НИУ ВШЭ

Архитектура вычислительных систем

Индивидуальное домашнее задание №2

Вариант 21

Студент: Лебедев Андрей Андреевич БПИ234

Москва
2024

1 Цель

Задача заключалась в разработке программы, вычисляющей число π с точностью не хуже 0.05% с использованием ряда Нилаканта. Решение было реализовано на языке ассемблера для архитектуры RISC-V с использованием операций с плавающей точкой и системных вызовов для ввода/вывода.

2 Обзор структуры программы

Программа состоит из нескольких компонентов:

- **Основная программа** (`main.s`): Обрабатывает взаимодействие с пользователем (ввод данных, валидация, управление выполнением).
- **Макросы вводавывода** (`io_macros.s`): Реализует вспомогательные макросы для вывода строк, чисел с плавающей точкой и ввода данных.
- **Подпрограмма для вычисления π с использованием ряда Нилаканта** (`pi_via_nilakantha.s`): Реализует вычисление числа π с помощью ряда Нилаканта.
- **Тестирование** (`test.s`): Запускает заранее заданные тесты с известными значениями числа π и точности, сравнивает их с вычисленными результатами и выводит результаты.

3 Вычисление числа π с использованием ряда Нилаканта

Ряд Нилаканта — это бесконечный ряд для приближенного вычисления числа π, который выглядит так:

$$\pi = 3 + \frac{4}{2 \cdot 3 \cdot 4} - \frac{4}{4 \cdot 5 \cdot 6} + \frac{4}{6 \cdot 7 \cdot 8} - \cdots$$

Каждый член ряда имеет вид:

$$\text{Член}_n = \frac{4}{(2n) \cdot (2n+1) \cdot (2n+2)}$$

Каждый член ряда добавляется или вычитается в зависимости от четности индекса. Изначально π устанавливается в 3, и затем добавляются или вычитаются члены этого ряда до тех пор, пока погрешность не станет меньше заданной точности.

4 Ключевые этапы программы

4.1 Основная программа (`main.s`)

Программа начинается с запроса у пользователя, желает ли он вычислить значение π с заданной точностью или запустить тесты. В случае выбора вычисления π, программа запрашивает точность и вызывает подпрограмму для вычисления числа π с использованием ряда Нилаканта. В случае тестирования программа выполняет заранее заданные тесты.

```
.include "io_macros.s"          # Include the macros library for I/O operations

.data
    prompt_result: .asciz "\nResult of computing pi: "         # Message to display the
        result of pi computation
    prompt_result_accuracy: .asciz "Accuracy (%): "            # Message to display the
        accuracy of pi computation

.text
.global main

# Main program entry point
main:
    # Input and validate data
    input_and_validate t1          # Input and validate the result (e.g., user input)

    # If t1 equals 0 (input error), jump to the testing section
    beqz t1, testing               # If t1 == 0, jump to the testing section

    # If t1 is non-zero, proceed with the computation
    bnez t1, computing             # If t1 != 0, jump to the computing section for pi
        calculation

# Pi computation using Nilakantha series
computing:
    input_and_validate_float fa0   # Input and validate the floating-point number (e.g.,
        accuracy or iteration limit)
    jal pi_via_nilakantha          # Jump to the subroutine for computing pi using the
        Nilakantha series
    fmv.s ft0, fa1                 # Move the computed pi result from register fa1 to ft0
        (floating-point register)
    fmv.s ft1, fa2                 # Move the accuracy result from register fa2 to ft1

    # Output the result
    print_string prompt_result      # Print the string with the result prompt
    print_float ft0                 # Print the computed value of pi (from ft0 register)
    print_string prompt_result_accuracy  # Print the string with the accuracy prompt
    print_float ft1                 # Print the computed accuracy (from ft1 register)

    # Exit the program
    j exit_program                  # Jump to the exit program label

# Testing section
testing:
    jal test                        # Jump to the test subroutine (e.g., tests for
        correctness)
    j exit_program                  # After testing, jump to the exit program label

# Program exit
exit_program:
    li a7, 10                       # System call code for exiting the program (code 10)
    ecall                           # Make the system call to exit the program
```

4.2 Макросы ввода/вывода (io_macros.s)

