

ИДЗ-2 Кокорев Артём  
Вариант 22

Файлы расположены в публичном репозитории <https://github.com/w1sq/ABC>

Задание:

Разработать программы на языке Ассемблера процесса RISC-V, с использованием команд арифметического сопроцессора, выполняемые в симуляторе RARS. Разработанные программы должны принимать числа в допустимом диапазоне. Например, нужно учитывать области определения и допустимых значений, если это связано с условием задачи.

Разработать программу вычисления числа  $\pi$  с точностью не хуже 0,1% посредством дзета-функции Римана.

Код на 9 баллов:

```
.data
# Константы
pi_const: .double 3.14159265358979323846 # Эталонное значение
π
epsilon: .double 0.001 # Точность 0.1%
one: .double 1.0
two: .double 2.0
six: .double 6.0
zero: .double 0.0
# Сообщения
prompt_mode: .string "Выберите режим (1 – ручной ввод точности,
2 – автотест): "
prompt_epsilon: .string "Введите точность вычисления (например,
0.001): "
result_msg: .string "Вычисленное значение π = "
test_msg: .string "Тест #"
accuracy_msg: .string "Точность: "
error_msg: .string "Ошибка: точность должна быть положительной\
n"
correct_msg: .string "Тест пройден успешно\n"
incorrect_msg: .string "Тест не пройден\n"
newline: .string "\n"
# Массив тестовых точностей
test_epsilons: .double 0.1, 0.01, 0.001
test_count: .word 3
```

```
.text
.globl main
```

```
# Макросы для работы с числами с плавающей точкой
```

```
.macro load_double(%reg, %label)
la t0, %label
fld %reg, 0(t0)
.end_macro
```

```
.macro store_double(%reg, %label)
la t0, %label
fsd %reg, 0(t0)
.end_macro
```

```
.macro print_double(%reg)
fmv.d fa0, %reg
li a7, 3
ecall
.end_macro
```

```
.macro read_double(%dest)
li a7, 7
ecall
fmv.d %dest, fa0
.end_macro
```

```
# Макрос для вывода строки
```

```
.macro print_string(%label)
la a0, %label
li a7, 4
ecall
.end_macro
```

```
# Макрос для ввода целого числа
```

```
.macro read_int(%reg)
li a7, 5
```

```
ecall
mv %reg, a0
.end_macro
```

*main:*

*# Сохраняем регистры*

```
addi sp, sp, -16
sw ra, 0(sp)
fsw fs0, 8(sp)
```

*# Выбор режима работы*

```
print_string(prompt_mode)
read_int(t0)
```

```
li t1, 1
beq t0, t1, manual_mode
li t1, 2
beq t0, t1, auto_test
j main
```

*manual\_mode:*

*# Ручной ввод точности*

```
print_string(prompt_epsilon)
read_double(fs0)
```

*# Проверка корректности точности*

```
la t0, one
fld ft0, 0(t0)
fsub.d ft0, ft0, ft0 # Получаем ноль путем вычитания
flt.d t0, fs0, ft0
bnez t0, epsilon_error
```

*# Вычисление  $\pi$*

```
fmv.d fa0, fs0
jal calculate_pi
```

*# Вывод результата*

```
print_string(result_msg)
print_double(fa0)
print_string(newline)
```

```
j exit
```

```
epsilon_error:
```

```
print_string(error_msg)
```

```
j manual_mode
```

```
auto_test:
```

```
la s0, test_epsilon
```

```
lw s1, test_count
```

```
li s2, 0
```

```
auto_test_loop:
```

```
addi s2, s2, 1 # Увеличиваем номер теста
```

```
bgt s2, s1, exit
```

```
# Вывод информации о тесте
```

```
print_string(test_msg)
```

```
mv a0, s2
```

```
li a7, 1
```

```
ecall
```

```
print_string(newline)
```

```
# Загрузка тестовой точности
```

```
addi t0, s2, -1 # Индекс начинается с 0
```

```
slli t0, t0, 3 # Умножаем на 8 (размер double)
```

```
add t0, s0, t0 # Добавляем к базовому адресу
```

```
fld fs0, 0(t0) # Загружаем значение
```

```
print_string(accuracy_msg)
```

```
print_double(fs0)
```

```
print_string(newline)
```

```
# Вычисление  $\pi$ 
```

```
fmv.d fa0, fs0
```

```
jal calculate_pi
```

```
# Вывод результата
```

```
print_string(result_msg)
```

```
print_double(fa0)
```

```
print_string(newline)
```

```
# Проверка результата
```

```
jal check_result
```

```
j auto_test_loop
```

```

# Подпрограмма вычисления  $\pi$  через дзета-функцию Римана
calculate_pi:
addi sp, sp, -32
sw ra, 0(sp)
fsd fs0, 8(sp)
fsd fs1, 16(sp)
fsd fs2, 24(sp)
fmv.d fs0, fa0 # Сохраняем точность
# Определяем количество итераций на основе точности
li t1, 100 # Базовое количество итераций
load_double(ft0, one)
li t0, 10
fcvt.d.w ft1, t0
fdiv.d ft0, ft0, ft1 #  $ft0 = 0.1$ 
flt.d t0, fs0, ft0 # точность < 0.1?
beqz t0, init_vars # если нет, используем 100 итераций
li t1, 1000 # Увеличиваем количество итераций
fdiv.d ft0, ft0, ft1 #  $ft0 = 0.01$ 
flt.d t0, fs0, ft0 # точность < 0.01?
beqz t0, init_vars # если нет, используем 1000 итераций
li t1, 10000 # Увеличиваем количество итераций
fdiv.d ft0, ft0, ft1 #  $ft0 = 0.001$ 
flt.d t0, fs0, ft0 # точность < 0.001?
beqz t0, init_vars # если нет, используем 10000 итераций
li t1, 50000 # Максимальное количество итераций
init_vars:
load_double(fs2, one) #  $n = 1$ 
fmv.d fs1, ft0 #  $zeta_2 = 0$  (используем ft0, который уже
обнулен)
li t2, 0 # счетчик итераций

sum_loop:
bge t2, t1, sum_done
# Вычисляем  $1/n^2$ 
fmv.d fa0, fs2
fmul.d fa0, fa0, fa0 #  $n^2$ 

