверс массива чисел - запись разрядов в обратном порядке. Для отображения корректности работы программы используется массив с 11-ю элементами 0, 1, ..., 10.

Алгоритм:

- Запись в регистр адреса зеркального элемента массива + 1.
- Запись в регистры значений текущего и зеркального элементов. Значения из ячеек памяти с известными адресами, адрес зеркальной ячейки - адрес, полученный на первом шаге, минус 4 (адрес предыдущего элемента).
- Запись значения текущей ячейки в зеркальную и наоборот.
- Повторение всех предыдущих шагов пока не останется необработанных элеметов.

В качестве рабочей переменной цикла используется регистр a2, куда изначально записывается длина массива. На каждой итерации цикла из данного регистра вычитается 2 - кол-во элементов, обработанных за одну итерацию. Для записи и чтения элементов массива достаточно знать номер ячейки с первым (или нулевым) элементом массива. Адрес этой ячейки хранится в регистре a3 и отвечает за адрес текущего элемента, на каждой итерации цикла увеличиваясь на 4 (адрес следующей ячейки).

4.2. Выбор симулятора RISC-V

В данной лабораторной работе используется симулятор RISC-V Jupiter, расположенный по ссылке https://github.com/andrescv/Jupiter/releases. Симулятор представляет собой простую среду разработки, имеет текстовый редактор, позволяет выполнять программу в режиме пошаговой отладки и отслеживать состояние регистров и памяти.

4.3. Программа, реализующая реверс массива

Листинг программы с поясняющими комментариями:

```
# Реверс массива
.text
start:
.globl start
 lw a2, array_length # a2 - длина массива
 la a3, array # a4 - адрес начала массива

loop:
 # branch less-then zero
 bltz a2, loop_exit # if ( a2 < 0 ) goto loop_exit

 slli a4, a2, 2 # a4 = a2 << 2 = a2 * 4
 add a4, a3, a4 # a4 = a3 + a4 = a3 + a2 * 4

 lw t0, 0(a3) # t0 = array[i]
 lw t1, -4(a4) # t1 = array[length - i]
 sw t1, 0(a3) # array[i] = array[length - i]
 sw t0, -4(a4) # array[length - i] = array[i]

 addi a3, a3, 4 # i += 1
 addi a2, a2, -2 # a2 -= 2

 # jump and link
 jal zero, loop # goto loop

loop_exit: # выход из цикла
 li a0, 10 # x10 = 10
 ecall # ecall при значении x10 = 10 => останов симулятора

.rodata
array_length: # длина массива
 .word 11
.data
array: # элементы массива
 .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Address	+3	+2	+1	+0
0x00010074	00	00	00	0a
0x00010070	00	00	00	09
0x0001006c	00	00	00	08
0x00010068	00	00	00	07
0x00010064	00	00	00	06
0x00010060	00	00	00	05
0x0001005c	00	00	00	04
0x00010058	00	00	00	03
0x00010054	00	00	00	02
0x00010050	00	00	00	01
0x0001004c	00	00	00	00
0x00010048	00	00	00	0b

Address	+3	+2	+1	+0
0x00010074	00	00	00	00
0x00010070	00	00	00	01
0x0001006c	00	00	00	02
0x00010068	00	00	00	03
0x00010064	00	00	00	04
0x00010060	00	00	00	05
0x0001005c	00	00	00	06
0x00010058	00	00	00	07
0x00010054	00	00	00	08
0x00010050	00	00	00	09
0x0001004c	00	00	00	0a
0x00010048	00	00	00	0b

Рис. 4.3.1. Массив до и после реверса.

4.4. Тестовая программа и подпрограмма

Листинг программы с поясняющими комментариями:

```
# Тестовая программа и подпрограмма
# setup.s
.text
start:
.globl start
 call main # вызов фции- main
finish:
 mv a1, a0 # a1 = a0
 li a0, 17 # a0 = 17
 ecall # выход с кодом завершения

# main.s
.text
main:
.globl main
 addi sp, sp, -16 # выделение памяти в стеке
 sw ra, 12(sp) # сохранение ra

 lw a0, array_length # a0 - длина массива
 la a1, array # a1 - адрес начала массива
 call reverse_array # вызов фции- реверса

 lw ra, 12(sp) # восстановление ra
 addi sp, sp, 16 # освобождение памяти в стеке
 li a0, 0
 ret # возврат

 .rodata
array_length: # длина массива
 .word 11
.data
array: # элементы массива
 .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

# reverse_array
.text
reverse_array:
.globl reverse_array
loop:
 # branch less-then zero
 bltz a0, loop_exit # if ( a2 < 0 ) goto loop_exit

 slli a7, a0, 2 # a7 = a0 << 2 = a0 * 4
 add a7, a1, a7 # a7 = a1 + a7 = a1 + a0 * 4

 lw t0, 0(a1) # t0 = array[i]
 lw t1, -4(a7) # t1 = array[length - i]
```

```
48 sw t1, 0(a1) # array[i] = array[length - i]
49 sw t0, -4(a7) # array[length - i] = array[i]
50
51 addi a1, a1, 4 # i + 1
52 addi a0, a0, -2 # a2 -= 1
53
54 # jump and link
55 jal zero, loop # goto loop
56
57 loop_exit: # выход из цикла
58 ret # возврат
```

Address	+3	+2	+1	+0
0x000100a4	00	00	00	0a
0x000100a0	00	00	00	09
0x0001009c	00	00	00	08
0x00010098	00	00	00	07
0x00010094	00	00	00	06
0x00010090	00	00	00	05
0x0001008c	00	00	00	04
0x00010088	00	00	00	03
0x00010084	00	00	00	02
0x00010080	00	00	00	01
0x0001007c	00	00	00	00
0x00010078	00	00	00	0b

Address	+3	+2	+1	+0
0x000100a4	00	00	00	00
0x000100a0	00	00	00	01
0x0001009c	00	00	00	02
0x00010098	00	00	00	03
0x00010094	00	00	00	04
0x00010090	00	00	00	05
0x0001008c	00	00	00	06
0x00010088	00	00	00	07
0x00010084	00	00	00	08
0x00010080	00	00	00	09
0x0001007c	00	00	00	0a
0x00010078	00	00	00	0b

Рис. 4.4.1. Массив до и после реверса.

5. Вывод

В ходе лабораторной работы написано две программы на языке ассемблера RISC-V, осуществляющие реверс массива чисел (запись разрядов в обратном порядке), описан общий алгоритм реализации данной функциональности. Первая программа представляет собой непосредственную реализацию заданной функциональности, вторая программа с использованием тестовой программы и замыкания заданной функциональности в подпрограмму использует больше возможностей RISC-V. Работа обоих программ протестирована в симуляторе Jupiter, приведены соответствующие результаты, обе программы работают корректно.