Эти макросы реализуют основные операции ввода/вывода, такие как вывод строк и чисел с плавающей точкой, а также ввод чисел с плавающей точкой.

```
.data
    prompt_start:              .asciz "To calculate the value of pi using the Nilakantha series, enter [1]\nTo run the tests, enter [0]\n"  # Prompt for the user to choose between calculating pi or running tests
    prompt_invalid_input:      .asciz "Invalid input! Please try again\n"  # Message displayed if the user enters invalid input
    prompt_input:              .asciz "Your decision: "  # Prompt for user input
    prompt_input_accuracy:     .asciz "\nInput Enter the percentage accuracy (0.000001 <= a <= 0.05): "  # Prompt for user input on accuracy
    space:                     .asciz " "  # Space character (not used in the provided code)
    new_line:                  .asciz "\n"  # Newline character for formatting output
    max:                       .float 0.051  # Maximum allowed accuracy value
    min:                       .float 0.000001  # Minimum allowed accuracy value

# Macro for printing a string to the screen
.macro print_string %str
    li a7, 4                 # System call for printing a string
    la a0, %str              # Load the address of the string into a0
    ecall                    # Make the system call to print the string
.end_macro

# Macro for inputting a floating-point number
.macro input_float %result
    li      a7, 6            # System call for reading a floating-point number
    ecall                    # Make the system call to read the input
    fmv.s   %result, fa0     # Move the floating-point result into the provided variable
.end_macro

# Macro for printing a floating-point value
.macro print_float %value
    fmv.s   fa0, %value      # Move the floating-point value into fa0 register for printing
    li      a7, 2            # System call for printing a floating-point number
    ecall                    # Make the system call to print the floating-point value
    print_string new_line    # Print a newline after the number
.end_macro

# Macro for input and validation of user decision (0 or 1)
.macro input_and_validate %value
    print_string prompt_start  # Print the prompt to the user (choose calculation or testing)
loop:
    print_string prompt_input  # Ask for user input (their decision)
    li a7, 5                   # System call for reading an integer (user's choice)
    ecall                      # Make the system call to read the input

    mv t1, a0                  # Move the input value into register t1
    li t2, 2                   # Load the value 2 into register t2 (to check the valid range)

    bltz t1, invalid_input     # If t1 < 0, jump to invalid_input (input is negative)
    bge t1, t2, invalid_input  # If t1 >= 2, jump to invalid_input (input is not 0 or 1)
    j done_input               # If input is valid (0 or 1), jump to done_input

invalid_input:
    print_string prompt_invalid_input  # Print the invalid input message
    j loop                             # Jump back to the input loop for retry

```

```
done_input:
    mv %value, a0              # Move the valid input value (0 or 1) to the provided variable
.end_macro


# Macro for input and validation of user decision on floating-point accuracy (0.000001 <= a <= 0.05)
.macro input_and_validate_float %value
    print_string prompt_start  # Print the prompt to the user (choose accuracy value)
    la t1, min                 # Load the address of the minimum accuracy value
    la t2, max                 # Load the address of the maximum accuracy value
    flw ft0, 0(t1)             # Load minimum accuracy value into ft0
    flw ft1, 0(t2)             # Load maximum accuracy value into ft1
loop:
    print_string prompt_input_accuracy  # Ask for user input on accuracy

    input_float ft2            # Get the floating-point input value (accuracy)

    flt.s t1, ft2, ft0         # Compare if input is less than minimum allowed value
    bnez t1, invalid_input2    # If less, jump to invalid_input2

    fge.s t1, ft2, ft1         # Compare if input is greater than or equal to maximum allowed value
    bnez t1, invalid_input2    # If greater, jump to invalid_input2

    j done_input2              # If input is valid, jump to done_input2

invalid_input2:
    print_string prompt_invalid_input  # Print the invalid input message
    j loop                             # Jump back to the input loop for retry

done_input2:
    fmv.s %value, ft2          # Move the valid input value (accuracy) to the provided variable
.end_macro
```

4.3 Подпрограмма для вычисления π (pi_via_nilakantha.s)