```

```

load_double(ft0, one)
fdiv.d ft0, ft0, fa0 #  $1/n^2$ 
# Добавляем к сумме
fadd.d fs1, fs1, ft0
# Увеличиваем n и счетчик
load_double(ft0, one)
fadd.d fs2, fs2, ft0 #  $n++$ 
addi t2, t2, 1 # счетчик++
j sum_loop

```

*sum\_done:*

```

#  $\pi^2 = 6 * \zeta(2)$ 
load_double(ft0, six)
fmul.d fs1, fs1, ft0
fsqrt.d fa0, fs1
# Восстанавливаем регистры
fld fs2, 24(sp)
fld fs1, 16(sp)
fld fs0, 8(sp)
lw ra, 0(sp)
addi sp, sp, 32
ret

```

*# Подпрограмма проверки результата*

*check\_result:*

```

addi sp, sp, -32
fsd fs0, 0(sp)
fsd fs1, 8(sp)
fsd fs2, 16(sp) # Сохраняем текущую точность
fmv.d fs2, fs0 # Сохраняем текущую точность
fmv.d fs0, fa0 # Сохраняем вычисленное значение
la t0, pi_const
fld fs1, 0(t0)
# Вычисляем относительную погрешность
fsub.d ft0, fs0, fs1
fabs.d ft0, ft0
fdiv.d ft0, ft0, fs1

```

```

# Сравниваем с текущей точностью
flt.d t0, ft0, fs2 # Сравниваем с сохраненной точностью из fs2
beqz t0, check_fail
print_string(correct_msg)
j check_done
check_fail:
print_string(incorrect_msg)
check_done:
fld fs2, 16(sp) # Восстанавливаем текущую точность
fld fs1, 8(sp)
fld fs0, 0(sp)
addi sp, sp, 32
ret

exit:
# Восстанавливаем регистры и завершаем программу
lw ra, 0(sp)
flw fs0, 8(sp)
addi sp, sp, 16
li a7, 10
ecall

```

Метод решения задачи заключается в вычислении числа  $\pi$  через дзета-функцию Римана. Конкретно используется следующий алгоритм:

1. Математическая основа2:
  - Используется связь между  $\zeta(2)$  (дзета-функцией Римана для  $s=2$ ) и числом  $\pi$
  - $\zeta(2) = \pi^2/6$
  - Следовательно,  $\pi = \sqrt{6 * \zeta(2)}$
2. Вычисление  $\zeta(2)$ :
  - $\zeta(2) = 1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 + \dots$
  - Это бесконечный ряд, который сходится к  $\pi^2/6$
  - В программе мы суммируем конечное число членов ряда
3. Алгоритм:
  1. Инициализация суммы ( $\zeta(2) = 0$ )
  2. Для  $n$  от 1 до  $N$ :
    - Вычисляем  $1/n^2$
    - Добавляем результат к сумме
  3. Умножаем сумму на 6
  4. Извлекаем квадратный корень

#### 4. Точность вычислений:

- Количество итераций (N) определяется требуемой точностью
- Чем меньше требуемая точность, тем больше итераций нужно
- В программе используется следующая зависимость:
  - \* для точности 0.1: 100 итераций
  - \* для точности 0.01: 1000 итераций
  - \* для точности 0.001: 10000 итераций
  - \* для более высокой точности: 50000 итераций

#### 5. Проверка результата:

- Вычисляется относительная погрешность:  $|\pi_{\text{вычисленное}} - \pi_{\text{точное}}| / \pi_{\text{точное}}$
- Если погрешность меньше заданной точности, тест считается успешным

Источник: [https://ru.wikipedia.org/wiki/%D0%A0%D1%8F%D0%B4\\_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D1%85\\_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2](https://ru.wikipedia.org/wiki/%D0%A0%D1%8F%D0%B4_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D1%85_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2)

#### Результаты запуска:

```
Выберите режим (1 - ручной ввод точности, 2 - автотест): 1
Введите точность вычисления (например, 0.001): 0.001
Вычисленное значение  $\pi$  = 3.1424519775309836
```

```
-- program is finished running (0) --
```

```
Выберите режим (1 - ручной ввод точности, 2 - автотест): 2
Тест #1
Точность: 0.1
Вычисленное значение  $\pi$  = 3.226438191118707
Тест пройден успешно
Тест #2
Точность: 0.01
Вычисленное значение  $\pi$  = 3.150175772887819
Тест пройден успешно
Тест #3
Точность: 0.001
Вычисленное значение  $\pi$  = 3.1424519775309836
Тест пройден успешно
```

```
-- program is finished running (0) --
```