Зенин Вадим ИДЗ по АВС. Вариант -37

Отчет на 6-7 баллов.

Отчет выполнен сразу на 6-7 баллов, поскольку семинарист разрешил так сделать.

Мой вариант - 37.

37. Разработать программу, определяющую корень уравнения $x^5 - x - 0.2 = 0$ **методом хорд** с точностью от 0,001 до 0,00000001 в диапазоне [1;1.1]. Если диапазон некорректен, то подобрать корректный диапазон.

Macrolib

В этом блоке показаны различные макросы используемые далее в программе.

```
1
2  # Печать содержимого регистра, если там храниться double значение.
3  .macro print_double(%x)
4          li a7, 3
5          fmv.d fa0, %x
6          ecall
7          newline
8  .end_macro
9
10 # Ввод дробного числа(double) числа с консоли в указанный регистр,
11 # исключая регистр a0
12 .macro read_double(%x)
13    push_d(fa0)
14    li a7, 7
15    ecall
16    fmv.d %x, fa0
17    pop_d(fa0)
18
19 .end_macro
20 # Печать строки передаваемой в макро.
21 .macro print_str (%x)
22    .data
23 str:
24    .asciz %x
25    .text
26    push (a0)
27    li a7, 4
28    la a0, str
29    ecall
30    pop  (a0)
31    .end_macro
32
33 # Печать символа передаваемой в макро.
34 .macro print_char(%x)
35    push (a0)
36    li a7, 11
37    li a0, %x
38    ecall
39    pop  (a0)
40    .end_macro
41
42 # Перевод строки.
43 .macro newline
44    print_char('\n')
45    .end_macro
```

```
46
47  # Завершение программы
48  .macro exit
49      li a7, 10
50      ecall
51  .end_macro
52
53  # Сохранение заданного регистра на стеке
54  .macro push(%x)
55          addi    sp, sp, -4
56          sw      %x, (sp)
57  .end_macro
58
59  # Выталкивание значения с вершины стека в регистр
60  .macro pop(%x)
61          lw      %x, (sp)
62          addi    sp, sp, 4
63  .end_macro
64
65  # Сохранение заданного регистра на стеке
66  .macro push_d(%x)
67          addi    sp, sp, -4
68          fsd     %x, (sp)
69  .end_macro
70
71  # Выталкивание значения с вершины стека в регистр
72  .macro pop_d(%x)
73          fld     %x, (sp)
74          addi    sp, sp, 4
75  .end_macro
76
```

```
78  # Возведение дробного числа в целочисленную степень %pow
79  .macro pow_d(%x ,%pow)
80  .data
81          null: .double 0,0
82  .text
83          push(t1)
84          push_d(ft0)
85          push_d(ft1)
86          li t1, 1                            # Итератор
87          beqz %pow, if_pow0
88          fcvt.d.w  fa0, t1
89          addi %pow, %pow, 1
90  loop:
91          bge t1 %pow end
92          fmul.d fa0, fa0, %x
93          addi t1 t1 1
94          j loop
95  if_pow0:
96          fcvt.d.w ft1, t1
97          fld ft0 null t0
98          fadd.d fa0, ft1, ft0
99  end:
100         pop_d(ft1)
101         pop_d(ft0)
102         pop(t1)
103 .end_macro
104
105 # Макрос, который считает значение моей x^5-x-0.2=0
106 .macro my_func(%x)
107 .data
108         const: .double 0.2
109         null: .double 0.0
110 .text
111         push_d(ft5)
112         push_d(ft6)
113         push(t0)
114
115         fld ft5 null t0
116         fld ft6 const t0
117         fsub.d ft5, ft5, ft6
118         fsub.d ft5, ft5, %x
119         li t0 5
120         pow_d(%x, t0)
121         fadd.d fa0, fa0, ft5
122
123         pop(t0)
124         pop_d(ft6)
125         pop_d(ft5)
126 .end_macro
```

Блок .data

На этом скриншоте показаны блок .data

```
.include "macrolib.asm"

.globl  A_x0, B_x1, array_for_rounding
.data
        array_for_rounding: .word 1000, 10000, 100000, 1000000  # Массив для округления чисел
        A_x0: .double 1.0                                       # Левая граница интервала
        B_x1: .double 1.1                                       # Правая граница интервала
```

