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 для RISC-V

Вариант: 3

Выполнила студентка гр. 3530901/00002_____С.Е. Бельская
(подпись)

Принял преподаватель _____Д.С. Степанов
(подпись)

«___»_____2021 г.

Санкт-Петербург

2021

Задачи

1. Разработать программу на языке ассемблера RISC-V, реализующую сортировку методом выбора массива чисел, отладить программу в симуляторе Jupiter. Массив данных и длина массива располагаются в памяти по фиксированным адресам.
2. Выделить сортировку в подпрограмму, организованную в соответствии с ABI, разработать использующую её тестовую программу. Адрес обрабатываемого массива и длину массива передавать через параметры подпрограммы в соответствии с ABI. Тестовая программа должна состоять из инициализирующего кода, кода завершения, подпрограммы main и тестируемой подпрограммы.

Алгоритм сортировки на языке C++

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main ()
{
	int length;
	cin >> length;
	int array[length];
	for (int i = 0; i < length - 1; i ++)
		cin >> array[i];
	int temp, min;
	for (int out; out < length - 2; out ++)
	{
		min = out;
		for (int in = out + 1; in <length - 1; in ++)
			if (array[in] < array[min]) min = in;
		temp = array[out];
		array[out] = array[min];
		array[min] = temp;
	};
	for (int i = 0; i < length - 1; i++)
		cout << array[i] << ' ';
	return 0;
}
```

Реализация программы

Код на языке ассемблера RISC-V

```
.text
__start:
.globl __start
lw a2, array_lenght #Длина массива
li a3, 0           #i
li a4, 0            #j
la a5, array        #addr1 min
la a6, array        #addr2 least
la a7, array

loop_i:
bgeu a3, a2, loops_exit #if (a3>=a2) exit if (i>= arr_len)
addi a6, a5, 0 #a6 = a5 + 0
addi a3, a3, 1 #a3 = a3 + 1
addi a4, a3, 0 #a4 = a3 + 0
loop_j:  #j= min + 1
bgeu a4, a2, loop_j_exit #if (a4>=a2) exit
slli t1, a4, 2 #t1 = a4 << 2 = a4*4
addi a4, a4, 1 #a4 = a4 + 1
add t1, a7, t1 #t1 = a7 + t1
lw t2, 0(t1) #t2 = arr[j]
lw t3, 0(a6) #t3 = arr[least]
bgtu t2, t3, loop_j #arr[j] > arr[least]
addi a6, t1, 0
j loop_j

loop_j_exit:
lw t4, 0(a5)
lw t5, 0(a6)
sw t4, 0(a6) #least=min
sw t5, 0(a5) #min =least
addi a5, a5, 4
j loop_i

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

.rodata
array_lenght:
  .word 4

.data
array:
  .word 4, 3, 2, 1
```

Рис. 1 Код основной программы.

Проверим работоспособность алгоритма

```
0x00010094    00   00   00   01
0x00010090    00   00   00   02
0x0001008c    00   00   00   03
0x00010088    00   00   00   04
0x00010084    00   00   00   04
```

Рис. 2 Исходные данные

После запуска алгоритма получим:

```
0x00010094    00   00   00   04
0x00010090    00   00   00   03
0x0001008c    00   00   00   02
0x00010088    00   00   00   01
0x00010084    00   00   00   04
```

Рис. 3 Результат работы программы

Как видно, программа работает корректно.

Реализация подпрограммы

```
.text
__start:
.globl __start
  call sortMain
finish:
  mv a1, a0 # a1 = a0
  li a0, 17 # a0 = 17
  ecall # выход с кодом завершения
```

Рис. 4 Тестирующая программа

```
.text
sortSub:
.globl sortSub
loop_i:
bgeu a3, a2, loops_exit #if ((a3>=a2) exit if (i>= arr_len)
addi a6, a5, 0
addi a3, a3, 1 #min++
addi a4, a3, 0
loop_j:  #j= min + 1
bgeu a4, a2, loop_j_exit
slli t1, a4, 2
addi a4, a4, 1
add t1, a7, t1
lw t2, 0(t1) #t2 = arr[j]
lw t3, 0(a6) #t3 = arr[least]
bgtu t2, t3, loop_j #arr[j] > arr[least]
addi a6, t1, 0
j loop_j

loop_j_exit:
lw t4, 0(a5)
lw t5, 0(a6)
sw t4, 0(a6) #least=min
sw t5, 0(a5) #min =least
addi a5, a5, 4
j loop_i

loops_exit:
 ret
```

Рис. 5 Подпрограмма sortSub

Подпрограмма из предыдущего пункта. Завершается она не завершением работы программы, а выходом из подпрограммы (ret)

```
.text
sortMain:
.globl sortMain
lw a2, array_lenght #Длина массива
li a3, 0            #i
li a4, 0            #j
la a5, array        #addr1 min
la a6, array        #addr2 least
la a7, array

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

  call sortSub  # call fun

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

  ret        # } return 0;
.rodata
array_lenght:
  .word 4

.data
array:
  .word 4, 3, 2, 1
```

Рис. 6 Подпрограмма sortMain

В подпрограмме sortMain задается массив и его длина. Затем вызывается подпрограмма sortSub.

Вызов подпрограммы sortSub «обернут» выделением памяти в стеке, так как нам нужно в регистр ra сохранить значение.

При отсутствии этой «обёртки» регистр ra будет перезаписываться, и мы теряем откуда изначально пришли.

В случае 32-разрядной версии RISC-V для сохранения значения rав 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) стекрастет 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 привозврате 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

Проверим работоспособность

```
0x000100c0    00   00   00   01
0x000100bc    00   00   00   02
0x000100b8    00   00   00   03
0x000100b4    00   00   00   04
0x000100b0    00   00   00   04
```

Рис. 7 Исходные данные

После запуска программы получим

```
0x000100c0    00   00   00   04
0x000100bc    00   00   00   03
0x000100b8    00   00   00   02
0x000100b4    00   00   00   01
0x000100b0    00   00   00   04
```

Рис. 8 Результаты тестирования

Как видно, программа работает корректно

Вывод

В ходе данной работы был реализован алгоритм сортировки массива методом выбора на языке ассемблера RISC-V. Была написана как сама программа, так и её представление в виде подпрограмм. Результаты совпали с ожидаемыми.

В сравнении с инструкциями в EDSAC, инструкции в RISC-V более компактны и читабельны.