АВС: ИДЗ-2

Ноябрь 2024

Гобец Иван Евгеньевич. БПИ 237. Вариант 13

Условие. Вариант 13

Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже $0,1\%$ значение функции e^x для заданного параметра x.

Отчет на оценку 6-7

Отчет сразу начинает с оценки на 6-7, а не на 4-5, так как, спросив у семинариста, он сказал, что можно так сделать.

- Далее буду предоставлять блоки по различным частям кода (для понятности, чтобы не было все подряд).
- Ещё я описывал, все строчки кода комментариями для более понятного пояснения в отчете.

Блок .data

```
.data
        start_information:      .asciz "Welcome. This work is done by Gobets Ivan group BPI-237. Option - 13"
        get_x_string:           .asciz "Enter value of x: "
        new_line:               .asciz "\n"

        epsilon:                .double 0.001   # Значение epsilon для точности
```

Блок с макросами

- **read_int(%x)** - Читает целое число из ввода и сохраняет его в указанный регистр. Параметры: %x - регистр для хранения результата.
- **print_double(%x)** - Выводит число типа double, хранящееся в указанном регистре. Параметры: %x - регистр, содержащий число для вывода.
- **print_string(%x)** - Выводит строку по метке. Параметры: %x - метка, содержащая адрес строки.
- **end_program** - Завершает выполнение программы. Параметры: Нет.

Постарался больше пояснений оставить в комментариях, чтобы было нагляднее и легече читать код.

```
# Макрос для чтения числа типа int
.macro  read_int(%x)                    # Определение макроса read_int, принимающего один параметр %x (регистр для хранения результата).
        li, a7, 5                       # Загружаем код системного вызова 5 (read_int) в регистр a7 для ввода целого числа.
        ecall                           # Выполняем системный вызов.
        mv %x, a0                       # Копируем полученное значение из регистра a0 в указанный регистр %x.
.end_macro

# Макрос для вывода double
.macro print_double(%x)                 # Определение макроса print_double, принимающего регистр %x (число для вывода).
        li, a7, 3                       # Загружаем код системного вызова 3 (print_double) в регистр a7 для вывода числа.
        fmv.d fa0, %x                   # Копируем значение из %x в регистр fa0 для передачи в системный вызов.
        ecall                           # Выполняем системный вызов.
.end_macro

# Макрос для вывода строки
.macro print_string(%x)                 # Определение макроса print_string, принимающего метку %x (адрес строки).
        la, a0, %x                      # Загружаем адрес строки (метки) %x в регистр a0.
        li, a7, 4                       # Загружаем код системного вызова 4 (print_string) в регистр a7 для вывода строки.
        ecall                           # Выполняем системный вызов.
.end_macro

# Макрос для завершения программы
.macro end_program                      # Определение макроса end_program без параметров.
        li, a7, 10                      # Загружаем код системного вызова 10 (exit) в регистр a7 для завершения программы.
        ecall                           # Выполняем системный вызов.
.end_macro
```

Блок main

- Выводится приветствие(“Welcome. This work is done by Gobets Ivan group BPI-237. Option - 13”) и перенос строки(new_line) через макрос **print_string**
- Выводим “Enter value of x: “
- Запрашиваем int с помощью макроса read_int и сохоаняем в регистре t0
- Конвертируем в double в регист ft0 получнное значение от пользователя
- Вызываем функцию **calculate_e** и передаем в нее значение
- Далее прыгаем в подпрограмму check_minus, где t0 - значение от пользователя в формате int, ft0 - тоже значение, но в формате double

```
.text
main:
        print_string(start_information)             # Выводим стартовую информацию
        print_string(new_line)                      # Выводим новую строку
        print_string(get_x_string)                  # Выводим строку для запроса x

        read_int(t0)                                # Запрашиваем у пользователя значение x
        fcvt.d.w        ft0,    t0                  # Конвертим int в double

        # t0 - значение x
        j               check_minus                 # Делаем джамп в подпрограмму определения знака
```

Блок check_minus

Блок в котором мы опредлеям какой знак введого пользователем число.

