

АВС | ИДЗ-2 | Вариант 20

Потякин Арсений Юрьевич, БПИ237

TODO:

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

Источники информации с описанием метода решения:

- [Сайт МГУ 1](#)
- [Сайт МГУ 2](#)

Код можно найти здесь: [GitHub](#)

Метод решения задания:

Число π можно представить формулой Виета следующим образом:

$$\frac{2}{\pi} = \prod_{n=1}^{\infty} \frac{2}{\sqrt{2 + \sqrt{2 + \dots + \sqrt{2}}}}$$

При каждой итерации количество вложенных корней увеличивается на один. Чтобы приближенно найти π , необходимо использовать конечное число итераций, обозначенное как N , и преобразовать формулу:

$$\pi \approx 2 \cdot \prod_{i=1}^N \frac{2}{\sqrt{2 + \sqrt{2 + \dots + \sqrt{2}}}}$$

Как это реализовано в коде (внутри файла subroutine.s подпрограмма COMPUTE_PI):

1. Устанавливается начальное значение произведения $\text{prod} = 1.0$.
2. Инициализируется переменная $\text{current_value} = 2.0$ для вычисления вложенных корней.
3. На каждой итерации i выполняется:
 - $\text{current_value} = \text{sqrt}(2 + \text{current_value})$, где начальное $\text{current_value} = 2$.
 - $\text{factor} = 2 / \text{current_value}$ — фактор для произведения.

- Обновляется произведение: $\text{prod} = \text{prod} * \text{factor}$.
- После завершения N итераций, итоговое произведение prod умножается на 2, чтобы получить приближенное значение для π : $\text{pi_approx} = 2 * \text{prod}$.
 - Значение pi_approx сохраняется в регистре fa0 и возвращается в основной код программы.

Вычисление погрешности:

Абсолютная ошибка определяется как $|\pi_{\text{ref}} - \pi_{\text{computed}}|$ (ref - эталонное значение, computed - высчитанное для N итераций). Относительная ошибка умножается на 100 для перевода в проценты. **Если относительная ошибка меньше 0,05%, тест считается пройденным, и выводится сообщение - Passed, иначе - Failed.**

Точность 0.05% означает, что максимально допустимая погрешность равняется: $3.141592653589793 \times 0.0005 \approx 0.0015707963267949$

Тестовые прогоны программы с ручным вводом:

```
If you want a deviation of less than 0.05%, enter a value greater than 10
n: 0
n must be greater than 0
```

Рис. 1: Валидация данных: $n \leq 1$ не принимается

```
If you want a deviation of less than 0.05%, enter a value greater than 10
n: 10
3.1415923455701176
-- program is finished running (0) --
```

Рис. 2: Как видно, при $n \approx 10$ значение π имеет погрешность менее 0.05%

```
If you want a deviation of less than 0.05%, enter a value greater than 10
n: 1
3.0614674589207183
-- program is finished running (0) --
```

Рис. 3: При слишком малом n погрешность значительно возрастает

```
If you want a deviation of less than 0.05%, enter a value greater than 10
n: 10000
3.141592653589794
-- program is finished running (0) --
```

Рис. 4: При больших значений n найденное значение идеально повторяет число π , которое может уместиться в тип `double`

Автоматические тестовые прогоны:

```
1: 3.0614674589207183 - Failed
10: 3.1415923455701176 - Passed
50: 3.141592653589794 - Passed
100: 3.141592653589794 - Passed
500: 3.141592653589794 - Passed
700: 3.141592653589794 - Passed
1000: 3.141592653589794 - Passed
1500: 3.141592653589794 - Passed
2000: 3.141592653589794 - Passed
10000: 3.141592653589794 - Passed
```

Рис. 5: Результат прогона автоматических тестов

Код программы расположен на [GitHub](#)

Обоснование выполненных критериев:

```

loop_start:
    # Загрузка 'i' в t1
    lw t1 -4(sp)
    # Сравнение 'i' и 'N' (t0)
    bgt t1 t0 loop_end    # Если i > N => выход из цикла

    # Вычисление a = sqrt(2.0 + a)
    fld ft0 8(sp)         # Загружаем 'a' в ft0
    fadd.d ft2 ft1 ft0     # ft2 = 2.0 + a
    fsqrt.d ft0 ft2        # ft0 = sqrt(2.0 + a)
    fsd ft0 8(sp)          # Сохраняем новое значение 'a' на стеке 8(sp)

    # Вычисление p *= a / 2.0
    fld ft3 0(sp)         # Загружаем 'p' в ft3
    fdiv.d ft2 ft0 ft1     # ft2 = a / 2.0
    fmul.d ft3 ft3 ft2     # ft3 = p * (a / 2.0)
    fsd ft3 0(sp)         # Сохраняем обновленное значение 'p' на стеке 0(sp)

    # Инкремент i

