Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Низкоуровневое программирование

Отчет по лабораторной работе №1
RISC-V

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

Санкт-Петербург
2021

Содержание

1. Цель работы

- Разработать программу на языке ассемблера RISC-V, реализующую сортировку массива выбором. Массив данных и другие параметры располагаются в памяти по фиксированным адресам.
- Выделить определенную вариантом задания функциональность в подпрограмму, организованную в соответствии с ABI, разработать использующую ее тестовую программу.

2. Программа работы

- Установить симулятор VSim/Jupiter
- Изучить и разработаться с RISC-V
- Разработать требуемые программы в соответствии с заданным вариантом

3. Теория

RISC (англ. Restricted (reduced) Instruction Set Computer — «компьютер с сокращённым набором команд») — архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим. RISC-V — открытая и свободная система команд и процессорная архитектура на основе концепции RISC. RISC-V имеет 32 (или 16 для встраиваемых применений) целочисленных регистра. При реализации вещественных групп команд есть дополнительно 32 вещественных регистра. В ассемблере RISC-V для регистров x0-x31 определены синонимы в соответствии с соглашениями, установленными ABI (application binary interface), так, синонимом “x10” является “a0”.

Регистр	Имя в ABI	Описание	Тип
32 целочисленных регистра			
x0	zero	Hard-wired zero	—
x1	ra	Return address	Вызывающий
x2	sp	Stack pointer	Вызывающий
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5	t0	Temporary/alternate link register	Вызывающий
x6-7	t1-2	Temporaries	Вызывающий
x8	s0/fp	Saved register/frame pointer	Вызывающий
x9	s1	Saved register	Вызывающий
x10-11	a0-1	Function arguments/return values	Вызывающий
x12-17	a2-7	Function arguments	Вызывающий
x18-27	s2-11	Saved registers	Вызывающий
x28-31	t3-6	Temporaries	Вызывающий
32 вещественных регистра (опционально)			
f0-7	ft0-7	FP temporaries	Вызываемый
f8-9	fs0-1	FP saved registers	Вызываемый
f10-11	fa0-1	FP arguments/return values	Вызываемый
f12-17	fa2-7	FP arguments	Вызываемый
f18-27	fs2-11	FP saved registers	Вызываемый
f28-31	ft8-11	FP temporaries	Вызываемый

Рисунок 3.1. Регистры

Программа может содержать в себе метки, псевдоинструкции (и инструкции соответственно), комментарии и директивы.

- Комментарии открываются символом “#” и продолжаются до конца строки.
- Метки обозначают определенные «точки» программы, их использование позволяет поручить подсчет адресов ассемблеру.
- Псевдоинструкции транслируются ассемблером в последовательность инструкций. Это упрощает процесс написания.
- Инструкции - системы команд RISC-V, обеспечивающую выполнение требуемого действия. Инструкции базового набора имеют длину 32 бита.
- Директивы указывают ассемблеру размещать последующие инструкции в секции кода (Например “text” или "data")

Директива .data указывает ассемблеру размещать последующие слова в секции (изменяемых) данных.

Директива .word указывает ассемблеру сформировать последовательность 32-разрядных значений (машинных слов), соответствующих указанным константам, разбить их на байты и разместить полученные значения в текущей секции.

Сразу отметим, что поскольку каждый элемент массива занимает 4 байта памяти, итерация по адресам может происходить следующим образом: <адрес элемента i+1> = <адрес элемента i> + 4

