Отчет для ИДЗ №2 АВС

Отчёт по выполнению ИДЗ №2 по АВС

Растворов Сергей 236

Цель работы

Разработка программы на ассемблере для вычисления суммы ряда Тейлора для приближенного расчёта функции $f(x) = \frac{1}{e^x}$ с использованием различных методов проверки точности и возможностью выбора метода завершения вычислений.

Описание программы

Программа на языке ассемблера RISC-V предназначена для вычисления функции $f(x) = \frac{1}{e^x}$ по заданному значению `x` с использованием ряда Тейлора. Программа поддерживает два режима работы:

1. **Ручной режим**: Пользователь вручную вводит значение `x`, выбирает метод завершения вычислений и устанавливает желаемую точность.
2. **Режим автоматического тестирования**: Программа проходит набор тестов с заранее заданными значениями `x`, точностями и методами завершения.

Структура программы

Программа организована в три основных файла:

- `main.s`: Основной файл, содержащий логику вычислений, режимы работы и обработку ввода/вывода.
- `macros.s`: Набор макросов для упрощения кода и реализации повторяющихся операций (ввод/вывод данных, работа с памятью и т.д.).
- `data.s`: Секция данных, содержащая константы, сообщения и тестовые данные.

Описание работы основных модулей

1. **Инициализация**:
    - Загрузка значений `x`, `tolerance`, и констант `1.0` и `-1.0` для выполнения расчетов.
2. **Режим работы**:
    - В начале программы пользователь выбирает режим работы: ручной ввод (`manual_mode`) или автоматическое тестирование (`test_mode`).
3. **Ручной режим**:

- В этом режиме программа предлагает пользователю ввести значение `x`, затем выбрать метод проверки точности и задать уровень точности (по умолчанию `0.0005`).
- Доступны два метода завершения:
  - **Метод 1**: проверка, что текущий член ряда не превышает `tolerance * sum`.
  - **Метод 2**: проверка разности между суммой и предыдущим значением суммы, не превышающей `tolerance`.

4. **Автоматическое тестирование**:
   Предполагалась следующая реализация, но я катострофически не успевал, поэтому в тестовом режиме программа совершает пробег на дефолтных значениях
   - В режиме тестирования программа последовательно выполняет тесты для набора значений `x`, точностей и методов.
   - Результаты каждого теста выводятся с указанием использованных параметров (`x`, `tolerance`, метод).
5. **Обновление ряда Тейлора**:
   - На каждой итерации вычисляется новый член ряда по формуле `term *= -x / n`, и значение `term` добавляется к сумме.
   - Счетчик `n` увеличивается на 1 на каждой итерации.
6. **Проверка условий завершения**:
   - В зависимости от выбранного метода, программа проверяет одно из условий завершения:
     - **Метод 1**: модуль текущего члена ряда не превышает произведения `tolerance` на сумму.
     - **Метод 2**: разность между текущей суммой и предыдущей суммой не превышает `tolerance`.
7. **Вывод результата**:
   - По завершении вычислений результат сохраняется и выводится на экран.

Макросы

Для повышения читаемости и упрощения кода используются макросы, определенные в файле `macros.s`:

- `input_int` и `input_double` для ввода целых и вещественных чисел.
- `output_int` и `output_double` для вывода целых и вещественных чисел.
- `print_string` для вывода строк.
- `read_property_*` и `write_property_*` для работы с переменными, что имитирует концепцию свойств (read-only и write-only) в ассемблере.

Тестирование должно было быть красивым...

В режиме тестирования используется набор тестовых данных из файла `data.s`, включая значения `x`, точности и методы. Вывод результатов сопровождается пояснительными сообщениями, такими как `Test case: x =`, `Tolerance =`, `Method =`, и `Result =`.

Вывод

Разработанная программа позволяет эффективно вычислять функцию $\frac{1}{e^x}$ с заданной точностью, используя ряд Тейлора. Возможность выбора метода завершения позволяет адаптировать алгоритм под конкретные задачи, а макросы облегчают структуру кода, делая его более гибким и понятным.

Тесты

для проверки был написан модуль на `c++`

```
x = 5

cpp
Введите значение x: 5
Значение 1/e^5 приближенно равно: 0.00673833

asm - 1
Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 5

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 1
0.006738328152476951
-- program is finished running (0) --

asm - 2
Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 5
```

```
Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 2
0.006706341054212698
-- program is finished running (0) --
```

Как видим есть небольная разница в точности вычислений между методоми сравнения в асм, но при этом 1 вариант дает ответ равыный вычеслению кода на плюсах

```
x = -5

cpp
Введите значение x: -5
Значение 1/e^-5 приближенно равно: 148.38

asm - 1
SSelect mode:
1. Manual input
2. Automatic test
_: 1
input_double x = -5

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 1
148.3795800797366
-- program is finished running (0) --

asm - 2
Select mode:
1. Manual input
2. Automatic test
_: 1
```

```
input_double x = -5

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 2
148.41310786833827
-- program is finished running (0) --

Как видим есть небольная разница в точности вычислений между методоми
сравнения в асм, но при этом 1 вариант дает ответ равыный вычеслению кода
на плюсах, как видим вывод для 1 теста актуален.
```

