

Матвеева Ольга Романовна, БПИ239, Вариант 20

Условие: Разработать программу вычисления числа π с точностью не хуже 0,05% посредством произведения элементов ряда Виета.

Для вычисления числа π по ряду Виета используем следующую формулу:

$$2 \cdot \frac{2}{\sqrt{2}} \cdot \frac{2}{\sqrt{2+\sqrt{2}}} \cdot \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2}}}} \cdot \dots = \pi.$$

$$\pi(n) = 2 \cdot 2^n / a(n)$$

$$a(0) = \sqrt{2}, a(n) = \sqrt{2 + a(n-1)}$$

Текущая погрешность вычисляется как $|\pi(n-1) - \pi(n)|$

Ввод пользователя - число x , точность вычисления числа π , корректные значения которого лежат в диапазоне $(0; 0.0005]$ (0 - полная точность без погрешности).

Останавливаем вычисление π , когда $|\pi(n-1) - \pi(n)| \leq x$

Скриншоты тестовых прогонов:

```
Enter x (double number, accuracy): 0
Error: x must be in range (0; 0.0005]!

-- program is finished running (0) --

Reset: reset completed.

Enter x (double number, accuracy): -1
Error: x must be in range (0; 0.0005]!

-- program is finished running (0) --

Reset: reset completed.

Enter x (double number, accuracy): 0.0006
Error: x must be in range (0; 0.0005]!

-- program is finished running (0) --

Enter x (double number, accuracy): 0.000001
3.141592648614063
-- program is finished running (0) --

Enter x (double number, accuracy): 0.0005
3.141406718496502
-- program is finished running (0) --
```

4–5 баллов:

- Мое решение задачи реализовано на ассемблере. Ввод данных осуществляется с клавиатуры, вывод данных осуществляется на дисплей.
- В коде присутствуют комментарии, поясняющие выполняемые действия.
- Результаты тестовых прогонов с использованием скриншотов приведены.

6–7 баллов:

- Подпрограммы используются с передачей аргументов через соответствующие регистры, определяемые конвенцией по их использованию. К примеру, в моем коде, регистры ft0-ft6 используются функциями для временного хранения локальных переменных, которые не сохраняются. Для работы с числами с плавающей запятой типа double и передачи аргументов в функции используются регистры fa0-fa. Регистр a0 используется для возвращения значения из check_x, регистр fa3 хранит итоговое значение π после работы функции vieta. Нехватки соответствующих регистров нет и сохранение на стеке не требуется.
- Локальные переменные размещаются в свободных регистрах. Их хватает и сохранение на стеке не требуется.
- В местах вызова функций добавлены комментарии, описывающие передачу фактических параметров и перенос возвращаемого результата. Отмечено, в каких регистрах отображаются соответствующие фактические параметры.

```
main:
    jal get_x      # Call get_x to read x; x is stored in fa1 (return value)
    fld fa2, max_x, t0 # Load max_x into fa2 for check_x
    fld fa3, min_x, t0 # Load min_x into fa3 for check_x
    jal check_x     # Call check_x to validate x; returns 1 in a0 if valid, 0 if invalid
    beq a0, zero, end_program # If a0 == 0 (invalid), terminate the program

    # fa0 stores 2
    fld fa4, two, t0
    # fa1 stores x
    # fa2 stores pi[i-1]
    fld fa2, two, t0
    # fa3 stores result
    fld fa3, two, t0
    # fa4 stores a(0)
    fld fa4, min_x, t0

    jal vieta      # Call vieta; returns the calculated pi in fa3
    # Printing result
```

- Информация о проведенных изменениях отображена в отчете наряду с информацией, необходимой на предыдущую оценку.

Код на данном этапе:

```

.data
ask_x: .asciz "Enter x (double number, accuracy): "
error_msg: .asciz "Error: x must be in range (0; 0.0005]!\n"
result_msg: .asciz "pi = "
max_x: .double 0.0005
min_x: .double 0
two: .double 2

.text
main:
    jal get_x      # Call get_x to read x; x is stored in fa1 (return value)
    fld fa2, max_x, t0 # Load max_x into fa2 for check_x
    fld fa3, min_x, t0 # Load min_x into fa3 for check_x
    jal check_x     # Call check_x to validate x; returns 1 in a0 if valid, 0 if invalid
    beq a0, zero, end_program # If a0 == 0 (invalid), terminate the program

    # fa0 stores 2
    fld fa4, two, t0
    # fa1 stores x
    # fa2 stores pi[i-1]
    fld fa2, two, t0
    # fa3 stores result
    fld fa3, two, t0
    # fa4 stores a(0)
    fld fa4, min_x, t0

```

```

    # fa3 stores result

```

```

    fld fa3, two, t0

```

```

    # fa4 stores a(0)

