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

Вариант: 9

Выполнил студент гр. 3530901/90002 \_\_\_\_\_\_\_\_\_\_\_\_С. В. Пименов

(подпись)

Принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. С. Степанов

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Задача:** разработать программу на языке ассемблера RISC-V, реализующую вывод n-й строки треугольника Паскаля, отладить программу в симуляторе VSim. Массив (массивы) данных и другие параметры (преобразуемое число, длина массива, параметр статистики и пр.) располагаются в памяти по фиксированным адресам.

Выделить вывод n-й строки треугольника Паскаля 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.

|  |
| --- |
|  |

Рис 1. Псевдокод на языке Kotlin

При инициализации выравнивается показатель треугольника Паскаля для корректного вывода:

0: 1

1: 1 1

2: 1 2 1

…

Далее описывается цикл с помощью индекса и условия перехода. Выводится на печать число. Каждое последующее рассчитывается по приведенной ф-ле. При этом в программе сначала находится разность между показателем и индексом, далее выполняется умножение, а уже после целочисленное деление (с округлением вниз).

# Листинг

.text

start:

.globl start

li a0, 5 # показатель треугольника Паскаля, a0 = 3

li a2, 4 # const = 4, a2 = 4

li a3, 1 # const = 1, a3 = 1

add a1, a0, a3 # выделение памяти под элементы, a1 = a0 + a3

li a0, 9 # a0 = 9

ecall # системный вызов

sw a3, 0(a0) # записать 4 байта из x3 по адресу x0

mv s0, a0 # s0 = a0

li t1, 1 # t1 = 1

loop: # for (t1 = 1; t1 < t1 <= a1; t1++)

mv t2, t1 # t2 = t1

subloop: #for (t2 = t1; t2 >= 1; t2--)

sub t3, t2, a3 #синтез адреса памяти c[t2-1]

mul t3, a2, t3 # побитовое перемножение t3 и a2 с помещением полученных младших битов в t3

add t0, a0, t3 # t0 = a0 + t3

lw t4, 0(t0) # считать 4 байта по адресу t0 в a4

mul t3, a2, t2 #синтез адреса памяти c[t2]

add t0, a0, t3 # t0 = a0 + t3

lw t5, 0(t0) # считать 4 байта из t0 в t5

add t3, t4, t5 #c[t2] = c[t2-1]+c[t2]

sw t3, 0(t0) # записать 4 байта из x3 по адресу t0

sub t2, t2, a3 # вычитание, t2 = t2 - a3

bge t2, a3, subloop #t2 >= 1, вложенный цикл

add t1, t1, a3 # сумма t1 и a3, t1 = t1 + a3

blt t1, a1, loop # t1 < a1, выход из вложенного цикла

li t0, 0 # занулил t0

mv t1, a1 #записал в t1 адрес сегмента

sub a1, a1, a3 # вычитание, a1 = a1 - a3

print:

# напечатал число

li a0, 1 # a0 = 1

lw a1, 0(s0) # a1 = s0

ecall # системный вызов

li a0, 11 # a0 = 11

# напечатал пробел

li a1, ' ' # a1 = ' '

ecall # системный вызов

add s0, s0, a2 # синтезировал следующий адрес

add t0, t0, a3 # t0 = t0 + 1

bne t0, t1, print # t0 != t1 вызов корутины печати

li a0, 10 # a0 = 10

ecall # системный вызов

Вывод для данной программы, показатель = 3:

1 3 3 1

# Программа с подпрограммой

.text

start:

.globl start

call main # вызов основной подпрограммы

main:

.globl main

addi sp, sp, -16 # выделение памяти в стеке

sw ra, 12(sp) # сохранение ra

li a0, 5 # показатель треугольника Паскаля

add a1, a0, a3 # резервация сегмента под нужное количество элементов

li a0, 9 # a0 = 9

ecall # системный вызов

mv s0, a0 # сохранил адрес сегмента

mv s1, a1 # и размер

addi s1, s1, 1

mv a2, s0 # прописал их в параметры

mv a3, s1 # a3 = s1

jal ra, pascal # вызвали pascal

mv a2, s0 # приписали параметры

mv a3, s1 # a3 = s1

jal ra, print # вызвали print

mv a2, s0 # прописал параметры

mv a3, s1 # a3 = s1

li a0, 10 # a0 = 10

ecall # системный вызов

lw ra, 12(sp) # восстановление ra

addi sp, sp, 16

finish: # ф-ция завершения программы

mv a1, a0 # a1 = a0

li a0, 17 # a0 = 17

ecall # системный вызов

pascal: # void (int \*a2, int a3)

li t0, 1 # const 1

li t1, 4 # const 4

sw t0, 0(a2) # записать 4 байта по адресу a2

li t2, 1 # инициализация счетчика t2

loopPascal: # for (t2 = 1; t2 <= a3; t2++)