Блок main

```
.include "macrolib.asm"

.globl  A_x0, B_x1, array_for_rounding
.data
        array_for_rounding: .word 1000, 10000, 100000, 1000000  # Массив для округления чисел
        A_x0: .double 1.0                                       # Левая граница интервала
        B_x1: .double 1.1                                       # Правая граница интервала

.text
main:
        la t0 input                                             # Ввод и проверка ввода, на соответствие точности вычислений.
        jalr t0
        la t0 algorithm                                         # Основной алгортим, приближения корня методом хорд(параметры:
                                                                # ft0 – левая граница интервала, ft1 – правая, ft3 – приближенный корень)
        jalr t0
        la t0 rounding                                          # Алгоритм округления полученного числа (параметры: t0 – адрес массива,
                                                                # ft11 – заданная точность, ft9 – корректоная размерность округления, ft2 – число необходимое округлить)
        jalr t0

        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
        exit
```

Сначала запрашиваю у пользователя точность необходимых вычислений и проверяю ее.
Далее программа состоит из последовательных переходов в подпрограммы использую jalr.
В самом конце запускаю макрос для завершения программы.

Ввод данных

На данном скриншоте показана реализация процесса запроса точности вычислений у пользователя.
Вывожу на экран пользователю подсказку и считываю его число.

После проверяю точность на принадлежность необходимому массиву.

```
.include "macrolib.asm"
.globl input
.data
        array_accuracy: .double 0.001, 0.0001, 0.00001, 0.000001 # Массив возможных точностей
.text
input:
        push(ra)
        print_str("Введите точность вычислений [0.001:0.0000001]: ")
        read_double(ft11)                   # Считываем введённое значение в ft11

check_input:
        la t0, array_accuracy               # Загружаем адрес начала массива точностей в t0
        li t1, 4                            # Количество элементов в массиве
        li t2, 0                            # Счётчик для прохода по массиву

loop:
        beq t2, t1, repeat_input            # Если прошли все элементы и не нашли совпадения, запросить ввод повторно

        fld ft0, 0(t0)                      # Загружаем текущий элемент массива в ft0
        feq.d t3, ft0, ft11                 # Сравниваем введённое значение с элементом массива

        beqz t3, next_element               # Если не совпадает, переход к следующему элементу

        j end                               # Если совпало, завершить проверку

next_element:
        addi t2, t2, 1                      # Увеличиваем счётчик
        addi t0, t0, 8                      # Переходим к следующему элементу массива (каждый double занимает 8 байт)
        j loop                              # Возвращаемся в цикл

repeat_input:
        print_str("Некорректный ввод! Повторите попытку.\n")
        read_double(ft11)                   # Считываем введённое значение в ft11
        j check_input
end:
        pop(ra)
        ret
```

Основной алгоритм

Он состоит в том, чтобы приближать значение икса, используя уравнение хорды.

```
.include "macrolib.asm"
.globl algorithm

.text
# Зададим, два значения x_0 и x_1, такие что значения функции от этих аргументов будут разных знаков.
# Следующий шаг алгоритма: обновления приближения корня с помощью уравнения хорды.
# Итерации происходят до тех пор, пока разность между последовательными приближениями не станет, меньше заданной точности.
algorithm:
        push(ra)                                        # Пусть x_0 и x_1 равны границам нашего интервала.
        fld ft0 A_x0 t0
        fld ft1 B_x1 t0

loop:
        fsub.d fs2, ft1, ft0                    # значение x_1-x_0
        my_func(ft0)                            # Значение f(x_0)
        fmv.d fs3, fa0                          # Переносим в fs3

        my_func(ft1)                            # Значение f(x_1)
        fmv.d fs4, fa0                          # Переносим в fs4

        fmul.d fs2, fs2, fs3                    # Умножаем данную дробь на f(x_0)
        fsub.d fs3, fs4, fs3                    # Находим f(x_1) - f(x_0)

        fdiv.d fs2, fs2, fs3                    # Делим разность аргументов на разность значений

        fsub.d ft2, ft0, fs2                    # Находим x_2

check:
        fcvt.d.w   fs5, zero

        fsub.d fs5, ft2, ft1
        fabs.d fs5, fs5                         # Нахожу модуль разности x_i и x_{i-1}

        flt.d   t1, fs5, ft11                   # Сравниваю его с заданной точностью
        bnez   t1, end_loop                     # Выхожу из цикла, если точность больше либо равна


        fmv.d ft0, ft1
        fmv.d ft1, ft2
        fcvt.d.w   ft2, zero
        j loop
end_loop:
        pop(ra)
        ret
```

Тестовое покрытие

> Можно рассмотреть тестовое покрытие и увидеть, что программа работает корректно и рассмотрены все возможные допустимые значения.

```
Введите точность вычислений [0.001:0.0000001]: **** user input : 0.3456
Некорректный ввод! Повторите попытку.
**** user input : 0.256
Некорректный ввод! Повторите попытку.
**** user input : 0.001
Вот значение корня с заданной вами точностью :1.045
```

```
Введите точность вычислений [0.001:0.0000001]: **** user input : 0.456256
Некорректный ввод! Повторите попытку.
**** user input : 0.2564
Некорректный ввод! Повторите попытку.
**** user input : 0.0001
Вот значение корня с заданной вами точностью :1.0448
```

```
Введите точность вычислений [0.001:0.0000001]: **** user input : 0.00001
Вот значение корня с заданной вами точностью :1.04476
```

```
Введите точность вычислений [0.001:0.0000001]: **** user input : 0.456435
Некорректный ввод! Повторите попытку.
**** user input : 0.525252
Некорректный ввод! Повторите попытку.
**** user input : 0.000001
Вот значение корня с заданной вами точностью :1.044761
```

Отчет на 8 баллов.

