Архитектура компьютера и операционные системы ИДЗ №2

Мищенко Александр. БПИ248-1

Вариант 15

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, вычисляющую с помощью степенного ряда с точностью не хуже $0,05\%$ значение функции гиперболического косинуса $ch(x) = (e^x + e^{-x})/2$ для заданного параметра x

Отчет

Метод решения

Необходимо вычислить значение гиперболического косинуса для заданного аргумента x с относительной погрешностью не хуже 0.05%.

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Однако для реализации на ассемблере без использования встроенных экспоненциальных функций удобно воспользоваться разложением в степенной ряд Маклорена:

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$$

Каждый следующий член ряда выражается через предыдущий:

$$\text{term}_{n+1} = \text{term}_n \cdot \frac{x^2}{(2n+1)(2n+2)}$$

Таким образом, можно организовать итеративный процесс, не вычисляя факториалы напрямую.

Алгоритм вычисления

1. Ввести значение аргумента x.
2. Инициализировать:
$$\text{sum} = 1, \quad \text{term} = 1, \quad n = 1$$
3. На каждой итерации:
$$\text{term} \leftarrow \text{term} \cdot \frac{x^2}{(2n)(2n-1)}$$
$$\text{sum} \leftarrow \text{sum} + \text{term}$$
4. Проверить критерий остановки:
$$\left|\frac{\text{term}}{\text{sum}}\right| < \varepsilon$$
где $\varepsilon = 0.0005$ (что соответствует относительной точности 0.05%).
5. При выполнении условия точности завершить цикл.
6. Вывести результат $\cosh(x) \approx \text{sum}$.

Примечания

- Для хранения промежуточных значений используются регистры с плавающей точкой (`f`-регистры).
- Переменная `term` хранит текущий член ряда, а `sum` — накопленную сумму.
- На каждой итерации множитель $\frac{x^2}{(2n)(2n-1)}$ вычисляется через целые регистры, с последующим приведением к типу `double`.
- Цикл ограничен максимальным числом итераций (например, 50) для предотвращения зацикливания.
- После завершения расчёта результат помещается в регистр `f12` и выводится на экран.

Ссылка на источник с описанием метода вычисления гиперболического косинуса через разложение в ряд Тейлора

Тестирование и сравнение с высокоуровневым языком

В ходе работы была проведена проверка корректности реализации функции вычисления гиперболического косинуса

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

в нашей программе, выполненной на архитектуре RISC-V.
Для тестирования использовались входные данные:

$$x = [0,\ 1,\ 2,\ 3,\ 0.2,\ 5.6,\ -0.5].$$

```
Введите x: 0
ch(x)≈ 1.0
-- program is finished running (0) --
```

```
Введите x: 1
ch(x)≈ 1.5430803571428573
-- program is finished running (0) --

Введите x: 2
ch(x)≈ 3.7621869488536155
-- program is finished running (0) --

Введите x: 3
ch(x)≈ 10.067605012175324
-- program is finished running (0) --
```

```
Введите x: 0.2
ch(x)≈ 1.0200666666666667
-- program is finished running (0) --


Reset: reset completed.

Введите x: 5.6
ch(x)≈ 135.21006544381407
-- program is finished running (0) --
```

```
Введите x: -0.5
ch(x)≈ 1.1276258680555555
-- program is finished running (0) --
```

Аналогичные тесты можно провести самостоятельно запустив файл `tests.asm`.

Полученные из нашей программы результаты были вручную перенесены в программу на Python, где те же значения x были обработаны с использованием стандартной библиотеки `math` и выражения `(math.exp(x) + math.exp(-x)) / 2`.
Для каждого тестового значения было вычислено относительное отклонение между результатами RISC-V и Python по формуле:

$$\delta = \frac{|y_{riscv} - y_{python}|}{\max(y_{riscv}, y_{python})} \times 100\%.$$

Анализ показал, что реализация функции на RISC-V даёт результаты, совпадающие с эталонными вычислениями Python с точностью до 0.05%. Погрешность остаётся в допустимых пределах как для положительных, так и для отрицательных, целых и

вещественных значений аргумента.
Таким образом, можно заключить, что реализация функции ch(x) на архитектуре RISC-V является корректной.

[19]
✓ 0 сек.

```python
import math

x_list = [0, 1, 2, 3, 0.2, 5.6, -0.5]
riscv_results = [1.0, 1.5430803571428573, 3.7621869488536155, 10.067605012175324,
                 1.0200666666666667, 135.21006544381407, 1.1276258680555555]

print(f"{'x':>6} | {'Python Math':>20} | {'RISC-V':>20} | {'Diff (%)':>10}")
print("-" * 65)

for i in range(len(x_list)):
    result = (math.exp(x_list[i]) + math.exp(-x_list[i])) / 2
    diff = abs(result - riscv_results[i]) / max(result, riscv_results[i]) * 100
    print(f"{x_list[i]:6} | {result:20.15f} | {riscv_results[i]:20.15f} | {diff:10.6f}")
```

```
     x |          Python Math |               RISC-V |   Diff (%)
-----------------------------------------------------------------
     0 |    1.000000000000000 |    1.000000000000000 |   0.000000
     1 |    1.543080634815244 |    1.543080357142857 |   0.000018
     2 |    3.762195691083631 |    3.762186948853615 |   0.000232
     3 |   10.067661995777765 |   10.067605012175324 |   0.000566
   0.2 |    1.020066755619076 |    1.020066666666667 |   0.000009
   5.6 |  135.215052644934502 |  135.210065443814074 |   0.003688
  -0.5 |    1.127625965206381 |    1.127625868055556 |   0.000009
```