mv t3, t2 #инициализация счетчика t3

subPascal: # for (t3 = t2; t3 >= 1; t3--)

sub t4, t3, t0 # синтезируем адрес памяти c[t3-1]

mul t4, t4, t1 # перемножить t4 = t4 \* t1

add t5, a2, t4 # t5 = a2 + t4

lw t6, 0(t5) # считать 4 байта по адресу t5

mul t4, t3, t1 # синтезируем адрес памяти c[t3]

add t5, a2, t4 # t5 = a2 + t4

lw t4, 0(t5) # считать 4 байта по адресу t5

add t4, t4, t6 # c[t3] = c[t3-1]+c[t3]

sw t4, 0(t5) # записать 4 байта по адресу t5 из t4

sub t3, t3, t0 # t3 = t3 - t0

bge t3, t0, subPascal # t3 >= 1, subloop

add t2, t2, t0 # t2 = t2 + t0

blt t2, a3, loopPascal # t1 < a1, loop

jr ra # return

print:

# напечатать содержимое сегмента a2 размером a3

li t0, 0 # t0 = 0

conPrint:

# напечатал число

li a0, 1 # a0 = 1

lw a1, 0(a2) # a1 = a2

ecall # системный вызов

li a0, 11 # a0 = 11

# напечатал пробел

li a1, ' ' # a1 = ' '

ecall # системный вызов

addi a2, a2, 4 # a2 = a2 & 4

addi t0, t0, 1 # t0 = t0 & 1

bne t0, a3, conPrint # t0 != a3 вызов корутины печати

li a1, '\n' # a1 = '\n'

ecall # системный вызов

jr ra # возврат

Вывод для данной программы, показатель = 5:

1 5 10 10 5 1

Легко заметить, что программа работает корректно.

Таблица с использованными инструкциями:

|  |  |  |
| --- | --- | --- |
| **инструкция** | **аргументы** | **описание** |
| add | rd, r1, r2 | rd = r1 + r2 |
| addi | rd, r1, N | rd = r1 + N |
| and | rd, r1, r2 | rd = r1 & r2 |
| andi | rd, r1, N | rd = r1 & N |

|  |  |  |
| --- | --- | --- |
| **инструкция** | **аргументы** | **описание** |
| beq | r1, r2, addr | if(r1==r2)goto addr |
| beqz | r1, addr | if(r1==0)goto addr |
| bgeu | r1, r2, addr | if(r1>=r2)goto addr |
| bgtu | r1, r2, addr | if(r1> r2)goto addr |
| bltu | r1, r2, addr | if(r1< r2)goto addr |
| bne | r1, r2, addr | if(r1!=r2)goto addr |
| bnez | r1, addr | if(r1!=0)goto addr |
| call | func | вызов функции func |
| csrr | rd, csr | rd = csr |
| csrrs | rd, csr, N | rd = csr; csr |= N, атомарно |

|  |  |  |
| --- | --- | --- |
| **инструкция** | **аргументы** | **описание** |
| csrs | scr, rs | csr |= rs |
| csrs | scr, N | csr |= N |
| csrw | csr, rs | csr = rs |
| ecall |  | провоцирование исключения для входа в ловушку |
| j | addr | goto addr |
| la | rd, addr | rd = addr |
| lb | rd, N(r1) | считать 1 байт по адресу r1+N |
| lh | rd, N(r1) | считать 2 байта по адресу r1+N |
| li | rd, N | rd = N |

|  |  |  |
| --- | --- | --- |
| **инструкция** | **аргументы** | **описание** |
| lw | rd, N(r1) | считать 4 байта по адресу r1+N |
| mret |  | возврат из обработчика исключения |
| mv | rd, rs | rd = rs |
| or | rd, r1, r2 | rd = r1 | r2 |
| ori | rd, r1, N | rd = r1 | N |
| ret |  | возврат из функции |
| sb | rs, N(r1) | записать 1 байт по адресу r1+N |
| sh | rs, N(r1) | записать 2 байта по адресу r1+N |
| slli | rd, r1, N | rd = r1 << N |

|  |  |  |
| --- | --- | --- |
| **инструкция** | **аргументы** | **описание** |
| srli | rd, r1, N | rd = r1 >> N |
| sw | rs, N(r1) | записать 4 байта по адресу r1+N |
| xor | rd, r1, r2 | rd = r1 ^ r2 |
| xori | rd, r1, N | rd = r1 ^ N |

Взята с сайта: <https://habr.com/ru/post/533272/>

# Выводы

Был получен практический опыт работы с RISC-V, написаны программа и программа с подпрограммой.

При разработке использовался симулятор: <https://github.com/andrescv/Jupiter/releases/tag/v2.0.2>