АВС. ИДЗ-2

Гареев Данир БПИ-236

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.

Вариант 12

Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже $0,05\%$ значение функции $tan(x)$ для заданного параметра x.

Решение на 10 баллов

Для вычисления значения функции tan(x) с помощью степенного ряда с точностью не хуже $0,05\%$, можно использовать разложение в ряд Тейлора. Однако, разложение $tan(x)$ в ряд Тейлора не является простым, так как оно содержит числа Бернулли, что делает его сложным для реализации. Вместо этого, можно воспользоваться разложением $tan(x)$ через ряд Тейлора для $sin(x)$ и $cos(x)$, а затем разделить их.
Файлы хранятся на GitHub по ссылке.

Файлы:

1. **main.asm** – файл для взаимодействия с пользователем. Предлагается выбор автотестов, либо ввод собсвтенного значения. Предусмотрен повторный ввод данных.
2. **macros.asm** – библиотека с макросами:
   - **print_str(%str)** – макрос для вывода строки.
   - **print_float(%reg)** – макрос для вывода числа с плавующей точкой.
   - **print_pass_msg(%x)** – макрос для вывода уведомления о успешном тесте.
   - **print_fail_msg(%x)** – макрос для вывода уведомления о проваленном тесте.
   - **print_newline(%x)** – макрос для перехода на новую строку.
   - **print_test_result(%tan_result, %expected)** – макрос для вывода тестовых значений в формате $tan\_result =$ вычисленное значение $tan(x)$, $expected =$ ожидаемое значение
3. **solution.asm** – основное вычисление $tan(x)$.
4. **tests.asm** – автотесты:
   - $\frac{\pi}{4}$
   - $\frac{\pi}{6}$
   - $\frac{\pi}{3}$
   - $\frac{\pi}{0}$
   - $\frac{\pi}{12}$

Выполненные требования

На 4-5

1. Есть решение на ассемблере. Есть ввод и вывод данных на экран.
2. Есть комментарии, поясняющие выполняемые действия.
3. Есть полное тестовое покрытие.
4. Буфер для программы имеет фиксированный размер.

На 6-7

1. Используются подпрограммы с передачей аргументов через параметры.
2. Сохранение локальных переменных в свободных регистрах.
3. Есть комментарии к функциям.
4. Вся информация добавлена в отчёт.

На 8

1. Программа поддерживает мнократное использование с различным набором входных данных.
2. Есть доп. тестовая программа для автоматического тестирования.
3. Реализована аналогичная программа на python для проверки точности вычислений.

На 9

1. Добавлены макросы для реализации ввода вывода.

На 10

1. Программа разбита на несколько файлов.
2. Макросы выделены в отдельную автономную библиотеку.

Выполненые условия задачи

Результаты автотестов

```
Enter 1 to run tests, 2 to enter your own value: 1
tan(x) = 0.99999917 expected tan(x) = 1.0 delta = 8.34465E-5%
tan(x) = 0.57735074 expected tan(x) = 0.57735 delta = 1.2388598E-4%
tan(x) = 1.732059 expected tan(x) = 1.73205 delta = 5.230742E-4%
tan(x) = 0.0 expected tan(x) = 0.267949 delta = 100.0%
tan(x) = 0.26794878 expected tan(x) = 0.267949 delta = 7.785671E-5%
All tests passed!
If you want to finish program, enter '0' or press any other number to restart:
```

Пример работы программы

```
Enter 1 to run tests, 2 to enter your own value: 1
tan(x) = 0.99999917 expected tan(x) = 1.0 delta = 8.34465E-5%
tan(x) = 0.57735074 expected tan(x) = 0.57735 delta = 1.2388598E-4%
tan(x) = 1.732059 expected tan(x) = 1.73205 delta = 5.230742E-4%
tan(x) = 0.0 expected tan(x) = 0.267949 delta = 100.0%
tan(x) = 0.26794878 expected tan(x) = 0.267949 delta = 7.785671E-5%
All tests passed!
If you want to finish program, enter '0' or press any other number to restart: 1
Enter 1 to run tests, 2 to enter your own value: 2
Enter a value for x: 2
tan(x) = -2.185041
If you want to finish program, enter '0' or press any other number to restart: 0

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

Аналогичная программа на Python

hw-2.py