В этой подпрограмме реализовано вычисление числа π с использованием ряда Нилаканта. В цикле добавляются и вычитаются члены ряда, пока погрешность не станет меньше заданной точности.

```
.data
    pi:     .float 3.0         # Initialize pi with the initial approximation value 3.0
    one:    .float 1.0         # Constant value 1.0 for calculations
    four:   .float 4.0         # Constant value 4.0 for the numerator multiplier in the Nilakantha series
    two:    .float 2.0         # Constant value 2.0 for the denominator multiplier in the Nilakantha series
    hundred:    .float 100.0   # Constant value 100.0, used in normalization
    reference: .float 3.141592653589793      # Reference value of pi (for comparison)

.text
.globl pi_via_nilakantha
```

```
# Subroutine for calculating pi using the Nilakantha series
pi_via_nilakantha:
    # Input:
    # fa0 - accuracy (desired precision)
    # Output:
    # fa1 - result of the pi computation (final value of pi)
    # fa2 - computed accuracy

    la t0, hundred              # Load the address of the constant value 100 into register t0
    flw ft9, 0(t0)              # Load the value of 100.0 into floating-point register ft9
    fdiv.s fa0, fa0, ft9        # Divide the input accuracy by 100.0 to normalize the input

    # Load addresses of variables into registers
    la t0, pi                   # Load address of pi into t0 register
    la t1, one                  # Load address of one into t1 register
    la t2, two                  # Load address of two into t2 register
    la t3, four                 # Load address of four into t3 register
    la t4, reference            # Load address of the reference pi value into t4 register

    # Load initial values into floating-point registers
    flw ft11, 0(t4)             # ft11 = reference pi (3.141592653589793)
    flw ft0, 0(t0)              # ft0 = pi = 3.0 (initial approximation)
    flw ft1, 0(t1)              # ft1 = 1.0 (used in calculations for n and sign)
    flw ft2, 0(t2)              # ft2 = 2.0 (used in denominator of Nilakantha series)
    flw ft3, 0(t3)              # ft3 = 4.0 (used as multiplier in the numerator of Nilakantha series)

    # Set initial values for iteration index (n) and alternating sign (+1 or -1)
    flw ft4, 0(t1)              # ft4 = n = 1 (starting term index)
    flw ft5, 0(t1)              # ft5 = sign = 1 (starting with positive sign)

loop:
    # Compute the current term of the Nilakantha series: 4 / (2n * (2n+1) * (2n+2))
    fmul.s ft6, ft2, ft4        # ft6 = 2n (denominator part 2n)
    fdiv.s ft7, ft3, ft6        # ft7 = 4 / (2n)
    fadd.s ft6, ft6, ft1        # ft6 = 2n + 1
    fdiv.s ft7, ft7, ft6        # ft7 = 4 / (2n * (2n+1))
    fadd.s ft6, ft6, ft1        # ft6 = 2n + 2
    fdiv.s ft7, ft7, ft6        # ft7 = 4 / (2n * (2n+1) * (2n+2))

    # Multiply the term by the sign (alternating +1 or -1)
    fmul.s ft7, ft7, ft5        # ft7 = term * sign

    # Update the value of pi: pi = pi + term
    fadd.s ft8, ft0, ft7        # ft8 = pi + term

    # Check if the required number of iterations has been completed (based on accuracy)
    flt.s t1, ft11, ft4         # Compare n (iteration count) with the reference limit (ft11)
    fmv.s ft0, ft8              # Update the value of pi in ft0

    # If accuracy achieved, exit the loop
    fsub.s ft10, ft11, ft0      # Compute the difference between reference pi and current pi estimate
    fabs.s ft10, ft10           # Get the absolute value of the difference
```

```
    fdiv.s ft10, ft10, ft11      # Normalize the difference by dividing by reference pi value
    flt.s t6, fa0, ft10          # If normalized difference is less than accuracy, stop iteration
    beqz t6, done                # If the condition is met, exit the loop

    # Increment n (iteration index) and alternate the sign for the next term
    fadd.s ft4, ft4, ft1         # Increment n by 1
    fneg.s ft5, ft5              # Alternate the sign (positive <-> negative)

    j loop                       # Repeat the loop for the next iteration

done:
    # Multiply the final accuracy result by 100.0 (ft9) to obtain the computed accuracy
    fmul.s ft10, ft10, ft9
    fmv.s fa2, ft10             # Store the computed accuracy in fa2
    fmv.s fa1, ft0              # Store the final computed value of pi in fa1
    ret                         # Return from the subroutine
```

4.4 Тестирование (test.s)

