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

Отчёт по лабораторной работе№ 3

Дисциплина: Низкоуровневое программирование

Тема: Программирование RISC-V

Выполнил студент гр. 3530901/10005 Кузичева П. Д.

Преподаватель Коренев Д. А.

“_” _____ 2022 г.

Санкт-Петербург

Оглавление

Техническое задание

Написать программу на языке ассемблера RISC-V, которая реализует реверс исходного массива. Отладить ее в симуляторе Jupiter. Необходимо также выделить основной функционал в подпрограмму и написать для нее тестовую программу.

Метод решения

Реверс массива осуществляется посредством перестановки элементов. При реверсе первый элемент массива становится последним, а последний – первым, второй – предпоследним, а предпоследний – вторым и так далее. В случае, когда количество элементом массива нечетное число, средний элемент, то есть стоящий на середине массива, остается на своем месте. Перестановок надо сделать в два раза меньше, чем длина массива.

Например, рассмотрим массив [5, 8, 13, 3, 59]. Сначала будут переставлены элементы 5 и 59. После этой итерации получиться массив [59, 8, 13, 3, 5]. После будут переставлены на другие места элементы 8 и 3, и получиться массив [59, 3, 13, 8, 5]. Элемент 13 игнорируется, так как не имеет пары и находиться на середине массива.

Руководство программисту

Начальные данные к программе: адрес нулевого элемента массива (и соответственно сам массив) и его длина. В реализации без подпрограммы адрес и длина хранятся в регистрах a4 и a3 соответственно. В реализации через подпрограмму адреса остаются прежними.

Реализация программы

```
.text
__start:
.globl __start

    la a3, array_length
    lw a3, 0(a3) # загрузка длины массива
    la a4, array # адрес нулевого элемента
    slli a5, a3, 2 # адрес последнего элемента будет хранится в ячейке a5(a5 = a3*4)
    add a5, a4, a5 # a5 = a4 + a3 * 4
    addi a5, a5, -4 # коррекция адреса
    # a5 - адрес последнего элемента
    addi a2, a2, 1 # const = 1

loop:
    lw t1, 0(a4) # меняем местами элементы
    lw t0, 0(a5)
    sw t1, 0(a5)
    sw t0, 0(a4)
    addi a5, a5, -4 # переходим на следующий элемент в конце
    addi a4, a4, 4 # переходим на следующий элемент в начале
    addi a3, a3, -2 # отнимаем два от длины массива
    bgeu a2, a3, loop_exit #if( a2(1) >= a3(длина массива) goto loop_exit
    jal zero, loop # goto loop
loop_exit:

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

.rodata # неизменяемые данные
  array_length: # длина массива
    .word 5

.data # изменяемые  данные
  array: # массив
    .word 1, 2, 3, 4, 5
```

Результат работы программы

Входные данные – массив [1, 2, 3, 4, 5]. Ожидаемый результат - [5, 4, 3, 2, 1].

```
0x00010070      00    00    00    05

0x0001006c      00    00    00    04

0x00010068      00    00    00    03

0x00010064      00    00    00    02

0x00010060      00    00    00    01
```

Исходные данные

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

После запуска программы

Результат совпал с ожидаемым.

Реализация подпрограммы и тестовой программы

Стартовый файл программы

setup.s

```
1  .text
2  __start:
3  .globl __start
4
5      call main
6
7  finish:
8
9      mv a1, a0 # a1 = a0
10     li a0, 17
11     ecall # выход с кодом завершения
```

Тестовая программа

main.s

```
.text
main:
.globl main

    lw a1, array_length # загрузка длины массива
    la a0, array # адрес нулевого элемента

    addi sp, sp, -16 # выделение памяти в стеке
    sw ra, 12(sp) # сохранение ra

    call reverse # вызов подпрограммы

    lw ra, 12(sp) # восстановление ra
    addi sp, sp, 16 # освобождение памяти в стеке

    li a0, 0 # a0 = 0
    ret # return 0

.rodata # неизменяемые данные
  array_length: # длина массива
    .word 5

.data # изменяемые  данные
  array: # массив
    .word 1, 2, 3, 4, 5
```

Подпрограмма

reverse.s

```
.text
reverse:
.globl reverse
    # подпрограмма принимает два значения: адрес нулевого элемента и длину массива
    # в a0 - адрес 0-го элемента массива чисел типа unsigned
    # в a1 - длина массива
    slli a5, a1, 2 # адрес последнего элемента будет хранится в ячейке a5(a5 = a3*4)
    add a5, a0, a5 # a5 = a4 + a3 * 4
    addi a5, a5, -4 # коррекция адреса
    # a5 - адрес последнего элемента
    addi a2, a2, 1 # const = 1

loop:
    lw t1, 0(a0) # меняем местами элементы
    lw t0, 0(a5)
    sw t1, 0(a5)
    sw t0, 0(a0)
    addi a5, a5, -4 # переходим на следующий элемент в конце
    addi a0, a0, 4 # переходим на следующий элемент в начале
    addi a1, a1, -2 # отнимаем два от длины массива
    bgeu a2, a1, loop_exit #if( a2(1) >= a3(длина массива) goto loop_exit
    jal zero, loop # goto loop
loop_exit:
finish:
    ret
```

Результат работы подпрограммы и тестовой программы

Входные данные – массив [1, 2, 3, 4, 5]. Ожидаемый результат - [5, 4, 3, 2, 1].

```
0x00010078      00    00    00    05

0x00010074      00    00    00    04

0x00010070      00    00    00    03

0x0001006c      00    00    00    02

0x00010068      00    00    00    01
```

Исходные данные

0x00010078	00	00	00	01
0x00010074	00	00	00	02
0x00010070	00	00	00	03
0x0001006c	00	00	00	04
0x00010068	00	00	00	05

После запуска программы

Результат совпал с ожидаемым.

Вывод

В ходе выполнения лабораторной работы была разработана программа на языке ассемблера RISC-V, реализующая реверс массива. Программа была отлажена в симуляторе Jupiter. Результаты работы программы совпали с ожидаемыми.