```
import math

def compute_sin(x, epsilon=0.0005):
    term = x
    sin_x = term
    n = 1
    while abs(term) > epsilon:
```

```
        term *= -x * x / ((2 * n) * (2 * n + 1))
        sin_x += term
        n += 1
    return sin_x

def compute_cos(x, epsilon=0.0005):
    term = 1
    cos_x = term
    n = 1
    while abs(term) > epsilon:
        term *= -x * x / ((2 * n - 1) * (2 * n))
        cos_x += term
        n += 1
    return cos_x

def compute_tan(x, epsilon=0.0005):
    sin_x = compute_sin(x, epsilon)
    cos_x = compute_cos(x, epsilon)
    if abs(cos_x) < epsilon:
        raise ValueError("cos(x) is too close to zero, tan(x) is undefined")
    return sin_x / cos_x

#                                     x
test_values = [0, math.pi/12, math.pi/6, math.pi/4, math.pi/3]

#                                       tan(x) (
                  math.tan)
expected_tan_values = [math.tan(x) for x in test_values]

#
for x, expected_tan in zip(test_values, expected_tan_values):
    try:
        computed_tan = compute_tan(x)
        if expected_tan == 0:
            delta_percent = abs(computed_tan - expected_tan) * 100  #
                                                    0
        else:
            delta_percent = abs(computed_tan - expected_tan) / expected_tan * 100
        print(f"tan({x:.6f}) = {computed_tan:.8f} expected tan(x) = {expected_tan
           :.8f} delta = {delta_percent:.8E}%")
    except ValueError as e:
        print(f"tan({x:.6f}) is undefined: {e}")
```

Результат

```
tan(0.000000) = 0.00000000 expected tan(x) = 0.00000000 delta = 0.00000000E+00%
tan(0.261799) = 0.26794909 expected tan(x) = 0.26794919 delta = 3.97828583E-05%
tan(0.523599) = 0.57735282 expected tan(x) = 0.57735027 delta = 4.42647134E-04%
tan(0.785398) = 1.00000460 expected tan(x) = 1.00000000 delta = 4.60293811E-04%
tan(1.047198) = 1.73204104 expected tan(x) = 1.73205081 delta = 5.63823144E-04%
```

Исходный код

main.asm

```
.include "macros.asm"
.include "solution.asm"
.include "tests.asm"

.data
    prompt: .asciz "Enter 1 to run tests, 2 to enter your own value: "
    input_prompt: .asciz "Enter a value for x: "
```

```
    prompt_next: .asciz "If you want to finish program, enter '0' or press any
       other number to restart: "
    result_msg: .asciz "tan(x) = "

.text
.globl main
main:
    print_str(prompt)
    li a7, 5
    ecall
    li t0, 1
    beq a0, t0, run_tests
    li t0, 2
    beq a0, t0, user_input
    j main

user_input:
    print_str(input_prompt)
    input_float()
    call tan
    print_str(result_msg)
    print_float(fa0)
    print_newline()
    li   a7, 4
    la   a0, prompt_next
    ecall
    li   a7, 5
    ecall
    beqz a0, end
    j main

to_tests:
    call run_tests
end:
    li   a7, 10
    ecall
```

macros.asm

```
.data
    pass_msg: .asciz "All tests passed!\n"
    fail_msg: .asciz "Test failed!\n"
    newline: .asciz "\n"
    tan_msg: .asciz "tan(x) = "
    expected_msg: .asciz " expected tan(x) = "
    delta_msg: .asciz " delta = "
    percent_msg: .asciz "%\n"

.text
# Macro for printing a string
.macro print_str(%str)
    li a7, 4
    la a0, %str
    ecall
.end_macro