- В t4 храним 0, если число положительное, и 1, если отрицательное
- Если введедное число положительное то сразу переходим в блок с вычеслениями, если отрицательное, то:
  - В t4 теперь храним 0, а не 1
  - Далее меняем знак у числа на положительный
  - Теперь загружаем в ft0 значение числа
- Прыгаем в блок с вычеслениями

```
47  check_minus:
48          li              t4,     0                       # Загружаем в t4 0, т.е. flag = false – не отрицательное число
49          bgez            t0,     declaration_variables   # Если число больше или равно 0 то сразу переходим к расчетам
50
51          li              t4,     1                       # Загружаем в t4 1, т.е. flag = true – число отрицательно
52          neg             t0,     t0                      # Делаем число положительным
53          fcvt.d.w        ft0,    t0                      # Загружаес в ft0 положительное число
54
55          # ft0 – значение икса
56          j               declaration_variables           # Делаем джамп в подпрограмму с расчетами
```

Объявление переменнных

- Регистр t1 используется как факториал для знаменателя (в цикле увеличивается на 1 на каждом шаге)
- Регистр ft2 хранит значение x (параметр ряда e^x)
- Регистр ft3 хранит текущий член ряда (начиная с первого члена)
- Регистр ft6 хранит накопленную сумму ряда (сумма всех предыдущих членов)
- Регистр ft4 хранит произведение текущего члена ряда на x (для вычисления следующего члена)
- Регистр ft5 хранит факториал, который будет делиться на текущий член ряда
- Регистр ft9 используется для вычисления значения (epsilon × предыдущий член)
- Регистр ft8 хранит предыдущий член ряда для проверки на точность (epsilon)

```
58  declaration_variables:
59          li              a1,     1                       # Устанавливаем a1 в 1 (для использования в расчетах)
60          li              t1,     1                       # Устанавливаем t1 в 1 (для знаменателя)
61
62          fcvt.d.w        ft1,    a1                      # Начальная сумма
63          fmv.d           ft2,    ft0                     # Значение x
64          fmv.d           ft3,    ft1                     # Начальный член ряда
65          fmv.d           ft6,    ft1
66
67          la              t3,     epsilon                 # Загружаем адрес переменной epsilon
68          fld             ft7,    (t3)                    # Загружаем значение epsilon (0.001) в регистр ft7
69          fcvt.d.w        ft8,    t1                      # Преобразуем t1 в double и сохраняем в ft8
70
71          j               calculate_e_loop
```

Блок вычислений e^x

Значем, что $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

Важное замечание: на кождом шаге вместо того, чтобы вычислять факториал (мы там еще будем получать переполниение, такой способ неэффективен), мы будем просто умножать предыдущий член на x и делить n и прибавлять к сумме, такой способ более оптимальный.
В какой момент нужно остановиться ? На каждой итерации мы храним предыдущий член и текущий. Мы будем умножать предыдщий член на епсилон 0.001 и смотреть, если он меньше следующего, то завершаем наш цикл.
Выше я объяснил локигу/метод расчетов, для пояснения кода я подробно оставил комментрии к каждой строчке.

```
# Регистр t1 используется как факториал для знаменателя (в цикле увеличивается на 1 на каждом шаге)
# Регистр ft2 хранит значение x (параметр ряда e^x)
# Регистр ft3 хранит текущий член ряда (начиная с первого члена)
# Регистр ft6 хранит накопленную сумму ряда (сумма всех предыдущих членов)
# Регистр ft4 хранит произведение текущего члена ряда на x (для вычисления следующего члена)
# Регистр ft5 хранит факториал, который будет делиться на текущий член ряда
# Регистр ft9 используется для вычисления значения (epsilon * предыдущий член)
# Регистр ft8 хранит предыдущий член ряда для проверки на точность (epsilon)
calculate_e_loop:
        fmul.d          ft4,    ft3,    ft2             # Умножаем текущий член на x (ft3 = ft3 * ft2)

        fcvt.d.w        ft5,    t1                      # Преобразуем t1 в double для знаменателя
        fdiv.d          ft3,    ft4,    ft5             # Делим текущий член на t1 (ft3 = ft4 / ft5)

        fadd.d          ft6,    ft6,    ft3             # Добавляем текущий член к сумме (ft6 = ft6 + ft3)

        addi            t1,     t1,     1               # Увеличиваем t1 на 1 (для следующего шага)

        fmul.d          ft9,    ft8,    ft7             # Умножаем предыдущий шаг на epsilon
        fmv.d           ft8,    ft3                     # Сохраняем текущий член в ft8

        flt.d           t2,     ft9,    ft3             # Сравниваем значение epsilon с текущим членом
        bnez            t2,     calculate_e_loop        # Если текущий член не меньше epsilon, продолжаем цикл

        # ft6 - результат
        j               print_result
```

Вывод результата и завершение программы

В t4 у нас хранится знак исходного числа от пользователя.