В тестах программа запускает три заранее заданных теста с известными значениями точности и ожидаемого числа π. Для каждого теста вычисляется π, затем выводятся как ожидаемые, так и вычисленные значения, а также погрешность.

```
.include "io_macros.s"      # Include macros for input/output operations

.data
    prompt_exp_pi:          .asciz "Expected pi value: "
    prompt_exp_acur:        .asciz "Expected accuracy (%): "
    prompt_computed_pi:     .asciz "Computed pi value:   "
    prompt_computed_acur:   .asciz "Computed accuracy (%): "
    prompt_test:            .asciz "\n_____TEST_____\n"

    accuracy1:              .float 0.05
    expected1:              .float 3.1427128
    prompt_test1:           .asciz "\nTest 1\n"

    accuracy2:              .float 0.005
    expected2:              .float 3.141736
    prompt_test2:           .asciz "\nTest 2\n"

    accuracy3:              .float 0.00001
    expected3:              .float 3.1415925
    prompt_test3:           .asciz "\nTest 2\n"

.text
.globl test

# Main test subroutine
test:
    print_string prompt_test   # Print test separator
```

```
    #Test 1
    # Load accuracy and call pi computation
    la t0, accuracy1
    flw fa0, 0(t0)
    jal pi_via_nilakantha

    fmv.s ft0, fa1              # Store computed pi
    fmv.s ft1, fa2              # Store computed accuracy

    # Load expected pi and accuracy
    la t0, expected1
    flw ft2, 0(t0)
    la t0, accuracy1
    flw ft3, 0(t0)

    # Print expected and computed results
    print_string prompt_test1
    print_string prompt_exp_pi
    print_float ft2             # Expected pi value
    print_string prompt_exp_acur
    print_float ft3             # Expected accuracy

    print_string prompt_computed_pi
    print_float ft0             # Computed pi value

    print_string prompt_computed_acur
    print_float ft1             # Computed accuracy

    #Test 2
    # Load accuracy and call pi computation
    la t0, accuracy2
    flw fa0, 0(t0)
    jal pi_via_nilakantha

    fmv.s ft0, fa1              # Store computed pi
    fmv.s ft1, fa2              # Store computed accuracy

    # Load expected pi and accuracy
    la t0, expected2
    flw ft2, 0(t0)
    la t0, accuracy2
    flw ft3, 0(t0)

    # Print expected and computed results
    print_string prompt_test2
    print_string prompt_exp_pi
    print_float ft2             # Expected pi value
    print_string prompt_exp_acur
    print_float ft3             # Expected accuracy

    print_string prompt_computed_pi
    print_float ft0             # Computed pi value

    print_string prompt_computed_acur
    print_float ft1             # Computed accuracy

    #Test 3
    # Load accuracy and call pi computation
    la t0, accuracy3
```

```
    flw fa0, 0(t0)
    jal pi_via_nilakantha

    fmv.s ft0, fa1              # Store computed pi
    fmv.s ft1, fa2              # Store computed accuracy

    # Load expected pi and accuracy
    la t0, expected3
    flw ft2, 0(t0)
    la t0, accuracy3
    flw ft3, 0(t0)

    # Print expected and computed results
    print_string prompt_test3
    print_string prompt_exp_pi
    print_float ft2             # Expected pi value
    print_string prompt_exp_acur
    print_float ft3             # Expected accuracy

    print_string prompt_computed_pi
    print_float ft0             # Computed pi value

    print_string prompt_computed_acur
    print_float ft1             # Computed accuracy

    # End the program
    li a7, 10                    # Exit system call
    ecall
```

5 Результаты запусков программы

```
To calculate the value of pi using the Nilakantha series, enter [1]
To run the tests, enter [0]
Your decision: -10
Invalid input! Please try again
Your decision: 10
Invalid input! Please try again
Your decision: 1
To calculate the value of pi using the Nilakantha series, enter [1]
To run the tests, enter [0]

Input Enter the percentage accuracy (0.000001 <= a <= 0.05): 0.1
Invalid input! Please try again

Input Enter the percentage accuracy (0.000001 <= a <= 0.05): 0.0000000001
Invalid input! Please try again

Input Enter the percentage accuracy (0.000001 <= a <= 0.05): 0.0005

Result of computing pi: 3.1416073
Accuracy (%): 4.6293504E-4

-- program is finished running (0) --

Reset: reset completed.