# Macro for reading a floating-point number
.macro input_float()
    li a7, 6
    ecall
.end_macro

# Macro for printing a floating-point number
.macro print_float(%reg)
    fmv.s fa0, %reg
    li a7, 2
```

```
    ecall
.end_macro

# Macro for printing a success message
.macro print_pass_msg()
    li a7, 4
    la a0, pass_msg
    ecall
.end_macro

# Macro for printing a failure message
.macro print_fail_msg()
    li a7, 4
    la a0, fail_msg
    ecall
.end_macro

# Macro for printing a newline
.macro print_newline()
    li a7, 4
    la a0, newline
    ecall
.end_macro

# Macro for printing the test result
# Input: %tan_result = computed tan(x), %expected = expected value
.macro print_test_result(%tan_result, %expected)
    # Save registers
    addi sp, sp, -16
    fsw fs0, 0(sp)
    fsw fs1, 4(sp)
    fsw fs2, 8(sp)
    sw ra, 12(sp)

    # Load values
    fmv.s fs0, %tan_result  # fs0 = tan(x)
    fmv.s fs1, %expected    # fs1 = expected tan(x)

    # Compute delta = |tan(x) - expected| / expected * 100
    fsub.s ft0, fs0, fs1    # ft0 = tan(x) - expected
    fabs.s ft0, ft0         # ft0 = |tan(x) - expected|
    fdiv.s ft0, ft0, fs1    # ft0 = |tan(x) - expected| / expected
    li t0, 100
    fcvt.s.w ft1, t0        # ft1 = 100.0
    fmul.s ft0, ft0, ft1    # ft0 = delta (in percentage)

    # Print result
    print_str(tan_msg)
    print_float(fs0)
    print_str(expected_msg)
    print_float(fs1)
    print_str(delta_msg)
    print_float(ft0)
    print_str(percent_msg)

    # Restore registers
    flw fs0, 0(sp)
    flw fs1, 4(sp)
    flw fs2, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
.end_macro
```

solution.asm

```
.data
```

```
    epsilon: .float 0.00005

.text
# function to compute sin(x)
# input: fa0 = x
# output: fa0 = sin(x)
sin:
    fmv.s fa1, fa0          # fa1 = x
    fmul.s ft0, fa1, fa1    # ft0 = x^2 (saving x  for multiplication)
    fmv.s fa3, fa1          # fa3 = sum (starting with x)
    fmv.s fa2, fa1          # fa2 = current term of the series (x)
    li t0, 1                # t0 = n (iteration number, starting from 1)
    li t1, -1               # t1 = sign of the next term (-1)

sin_loop:
    # computing the denominator (2n)(2n + 1)
    li t2, 2
    mul t3, t0, t2          # t3 = 2n
    addi t4, t3, 1          # t4 = 2n + 1
    mul t5, t3, t4          # t5 = (2n)(2n + 1)
    # converting the denominator to float
    fcvt.s.w ft1, t5        # ft1 = (2n)(2n + 1)
    # updating the current term: term = term * (-x  ) / ((2n)(2n + 1))
    fmul.s fa2, fa2, ft0    # multiplying by x
    fdiv.s fa2, fa2, ft1    # dividing by (2n)(2n + 1)
    fcvt.s.w ft2, t1        # ft2 = sign (-1 or 1)
    fmul.s fa2, fa2, ft2    # multiplying by sign
    # adding to the sum
    fadd.s fa3, fa3, fa2
    # preparing for the next iteration
    addi t0, t0, 1          # increasing n
    # precision check
    fabs.s ft3, fa2         # |current term|
    la t6, epsilon
    flw ft4, 0(t6)          # loading epsilon
    flt.s t6, ft3, ft4      # |current term| < epsilon?
    beqz t6, sin_loop       # if not, continue loop
    fmv.s fa0, fa3          # returning sum
    ret