- Если число было положительным, то прыгаем в подпрограмму завершения программы.
- Если число отрицательное, то прыгаем в подпрограмму print_negative_result

- В подпрограмме print_negative_result мы в t1 кладем единичку, потом делим ее на результат вычислений и прыгаем в подпрограмму завершения программы. Делим мы единицу на наш результат, потому что мы считали для положительно икса, теперь нам нужно поделить единицу на результат, чтобы получить результат для отрицательного икса.

- Блок завершения программы, где выводим результат и завершаем программу.

```
print_result:
        beqz            t4,     end_loop                # Если t4 == 0 (число было не отрицательным), делаем джамп в завершение программы

        # ft6 - рузультат (нужно еще единцу поделить на это значение, т.к. мы не учитывали минус)
        j               print_negative_result

print_negative_result:
        li              t1,     1                       # Устанавливаем t1 в 1
        fcvt.d.w        ft10,   t1                      # Преобразуем t1 в double и сохраняем в ft10
        fdiv.d          ft6,    ft10,   ft6             # Делим 1 на результат (ft6 = 1 / ft6)

        j               end_loop

end_loop:
        print_double(ft6)                               # Выводим результат
        end_program()                                   # Используем макрос для завершения программы
```

Тестовое покрытие

- Тест при x = 0

```
Enter value of x: 0
1.0
```

- Тест при x = 10

```
Enter value of x: 10
22026.46579480671
```

- Тест при x = −10

```
Enter value of x: -10
4.539992976248486E-5
```

- Тест при x = 100

```
Enter value of x: 100
2.6881171418161336E43
```

- Тест при x = −100

```
Enter value of x: -100
3.7200759760208386E-44
```

- Тест при x = 1000

```
Enter value of x: 1000
Infinity
```

- Тест при x = −1000

```
Enter value of x: -1000
0.0
```

Отчет на 8

- В блок .data добавим строки для более красивого вывода для пользователя (еще было изменено “Test result” на “Test value”).

```
.data
        start_information:      .asciz "Welcome. This work is done by Gobets Ivan group BPI-237. Option - 13"
        tested_cases:           .asciz "Enter 0 if you want to run automatic tests, 1 for manual input: "
        get_x_string:           .asciz "Enter value of x: "
        value:                  .asciz "Entered value: "
        test_result:            .asciz "Test result: "
        result:                 .asciz "Value e^x: "
        new_line:               .asciz "\n"
        new_double_line:        .asciz "\n\n"

        epsilon:                .double 0.001   # Значение epsilon для точности
```

- Также был добвален макрос для вывода int

```
# Макрос для вывода целого числа
.macro print_int(%x)                    # Определение макроса print_int, принимающего регистр %x (число для вывода).
        li, a7, 1                       # Загружаем код системного вызова 1 (print_int) в регистр a7 для вывода числа.
        mv a0, %x                       # Копируем значение из %x в регистр a0 для передачи в системный вызов.
        ecall                           # Выполняем системный вызов.
.end_macro
```

- Изменили блок main, теперь мы запрашиваем у пользователя ввод для автотестов, если пользователь ввел 0, то начнутся автотесты, а если нет, то программа будет работать как обычно (ручной ввод).
- Также мы теперь используем jal, чтобы запомнить адрес откуда мы прыгаем, чтобы вернуться и закончить прграмму без подпрограммы.

```
.text
main:
        print_string(start_information)                 # Выводим стартовую информацию
        print_string(new_line)                          # Выводим новую строку

        print_string(tested_cases)                      # Выводим строку для запроса автотестов
        read_int(t3)                                    # В t1 запрашиваем для автотестов

        beqz            t3,     tests

        print_string(get_x_string)                      # Выводим строку для запроса x

        read_int(t0)                                    # Запрашиваем у пользователя значение x
        fcvt.d.w        ft0,    t0                      # Конвертим int в double

        # t0 - значение x
        jal             check_minus                     # Делаем джамп в подпрограмму определения знака

        end_program()
```

- Поменяли локику вывода рузультата, теперь если в t4 лежит 0, т.е. число положительное, то мы пгрыгаем в подпрограмму для вывода положительного, а если отрицательное, то в подпрограмму для вывода отрицательного (логика расчетов не изменилась).