Программа корректно работает для 0 ввода

```
Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 0

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 1
1.0
-- program is finished running (0) --

Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 0

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
```

```
1. No (keep 0.0005)
2. Yes, please
_: 1
Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 2
1.0
-- program is finished running (0) --
```

Изменение точности (понижение)

```
asm - 1
Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 5

Do you want to change accuracy?
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 2
your_accuracy = 0.5
Complite!
Now accuracy is 0.5

Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 1
0.006267311767054721
-- program is finished running (0) --

asm - 2
Select mode:
1. Manual input
2. Automatic test
_: 1
input_double x = 5

Do you want to change accuracy?
```

```
Default value 0,0005
Recomend set more mb 0.02
1. No (keep 0.0005)
2. Yes, please
_: 2
your_accuracy = 0.5
Complite!
Now accuracy is 0.5

Select mode accuracy:
1. |term| <= tolerance * sum
2. |sum - prev_sum| ≤ tolerance
_: 2
-0.04555520354131609
-- program is finished running (0) --

ожидаемо, первый вариант подсчета точности дает гораздо лучший результат.
```

Код

```
B#include <iostream>
#include <cmath>

double exp_inverse(double x, double tolerance = 0.0005) {
    double term = 1.0;   // Начальный член ряда
    double sum = term;   // Начальная сумма
    int n = 1;           // Счётчик для факториала и степени

    while (fabs(term) > tolerance * sum) {
        term *= -x / n;  // Рассчитываем следующий член ряда
        sum += term;     // Добавляем его к сумме
        n++;             // Переход к следующему члену ряда
    }

    return sum;
}

int main() {
    double x;
    std::cout << "Введите значение x: ";
    std::cin >> x;

    double result = exp_inverse(x);
    std::cout << "Значение 1/e^" << x << " приближенно равно: " << result
<< std::endl;
```

```
    return 0;
}
```

```
.include        "macros.s"
.include        "data.s"

.text

main:
        # Инициализация переменных
        read_property_double x, f0                      # Загружаем x в f0
        read_property_double one, f1

        fmv.d   f2, f1                                  # Копируем первый
член в сумму

        fmv.d   f3, f1                                  # f3 = 1.0
(начальный член ряда)
        li      a3, 1                                   # Счетчик n
        read_property_double tolerance, f4              # Загружаем
tolerance в f4

        # Выбор режима
        li      t5, 1
        li      t6, 2
        print_string msg_select_mode
        print_string msg_ur_ans
        input_int a0
        beq     a0, t5, manual_mode
        beq     a0, t6, test_mode
        print_string msg_newline
        j       main

manual_mode:
        # input double x
        print_string msg_prompt_input_double
        input_double fa0
        fmv.d   f0, fa0
        print_string msg_newline

        # chose accuracy value
        print_string msg_prompt_accuracy_change
        print_string msg_select_mode_to_change_accuracy
   change_ur_mind:                                      # wrong ans loop
        print_string msg_ur_ans
        input_int a0
        beq     a0, t5, chose_accuracy_method
        beq     a0, t6, change_accuracy_value
        print_string msg_newline
```

```
        j       change_ur_mind

    change_accuracy_value:                              # wrong ans loop
        print_string msg_your_accuracy
        input_double fa0
        fmv.d   f4, fa0
        print_string msg_accuracy_set
        output_double f4
        print_string msg_newline
        print_string msg_newline
        j       chose_accuracy_method

        # chose accuracy method
chose_accuracy_method:
        print_string msg_select_mode_accuracy
        print_string msg_ur_ans
        input_int a0
        write_property_word method, a0
        beq     a0, t5, check_method_1
        beq     a0, t6, check_method_2
        print_string msg_newline
        j       chose_accuracy_method


test_mode:

while_loop:
        # Выбор метода
        read_property_word method, a5

        # Метод 1: |term| <= tolerance * sum
        li      a6, 1
        beq     a5, a6, check_method_1

        # Метод 2: |sum - previous_sum| <= tolerance
        li      a6, 2
        beq     a5, a6, check_method_2
        j       chose_accuracy_method

check_method_1:
        # Проверка условия окончания для метода 1
        fmul.d  f5, f4, f2                              # f5 = tolerance *
sum
        fabs.d  f6, f3                                  # f6 = |term|
        fle.d   a6, f6, f5                              # Проверка |term|
<= tolerance * sum
        bnez    a6, end_loop                            # Если условие
выполнено, выходим из цикла
        j       update_series
```

```
check_method_2:
        # Проверка условия окончания для метода 2