```

```
To calculate the value of pi using the Nilakantha series, enter [1]
To run the tests, enter [0]
Your decision: 0

_____TEST_____

Test  1
Expected pi value: 3.1427128
Expected accuracy (%): 0.05
Computed pi value:  3.1427128
Computed accuracy (%): 0.035653587

Test 2
Expected pi value: 3.141736
Expected accuracy (%): 0.005
Computed pi value:  3.141736
Computed accuracy (%): 0.0045610485

Test 3
Expected pi value: 3.1415925
Expected accuracy (%): 1.0E-5
Computed pi value:  3.1415925
Computed accuracy (%): 7.589099E-6

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

6 Аналогичные результаты на C++

Аналогичные результаты можно получить, запустив программу, реализующую вычисление числа π с использованием ряда Нилаканта на языке C++. Ниже приведен фрагмент программы на C++, которая вычисляет число π с заданной точностью и сравнивает результат с известным значением:

```cpp
#include <iostream>
#include <cmath>   // For fabs() function

// Reference value of Pi
float reference = 3.141592653589793;

// Function to compute Pi using Nilakantha series
std::pair<float, float> computePi(float accuracy) {
    accuracy /= 100;  // Convert accuracy from percentage to decimal
    float pi = 3.0f;  // Initial value of pi
    int i = 1;         // Index for the series
    int sign = 1;      // Variable to alternate signs (+ or -)
    float current_accuracy = (fabs(reference - pi) / reference);  // Initial accuracy

    // Loop until the computed value is within the desired accuracy
    while (current_accuracy > accuracy) {

        // Compute the next term in the Nilakantha series
        float term = 4.0f / (2 * i * (2 * i + 1) * (2 * i + 2));

        // Add or subtract the term based on the current sign
        if (sign == 1) {
            pi += term;  // Add term if sign is positive
        } else {
            pi -= term;  // Subtract term if sign is negative
```

```cpp
        }

        // Increment the index and alternate the sign
        i++;
        sign = -sign;  // Alternate the sign between positive and negative

        // Recalculate the current accuracy
        current_accuracy = (fabs(reference - pi) / reference);
    }

    // Return computed Pi value and accuracy as a pair
    return std::make_pair(pi, current_accuracy * 100);
}

int main() {
    // Test 1: accuracy = 0.05%, expected Pi value ~ 3.1427128
    float accuracy1 = 0.05;
    float expected1 = 3.1427128;

    // Test 2: accuracy = 0.005%, expected Pi value ~ 3.141736
    float accuracy2 = 0.005;
    float expected2 = 3.141736;

    // Test 3: accuracy = 0.00001%, expected Pi value ~ 3.1415925
    float accuracy3 = 0.00001;
    float expected3 = 3.1415925;

    // Compute Pi for each test case
    std::pair<float, float> result1 = computePi(accuracy1);
    std::pair<float, float> result2 = computePi(accuracy2);
    std::pair<float, float> result3 = computePi(accuracy3);

    // Output the results for each test case
    std::cout << "Test  1  \n"
              << "Expected pi value: " << expected1 << "\n"
              << "Expected accuracy: " << accuracy1 << "\n"
              << "Computed pi value: " << result1.first << "\n"
              << "Computed accuracy (%): " << result1.second << "\n\n";

    std::cout << "Test  2  \n"
              << "Expected pi value: " << expected2 << "\n"
              << "Expected accuracy: " << accuracy2 << "\n"
              << "Computed pi value: " << result2.first << "\n"
              << "Computed accuracy (%): " << result2.second << "\n\n";

    std::cout << "Test  3  \n"
              << "Expected pi value: " << expected3 << "\n"
              << "Expected accuracy: " << accuracy3 << "\n"
              << "Computed pi value: " << result3.first << "\n"
              << "Computed accuracy (%): " << result3.second << "\n\n";

    return 0;
}
```

6.1 Результаты запуска программы на C++

```
Test 1
Expected pi value: 3.14271
Expected accuracy: 0.05
Computed pi value: 3.14271
Computed accuracy (%): 0.0356536

Test 2
Expected pi value: 3.14174
Expected accuracy: 0.005
Computed pi value: 3.14174
Computed accuracy (%): 0.00456105

Test 3
Expected pi value: 3.14159
Expected accuracy: 1e-05
Computed pi value: 3.14159
Computed accuracy (%): 7.5891e-06
```

7 Заключение

Программа успешно решает задачу вычисления числа π с точностью не хуже 0.05% с использованием ряда Нилаканта. Ввод точности и выбор между вычислением и тестами реализованы через системные вызовы. Подпрограмма для вычисления π корректно реализует серию Нилаканта с необходимыми проверками на точность. Тестирование позволяет проверять корректность вычислений на заранее заданных значениях.

8 Дополнительные замечания

Программа написана с учетом ограничений на точность, поддерживая диапазон точности от 0.000001 до 0.05.

Исходный код можно найти по ссылке