Потом возвращаемся в main по адресу.

```
print_result:
        beqz            t4,     print_positive_result   # Если t4 == 0 (число было не отрицательным), делаем джамп в вывод положительного рузультата

        # ft6 - рузультат (нужно еще единцу поделить на это значение, т.к. мы не учитывали минус)
        j               print_negative_result

print_positive_result:
        print_string(result)                            # Выводим строку для рузультата
        print_double(ft6)                               # Выводим результат

        jalr            ra                              # Возвращаемся по адрессу

print_negative_result:
        li              t1,     1                       # Устанавливаем t1 в 1
        fcvt.d.w        ft10,   t1                      # Преобразуем t1 в double и сохраняем в ft10
        fdiv.d          ft6,    ft10,   ft6             # Делим 1 на результат (ft6 = 1 / ft6)

        print_string(result)                            # Выводим строку для рузультата
        print_double(ft6)                               # Выводим результат

        jalr            ra                              # Возвращаемся по адрессу
```

Блок тестов. Тестируемые значения: 0, 10, −10, 100, −100, 1000, −1000

- В кажом блоке мы сначало в t1 загружаем значние нашего теста. Потом выводим строку “Test result: “ и потом печатаем значение теста, и переходим на новую строку.
- Далее конвертируем значние теста в double и прыгаем в подпрограмму check_minus (далее там происходят все расчеты) с запоменанием адреса (jal). После вычислений мы вернемся в тест и пойдем дальше по ним.
- Далее повторяем так происходит на всех тестах.
- Еще одно нововведение это вывод переноса два раза (new_double_line) для более красивого вывода.
- После всех тестов программа завершается с помощью макроса end_program.

```
tests:
        # Тест при x = 0
        test_1:
                        li              t0,     0       # Загружаем x = 0 в регистр t0

                        print_string(test_result)       # Печатаем строку "Test result: "
                        print_int(t0)                   # Печатаем значение x (0)
                        print_string(new_line)          # Печатаем новую строку

                        fcvt.d.w        ft0,    t0      # Преобразуем значение x (t0) в тип double и сохраняем в ft0
                        jal             check_minus     # Переходим к подпрограмме для обработки знака числа

        # Тест при x = 10
        test_2:
                        li              t0,     10

                        print_string(new_double_line)
                        print_string(test_result)
                        print_int(t0)
                        print_string(new_line)


                        fcvt.d.w        ft0,    t0
                        jal             check_minus

        # Тест при x = -10
        test_3:
                        li              t0,     -10

                        print_string(new_double_line)
                        print_string(test_result)
                        print_int(t0)
                        print_string(new_line)

                        fcvt.d.w        ft0,    t0
                        jal             check_minus
```

```
179         # Тест при x = 100
180         test_4:
181                         li              t0,     100
182
183                         print_string(new_double_line)
184                         print_string(test_result)
185                         print_int(t0)
186                         print_string(new_line)
187
188                         fcvt.d.w        ft0,    t0
189                         jal             check_minus
190
191         # Тест при x = -100
192         test_5:
193                         li              t0,     -100
194
195                         print_string(new_double_line)
196                         print_string(test_result)
197                         print_int(t0)
198                         print_string(new_line)
199
200                         fcvt.d.w        ft0,    t0
201                         jal             check_minus
202
203         # Тест при x = 1000
204         test_6:
205                         li              t0,     1000
206
207
208                         print_string(new_double_line)
209                         print_string(test_result)
210                         print_int(t0)
211                         print_string(new_line)
212
213                         fcvt.d.w        ft0,    t0
214                         jal             check_minus
215
216         # Тест при x = -1000
217         test_7:
218                         li              t0,     -1000
219
220                         print_string(new_double_line)
221                         print_string(test_result)
222                         print_int(t0)
223                         print_string(new_line)
224
225                         fcvt.d.w        ft0,    t0
226                         jal             check_minus
227
228         end_program()
```

- Пример автоматического тестирования:

```
Test value: 0
Value e^x: 1.0

Test value: 10
Value e^x: 22026.46579480671

Test value: -10
Value e^x: 4.539992976248486E-5

Test value: 100
Value e^x: 2.6881171418161336E43

Test value: -100
Value e^x: 3.7200759760208386E-44

Test value: 1000
Value e^x: Infinity

Test value: -1000
Value e^x: 0.0
```