```

```

    fld fa4, min_x, t0

```

```

    jal vieta      # Call vieta; returns the calculated pi in fa3

```

```

    # Printing result

```

```

    li a7, 4

```

```

    la a0, result_msg

```

```

    ecall

```

```

    # fa3 stores the calculated value of pi

```

```

    fmv.d fa0, fa3

```

```

    li a7, 3

```

```

    ecall

```

```

end_program: # Exit the program

```

```

    li a7, 10

```

```

    ecall

```

```

vieta:

```

```

    # Saving return address on the stack

```

```

    addi sp, sp, -4

```

```

    sw ra, 0(sp)

```

```

vieta_cycle:
    #  $a(i) = \sqrt{2 + a(i-1)}$ 
    fadd.d ft0, fa4, fa0 #  $ft0 = 2 + a(i-1)$ 
    fsqrt.d fa4, ft0     #  $a(i) = \sqrt{2 + a(i-1)}$ 
    #  $pi(i) = 2 / a(n)$ 
    fdiv.d ft3, fa0, fa4

    # Check error:  $|pi(i-1) - pi(i)|$ 
    fsub.d ft0, fa2, ft3 #  $ft0 = pi(i-1) - pi(i)$ 
    fabs.d ft0, ft0      #  $|pi(i-1) - pi(i)|$ 
    fle.d t2, ft0, fa1   # Check  $|pi(i-1) - pi(i)| \leq x$ 
    bnez t2, end_vieta   # If condition is met, exit

    # Update  $\pi(n)$ 
    fmul.d fa3, fa3, ft3

    # Update  $pi(i - 1)$ 
    fmv.d fa2, ft3

j vieta_cycle

end_vieta:
    # Return the result in fa3
    fmv.d fa3, ft3
    # Read return address from the stack

end_vieta:
    # Return the result in fa3
    fmv.d fa3, ft3
    # Read return address from the stack
    lw ra, 0(sp)
    addi sp, sp, 4
    ret

get_x:
    # Ask for x
    li a7, 4
    la a0, ask_x
    ecall

    # Get input number
    li a7, 7
    ecall
    # Save x in fa1 (return value)
    fmv.d fa1, fa0
    ret

check_x:
    mv t0, zero
    mv t1, zero
    li t3, 1
    # Check if x is in the range (0; 0.0005]

```

```

    li a7, 7
    ecall
    # Save x in fa1 (return value)
    fmv.d fa1, fa0
    ret

check_x:
    mv t0, zero
    mv t1, zero
    li t3, 1
    # Check if x is in the range (0; 0.0005]
    fle.d t0, fa1, fa3 # Check: x <= 0
    fgt.d t1, fa1, fa2 # Check: x > 0.0005
    beq t0, t3, wrong_x_branch # If x <= 0, branch to error
    beq t1, t3, wrong_x_branch # If x > 0.0005, branch to error
    li a0, 1 # If all checks pass, set a0 to 1 (valid)
    ret # Return from check_x

wrong_x_branch:
    # Print error message
    li a7, 4
    la a0, error_msg
    ecall
    li a0, 0 # Set a0 to 0 (invalid)
    ret

```

8 баллов:

- Разработанные подпрограммы поддерживают многократное использование с различными наборами исходных данных, включая возможность обработки в качестве параметров различных исходных данных.
- Реализовано автоматизированное тестирование за счет создания программы, осуществляющей прогон различных тестовых данных (вместо их ввода).

Скриншоты программы:

```
.data
ask_x: .asciz "Enter x (double number, accuracy): "
error_msg: .asciz "Error: x must be in range (0; 0.0005]!\n"
result_msg: .asciz "expected: %lf, result: %lf\n"
max_x: .double 0.0005
min_x: .double 0
two: .double 2
.align 3
tests: .double 0.0, -1.0, 0.0006, 0.000001, 0.0005
.align 3
answers: .double -1.0, -1.0, -1.0, 3.141592648614063, 3.141406718496502
num_tests: .word 5

.text
main:
    la t0, tests          # Load the address of the tests array
    la t1, answers        # Load the address of the answers array
    lw t2, num_tests       # Load the number of tests
    li t5, 0              # Initialize the index for iteration

test_loop:
    fld fa1, 0(t0)         # Load x into fa1 from tests array
    fld fa4, 0(t1)         # Load expected answer into fa4 from answers array
    jal check_x            # Check x
    beq a0, zero, error_case # If x is invalid, branch to error

    # Call function to compute pi
    fld fa2, two           # Load constant 2 into fa2
    fmv.d fa0, fa3         # Initialize fa0 to 0.0 for pi[n]
    jal vieta              # Call function vieta, result will be in fa3
```

```

# Print result
li a7, 4          # Syscall code for write
la a0, result_msg # Load message for output
ecall

fmv.d fa0, fa1    # Prepare x for output
li a7, 3          # Syscall code for printing floating point number
ecall

fmv.d fa0, fa3    # Load the result of pi
li a7, 3          # Syscall code for printing floating point number
ecall

j next_test       # Jump to the next test

error_case:
li a7, 4          # Syscall code for write
la a0, error_msg  # Load error message
ecall

next_test:
addi t0, t0, 8    # Move to the next element in tests
addi t1, t1, 8    # Move to the next element in answers
addi t5, t5, 1    # Increment the index
lw t2, num_tests  # Load the updated number of tests
blt t5, t2, test_loop # If there are more tests, continue

end_program:
li a7, 10         # Syscall code for exit
ecall