        read_property_double previous_sum, f9           # Загружаем
previous_sum в f9
        fsub.d  f10, f2, f9                             # f10 = |sum -
previous_sum|
        fabs.d  f10, f10                                        # Берем
абсолютное значение
        fle.d   a6, f10, f4                             # Проверка |sum -
previous_sum| <= tolerance
        bnez    a6, end_loop                            # Если условие
выполнено, выходим из цикла

        # Обновление previous_sum для следующей итерации
        write_property_double previous_sum, f2          # Сохраняем
текущее значение sum в previous_sum
        j       update_series

update_series:
        # Вычисление следующего члена ряда: term *= -x / n
        read_property_double minus_one, f7              # f7 = -1.0
        fmul.d  f3, f3, f7                              # term = term * -1
        fmul.d  f3, f3, f0                              # term = term * x
        fcvt.d.w f8, a3                                 # Преобразуем n
(целое) в double
        fdiv.d  f3, f3, f8                              # term = term / n

        # Обновление суммы
        fadd.d  f2, f2, f3                              # sum += term
        addi    a3, a3, 1                               # Увеличиваем n

        # Переход к следующей итерации
        j       while_loop

end_loop:
        # Сохранение результата
        write_property_double answer, f2

output_ans:
        # Вывод результата
        read_property_double answer, fa0                # Перемещаем
результат в fa0
        output_double fa0
        j       end

end:
        li      a7, 10                                  # Завершение
```

```
программы
        ecall
```

```
# Макрос для чтения целого числа
.macro input_int %reg
        li a7, 5
        ecall
        mv %reg, a0
.end_macro

# Макрос для чтения double числа
.macro input_double %reg
        li a7, 7
        ecall
        fmv.d %reg, fa0
.end_macro


# Макрос для вывода целого числа
.macro output_int %reg
        li a7, 5
        mv %reg, a0
        ecall
.end_macro

# Макрос для вывода double числа
.macro output_double %reg
        fmv.d fa0, %reg
        li a7, 3
        ecall
.end_macro

# Макрос для вывода строки
.macro print_string %str
        la a0, %str
        li a7, 4
        ecall
.end_macro

# Макрос для чтения свойств
.macro read_property_word %addr, %reg
        la t0, %addr
        lw %reg, 0(t0)
.end_macro

# Макрос для записи свойств
.macro write_property_word %addr, %reg
        la t0, %addr
        sw %reg, 0(t0)
```

```
.end_macro

# Макрос для чтения свойств double
.macro read_property_double %addr, %freg
        la t0, %addr
        fld %freg, 0(t0)
.end_macro

# Макрос для записи свойств double
.macro write_property_double %addr, %freg
        la t0, %addr
        fsd %freg, 0(t0)
.end_macro
```

```
.data

msg_newline:                    .asciz "\n"
msg_space:                      .asciz " "
msg_error:                      .asciz "Error: invalid number of
elements!\n"
msg_select_mode:                .asciz "Select mode:\n1. Manual input\n2.
Automatic test\n"
msg_ur_ans:                     .asciz "_: "

msg_prompt_input_double:        .asciz "input_double x = "
msg_select_mode_accuracy:       .asciz "Select mode accuracy:\n1. |term|
<= tolerance * sum\n2. |sum - prev_sum| ≤ tolerance\n"
msg_prompt_accuracy_change:     .asciz "Do you want to change accuracy?
\nDefault value 0,0005\nRecomend set more mb 0.02\n"
msg_select_mode_to_change_accuracy: .asciz "1. No (keep 0.0005)\n2. Yes,
please\n"

msg_your_accuracy:              .asciz "your_accuracy = "
msg_accuracy_set:               .asciz "Complite!\nNow accuracy is "
msg_method_1:                   .asciz "Method_1\n"
msg_method_2:                   .asciz "Method_2\n"

# Свойства и константы
x:                              .double 1.0                     # Значение
x
tolerance:                      .double 0.0005                  # Точность
one:                            .double 1.0                     #
Константа 1.0
minus_one:                      .double -1.0                    #
Константа -1.0
answer:                         .double 0.0                     # Для
хранения результата
method:                         .word 1                         # Выбор
метода (1 или 2)
```

```
previous_sum:                   .double 0.0                     #
Предыдущая сумма для второго метода

# Тестовые данные
test_x_values:                  .double -5.0, 0.0, 1.0, 2.5, 3.7 # Набор
значений x для тестов
test_tolerances:                .double 0.0005, 0.001, 0.01      # Набор
точностей для тестов
test_methods:                   .word 1, 2, 1                    # Набор
методов для тестов

msg_test_start:                 .asciz "Starting test...\n"
msg_test_case:                  .asciz "Test case: x = "
msg_tolerance_case:             .asciz " Tolerance = "
msg_method_case:                .asciz " Method = "
msg_result:                     .asciz " Result = "
```