- Для дополнительной проверки корректности вычислений осуществим аналогичные тестовые прогоны с использованием существующих библиотек на Python.

```python
import math

# Список значений x для тестирования
test_values = [0, 10, -10, 100, -100, 1000, -1000]

for x in test_values:
    try:
        # Рассчитываем e^x с использованием math.exp()
        result = math.exp(x)
    except OverflowError:
        # Обрабатываем случай, если число слишком большое для представления
        if x > 0:
            result = float('inf')  # Возвращаем "бесконечность" для больших значений
        else:
            result = 0.0  # Для очень больших отрицательных значений результат стремится к 0

    # Выводим результаты для каждого теста
    print(f"Test for x = {x}:")
    print(f"e^{x} = {result}\n")

```

- Результаты тестового прогона на Python совпадают с результатами, полученными в ходе работы программы на ассемблере.

```
Test for x = 0:
e^0 = 1.0

Test for x = 10:
e^10 = 22026.465794806718

Test for x = -10:
e^-10 = 4.5399929762484854e-05

Test for x = 100:
e^100 = 2.6881171418161356e+43

Test for x = -100:
e^-100 = 3.720075976020836e-44

Test for x = 1000:
e^1000 = inf

Test for x = -1000:
e^-1000 = 0.0
```

Отчет на 9

Макросы

Оставил много комментариев для более подробного отчета.

- **read_int(%x)** - Читает целое число из ввода и сохраняет его в указанный регистр. Параметры: %x - регистр для хранения результата.
- **print_int(%x)** - Выводит целое число, хранящееся в указанном регистре. Параметры: %x - регистр, содержащий число для вывода.
- **print_double(%x)** - Выводит число типа double, хранящееся в указанном регистре. Параметры: %x - регистр, содержащий число для вывода.

- **print_string(%x)** - Выводит строку по метке. Параметры: %x - метка, содержащая адрес строки.
- **end_program** - Завершает выполнение программы. Параметры: Нет.

```
# Макрос для чтения числа типа int
.macro  read_int(%x)                    # Определение макроса read_int, принимающего один параметр %x (регистр для хранения результата).
        li, a7, 5                       # Загружаем код системного вызова 5 (read_int) в регистр a7 для ввода целого числа.
        ecall                           # Выполняем системный вызов.
        mv %x, a0                       # Копируем полученное значение из регистра a0 в указанный регистр %x.
.end_macro

# Макрос для вывода целого числа
.macro print_int(%x)                    # Определение макроса print_int, принимающего регистр %x (число для вывода).
        li, a7, 1                       # Загружаем код системного вызова 1 (print_int) в регистр a7 для вывода числа.
        mv a0, %x                       # Копируем значение из %x в регистр a0 для передачи в системный вызов.
        ecall                           # Выполняем системный вызов.
.end_macro

# Макрос для вывода double
.macro print_double(%x)                 # Определение макроса print_double, принимающего регистр %x (число для вывода).
        li, a7, 3                       # Загружаем код системного вызова 3 (print_double) в регистр a7 для вывода числа.
        fmv.d fa0, %x                   # Копируем значение из %x в регистр fa0 для передачи в системный вызов.
        ecall                           # Выполняем системный вызов.
.end_macro

# Макрос для вывода строки
.macro print_string(%x)                 # Определение макроса print_string, принимающего метку %x (адрес строки).
        la, a0, %x                      # Загружаем адрес строки (метки) %x в регистр a0.
        li, a7, 4                       # Загружаем код системного вызова 4 (print_string) в регистр a7 для вывода строки.
        ecall                           # Выполняем системный вызов.
.end_macro

# Макрос для завершения программы
.macro end_program                      # Определение макроса end_program без параметров.
        li, a7, 10                      # Загружаем код системного вызова 10 (exit) в регистр a7 для завершения программы.
        ecall                           # Выполняем системный вызов.
.end_macro
```

Отчет на 10

Разбиение программ по файлам:

- **main.asm** - главный файл программы, который содержит точку входа и вызов всех подпрограмм.
- **iomod** - файл, который содержит подпрограммы по вводу данных.
- **tests.asm** - файл, который содержит подпрограммы для автоматического тестирования программы.
- **calculate_e** - файл, который содержит подпрограммы для расчета e^x.

Макросы выдедены в отдельную автономную библиотеку в файле **macrolib.asm**.