4. Ход выполнения работы

Общая идея заключается в том, что мы меняем максимальный элемент с последним необработанным, последовательно уменьшая количество необработанных элементов. Для этого внешний цикл будет сокращать границу необработанной части массива, а внутренний искать максимальный элемент среди необработанных. Листинг программы:

```
1  .text
2 start:
3 .globl start
4
5  lw a3, array_length # a3 = длина< массива>
6  la a4, array # a4 = адрес< го0- элемента массива>
7  slli a6, a3, 2 # a6 = a3*4
8  add a6, a4, a6 # a6 = a4+a3*4
9  addi a6, a6, -4        \#last el
10
11 loop:
12  bgeu zero, a3, loop_exit # if( 0 >= a3 ) goto loop_exit
13
14  li a2, 1 \# a2 = 1
15  addi a7, a4, 0  #max place
16  lw t2, 0(a7)  #max = 1 element
17  add a5, a4, zero   #a5 = first element
18
19  loop2:
20  bgeu a2, a3, loop2_exit # if( a2 >= a3 ) goto loop_exit
21  addi a5, a5, 4 # a5 = a5 +  4
22  addi a2, a2, 1 # a2 += 1
23  lw t3, 0(a5)
24  bgeu t2, t3, loop2 # if( t2 >= t3 ) goto loop_exit
25  addi a7, a5, 0  #max place
26  lw t2, 0(a7)     #max
27  jal zero, loop2
28  loop2_exit:
29
30  addi a3, a3, -1 # a3 = a3 - 4      len -1
31
32  lw t1, 0(a6) # t1 = array[last]
33  lw t0, 0(a7) # t0 = array[i]
34  sw t1, 0(a7) # array[i] = t1
35  sw t0, 0(a6) # array[last] = t0
36
37  addi a6, a6, -4 # a6 = a6 + (-4) = a6 - 4      last element -1
38
39  jal zero, loop # goto loop
40 loop_exit:
41 finish:
```

```
42  li a0, 10 # x10 = 10
43  li a1, 0 # x11 = 0
44  ecall # ecall при значении x10 = 10 => останов симулятора
45 .rodata
46 array_length:
47  .word 10
48 .data
49 array:
50  .word 6, 7, 8, 9, 9, 1, 2, 3, 4, 5
```

Теперь сделаем эту программу подпрограммой - будем ее вызывать из другого места - из main. Код, обеспечивающий вызов main и завершение работы, может использоваться «как есть» в самых разных программах. Учитывая это, мы разобьем текст программы на 2 файла (не учитвая саму подпрограмму): setup.s и main.s. Все параметры будем передавать из main. Листинг программы setup.s:

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

Листинг программы main.s:

```
1 # main.s
2 .text
3 main:
4 .globl main
5  addi sp, sp, -16 # выделение памяти в стеке
6  sw ra, 12(sp) # сохранение ra
7
8  la a0, array # }
9  lw a1, array_length # } sort( array, array_length );
10  call sort # }
11  li a0, 0 # }
12
13  lw ra, 12(sp) # восстановление ra
14  addi sp, sp, 16 # освобождение памяти в стеке
15  ret # } return 0;
16 .rodata
17 array_length:
18  .word 10
19 .data
20 array:
21  .word 6, 7, 8, 9, 9, 1, 2, 3, 4, 5
```

Листинг подпрограммы:

```
1 \#sort
2 .text
3 sort:
4 .globl sort
5
6  # в a0 - адрес го0- элемента массива чисел типа unsigned
7  # в a1 - длина массива
8
9  slli a6, a1, 2 # a6 = a1*4
10  add a6, a0, a6 # a6 = a0+a1*4
```

```
11  addi a6, a6, -4        #last el
12
13 loop:
14  bgeu zero, a1, loop_exit # if( 0 >= a1 ) goto loop_exit
15
16  li a2, 1 # a2 = 1
17  addi a7, a0, 0  #max place
18  lw t2, 0(a7)  #max = 1 element
19  add a5, a0, zero   #a5 = first element
20
21  loop2:
22  bgeu a2, a1, loop2_exit # if( a2 >= a1 ) goto loop_exit
23  addi a5, a5, 4 # a5 = a5 +  4
24  addi a2, a2, 1 # a2 += 1
25  lw t3, 0(a5)
26  bgeu t2, t3, loop2 # if( t2 >= t3 ) goto loop_exit
27  addi a7, a5, 0  #max place
28  lw t2, 0(a7)     #max
29  jal zero, loop2
30  loop2_exit:
31
32  addi a1, a1, -1 # a1 = a1 - 4      len -1
33
34  lw t1, 0(a6) # t1 = array[last]
35  lw t0, 0(a7) # t0 = array[i]
36  sw t1, 0(a7) # array[i] = t1
37  sw t0, 0(a6) # array[last] = t0
38
39  addi a6, a6, -4 # a6 = a6 + (-4) = a6 - 4      last element -1
40
41  jal zero, loop # goto loop
42 loop_exit:
43
44 ret
```

Обе программы реализуют заданный функционал, в чем можно удостовериться, запустив их в симуляторе.

5. Выводы

- Получены навыки работы с RISC-V, изучены его особенности.
- Разработана программа на языке ассемблера RISC-V, реализующая сортировку массива выбором. Массив данных и другие параметры располагаются в памяти по фиксированным адресам.
- Также функциональная часть выделена в подпрограмму и разработана использующая ее тестовая программа. Произведено ручное тестирования при различных значениях массива, которое позволяет утверждать, что программа работает корректно.