```

Рис. 6: Весь код содержит подробные комментарии выполняемых действий

```

;
; # Вызывает подпрограмму COMPUTE_PI
; .macro compute_pi
;     # Параметр N находится в a0
;     addi sp sp -8        # Выделяем место на стеке для параметра
;     sw a0 0(sp)         # Сохраняем параметр N из a0 в стек
;     jal ra COMPUTE_PI
;     addi sp sp 8         # Восстанавливаем стек
; .end_macro
;

```

Рис. 7: Подпрограмма обернута в макрос, данные отображаются на стек, адрес возврата сохраняется на стеке внутри макроса

```

# Вызов подпрограммы, переданный параметр находится в a0
compute_pi    # Вычисляем p с N итерациями
# Результат находится в fa0

```

Рис. 8: В местах вызова функций описываются переданные параметры и возвращаемые данные

```

1
2
3 # Читает целое число от пользователя и сохраняет в %reg
4 .macro read_int (%reg)
5     li a7 5
6     ecall
7     mv %reg a0
8 .end_macro
9
10 # Выводит в консоль число типа double
11 .macro print_double (%reg)
12     fmv.d fa0 %reg
13     li a7 3
14     ecall
15 .end_macro
16
17 # Вызывает подпрограмму COMPUTE_PI
18 .macro compute_pi
19     # Параметр N находится в a0
20     addi sp sp -8           # Выделяем место на стеке для параметра
21     sw a0 0(sp)           # Сохраняем параметр N из a0 в стек
22     jal ra COMPUTE_PI
23     addi sp sp 8           # Восстанавливаем стек
24 .end_macro
25
26 # Выводит в консоль строку по адресу (mstr – memory string)
27 .macro print_mstr (%str)
28     li a7 4
29     la a0 %str
30     ecall
31 .end_macro
32

```

Рис. 9: Программа содержит макросы для ввода и вывода данных, а также для обертывания подпрограммы

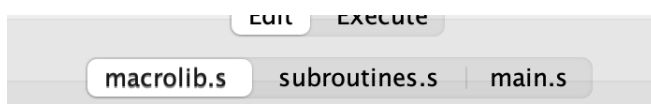
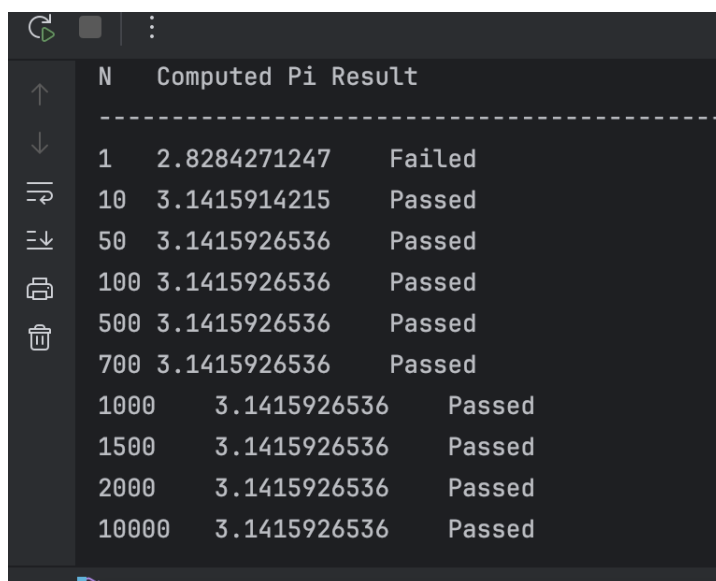


Рис. 10: Программа разбита на несколько единиц компиляции, макросы выделены в отдельную библиотеку



N	Computed Pi	Result
1	2.8284271247	Failed
10	3.1415914215	Passed
50	3.1415926536	Passed
100	3.1415926536	Passed
500	3.1415926536	Passed
700	3.1415926536	Passed
1000	3.1415926536	Passed
1500	3.1415926536	Passed
2000	3.1415926536	Passed
10000	3.1415926536	Passed

Рис. 11: Дополнительные тестовые прогоны аналогичных данных были проведены на языке C++, были получены аналогичные вердикты для каждого из значений n. Код программы на C++ можно найти на [GitHub](#)