# function to compute cos(x)
# input: fa0 = x
# output: fa0 = cos(x)
cos:
    #fmv.s fa1, fa0          # fa1 = x
    li t0, 1                # t0 = n (iteration number, starting from 1)
    li t1, -1               # t1 = sign of the next term (-1)
    fmul.s ft0, fa1, fa1    # ft0 = x^2 (saving x  for multiplication)
    fcvt.s.w fa3, t0        # fa3 = sum (starting with x)
    fcvt.s.w fa2, t0        # fa2 = current term of the series (x)

cos_loop:
    # computing the denominator (2n)(2n - 1)
    li t2, 2
    mul t3, t0, t2          # t3 = 2n
    addi t4, t3, -1         # t4 = 2n - 1
    mul t5, t3, t4          # t5 = (2n)(2n - 1)
    # converting the denominator to float
    fcvt.s.w ft1, t5        # ft1 = (2n)(2n - 1)
    # updating the current term: term = term * (-x  ) / ((2n)(2n - 1))
    fmul.s fa2, fa2, ft0    # multiplying by x
    fdiv.s fa2, fa2, ft1    # dividing by (2n)(2n - 1)
    fcvt.s.w ft2, t1        # ft2 = sign (-1 or 1)
    fmul.s fa2, fa2, ft2    # multiplying by sign
    # adding to the sum
```

```
    fadd.s fa3, fa3, fa2
    # preparing for the next iteration
    addi t0, t0, 1          # increasing n
    # precision check
    fabs.s ft3, fa2         # |current term|
    la t6, epsilon
    flw ft4, 0(t6)          # loading epsilon
    flt.s t6, ft3, ft4      # |current term| < epsilon?
    beqz t6, cos_loop       # if not, continue loop
    fmv.s fa0, fa3          # returning sum
    ret

# function to compute tan(x)
# input: fa0 = x
# output: fa0 = tan(x)
tan:
    addi sp, sp, -8         # allocating space in stack
    sw ra, 0(sp)            # saving ra
    call sin                # fa0 = sin(x)
    fmv.s fa4, fa0          # fa1 = sin(x)
    call cos                # fa0 = cos(x)
    fdiv.s fa0, fa4, fa0    # fa0 = sin(x) / cos(x)
    lw ra, 0(sp)            # restoring ra
    addi sp, sp, 8          # freeing stack
    ret
```

tests.asm

```
.data
    test_x1: .float 0.785398  # x =   /4, tan(  /4) = 1
    expected_tan1: .float 1.0
    test_x2: .float 0.523599  # x =   /6, tan(  /6)     0.57735
    expected_tan2: .float 0.57735
    test_x3: .float 1.0472    # x =   /3, tan(  /3)     1.73205
    expected_tan3: .float 1.73205
    test_x4: .float 0.0       # x = 0, tan(0) = 0
    expected_tan4: .float 0.0
    test_x5: .float 0.261799  # x =   /12, tan(  /12)     0.267949
    expected_tan5: .float 0.267949

.text
.globl run_tests
run_tests:
    # Test 1: x =   /4
    la t6, test_x1
    flw fa0, 0(t6)
    call tan
    la t6, expected_tan1
    flw ft0, 0(t6)
    print_test_result(fa0, ft0)

     # Test 2: x =   /6
    la t6, test_x2
    flw fa0, 0(t6)
    call tan
    la t6, expected_tan2
    flw ft0, 0(t6)
    print_test_result(fa0, ft0)

    # Test 3: x =   /3
    la t6, test_x3
    flw fa0, 0(t6)
    call tan
    la t6, expected_tan3
    flw ft0, 0(t6)
    print_test_result(fa0, ft0)
```

```
    # Test 4: x = 0
    la t6, test_x4
    flw fa0, 0(t6)
    call tan
    la t6, expected_tan5
    flw ft0, 0(t6)
    print_test_result(fa0, ft0)

    # Test 5: x =   /12
    la t6, test_x5
    flw fa0, 0(t6)
    call tan
    la t6, expected_tan5
    flw ft0, 0(t6)
    print_test_result(fa0, ft0)

    # All test passed
    print_pass_msg()
    li   a7, 4
    la   a0, prompt_next
    ecall
    li   a7, 5
    ecall
    beqz a0, end
    j main
```