```

```

vieta:
    # Saving return address on the stack
    addi sp, sp, -4
    sw ra, 0(sp)

vieta_cycle:
    #  $a(i) = \sqrt{2 + a(i-1)}$ 
    fadd.d ft0, fa4, fa0 #  $ft0 = 2 + a(i-1)$ 
    fsqrt.d fa4, ft0     #  $a(i) = \sqrt{2 + a(i-1)}$ 
    #  $pi(i) = 2 / a(n)$ 
    fdiv.d ft3, fa0, fa4

    # Check error:  $|pi(i-1) - pi(i)|$ 
    fsub.d ft0, fa2, ft3 #  $ft0 = pi(i-1) - pi(i)$ 
    fabs.d ft0, ft0      #  $|pi(i-1) - pi(i)|$ 
    fle.d t2, ft0, fa1   # Check  $|pi(i-1) - pi(i)| \leq x$ 
    bnez t2, end_vieta   # If condition is met, exit

    # Update  $pi(n)$ 
    fmul.d fa3, fa3, ft3

    # Update  $pi(i - 1)$ 
    fmv.d fa2, ft3

    j vieta_cycle

end_vieta:
    # Return the result in fa3
    fmv.d fa3, ft3
    # Read return address from the stack
    lw ra, 0(sp)
    addi sp, sp, 4

```



```

end_vieta:
    # Return the result in fa3
    fmv.d fa3, ft3
    # Read return address from the stack
    lw ra, 0(sp)
    addi sp, sp, 4
    ret

check_x:
    mv t0, zero          # Initialize t0 to 0
    mv t1, zero          # Initialize t1 to 0
    li t3, 1             # Set t3 to 1 for checking

    # Check if x is in the range (0; 0.0005]
    fle.d t0, fa1, max_x # Check: x <= 0.0005 (max value)
    fgt.d t1, fa1, min_x # Check: x > 0 (min value)
    beq t0, t3, wrong_x_branch # If x <= 0, branch to error
    beq t1, t3, wrong_x_branch # If x > 0.0005, branch to error

    li a0, 1             # If all checks pass, set a0 to 1 (valid)
    ret                 # Return from check_x

wrong_x_branch:
    # Print error message
    li a7, 4             # Syscall code for write
    la a0, error_msg     # Load error message
    ecall
    li a0, 0             # Set a0 to 0 (invalid)
    ret                 # Return from check_x

Error: x must be in range (0; 0.0005]!
Error: x must be in range (0; 0.0005]!
Error: x must be in range (0; 0.0005]!
expected: 3.141592648614063, result: 3.141592648614063
expected: 3.141406718496502, result: 3.141406718496502

-- program is finished running (0) --

```

- Для дополнительной проверки корректности вычислений осуществлены аналогичные тестовые прогоны с использованием Python.

```

import math

def main():
    tests = [0.0, -1.0, 0.0006, 0.000001, 0.0005]
    answers = [-1.0, -1.0, -1.0, 3.141592648614063, 3.141406718496502]

    for i in range(len(tests)):
        x = tests[i]
        if is_valid_input(x):
            result = vieta(x)
            if result:
                print(f"expected: {answers[i]}, result: {result}")
            else:
                print("Error: x must be in range (0; 0.0005]!")

def is_valid_input(x):
    return 0 < x <= 0.0005

```

```

def vieta(x):
    a = 0.0
    pi = 2.0
    new_pi = None
    error = float('inf')

    while error > x:
        a = math.sqrt(2 + a)
        new_pi = 2 / a
        error = abs(pi - new_pi)
        pi = new_pi

    return pi

if __name__ == "__main__":
    main()

```

```

Error: x must be in range (0; 0.0005]!
Error: x must be in range (0; 0.0005]!
Error: x must be in range (0; 0.0005]!
expected: 3.141592648614063, result: 3.141592648614063
expected: 3.141406718496502, result: 3.141406718496502
>>>

```

- Информация о проведенных изменениях добавлена в отчет.