- Разработанные подпрограммы должны поддерживать многократное использование с различными наборами исходных данных, включая возможность обработки в качестве параметров различных исходных данных.

Разработанные мной подпрограммы поддерживают работу с различными исходными данными.

- Можно заметить, что в тестовом покрытии представлена работа с различными исходными данными, включая их обработку в качестве параметров.
  Вывод все подпрограммы универсальны и соответствуют требованию

Тестовая программа

- Реализовать автоматизированное тестирование за счет создания **дополнительной тестовой программы**, осуществляющей прогон подпрограмм, осуществляющих вычисления для различных тестовых данных (вместо их ввода). Осуществить прогон тестов обеспечивающих покрытие различных ситуаций. В том случае, если исходные данные напрямую не прописаны, а точность в условии зафиксирована, использовать организацию вычислений с различной точностью.

На данном скриншоте представлена работа тестирования программы с различными вариациями точности.

```
.include "macrolib.asm"
# Загружаем возможные значения точности для тестов
.data
        test1: .double 0.001
        test2: .double 0.0001
        test3: .double 0.00001
        test4: .double 0.000001
.text
tests:
# Тестирование программы
test_1:
        fld ft11 test1 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
test_2:
        fld ft11 test2 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)

test_3:
        fld ft11 test3 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
test_4:
        fld ft11 test4 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
        exit
```

На данном скриншоте представлен вывод данных, которые подтверждают правильную работоспособность программы.

```
Вот значение корня с заданной вами точностью :1.045
Вот значение корня с заданной вами точностью :1.0448
Вот значение корня с заданной вами точностью :1.04476
Вот значение корня с заданной вами точностью :1.044761
```

- Для дополнительной проверки корректности вычислений осуществить аналогичные тестовые прогоны с использованием существующих библиотек и одного из языков программирования высокого уровня по выбору: C, C++, Python.

Для данного пункта я написал данный код на Питоне

```python
def algorithm(f, x0, x1, acc=0.001, max_iter=100000):  1 usage
    for i in range(max_iter):
        # Вычисляем значения функции в текущих точках
        f_x0 = f(x0)
        f_x1 = f(x1)
        # Формула для вычисления следующего приближения
        x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        # Проверяем условие завершения (если разница между x1 и x2 меньше заданной точности)
        if abs(x2 - x1) < 10**-(acc):
            print(f"Корень найден: x = {round(x2,acc)}")
            return x2
        # Обновляем значения для следующей итерации
        x0, x1 = x1, x2

def f(x):  1 usage
    return x ** 5 - x - 0.2

# Задаем начальные приближения x0 и x1
x0 = 1.0
x1 = 1.1

# Вызываем функцию
for i in range(4):
    root = algorithm(f, x0, x1, acc = i+3)
    print("Приближенное значение корня:", root)
```

И он дал крайне приближенные значения

```
Корень найден: x = 1.045
Приближенное значение корня: 1.044770307969021
Корень найден: x = 1.0448
Приближенное значение корня: 1.0447616869086973
Корень найден: x = 1.04476
Приближенное значение корня: 1.0447616869086973
Корень найден: x = 1.044762
Приближенное значение корня: 1.044761700075292
```

Отчет на 9

Макросы

Все макросы, используемые в программе, и их реализация предоставлены выше. В этом пункте я вкратце опишу их функционал.

- print_ double (%x) - выводит на экран число из регистра %x.
- read_double(%x) - считывает число в регистр %x, исключая a0.
- print_str (%x) - выводит на экран строку %x.
- print_char(%x) - выводит на экран символ %x.
- newline - переводит строку.
- exit - вызывает системный вызов 10 - завершение программы.
- push(%x) - сохранение регистра %x на стеке.
- pop(%x) - "выталкивание" значения с вершины стека в регистр %x.
- push_d(%x) - Аналогичные макросы для double.
- pop_d(%x) - "Аналогичные макросы для double.
- my_func(%x) - Считает значение функции с аргументом из %x
- pow_d(%x ,%pow) - Возводит содержимое %x, в целочисленную степень %pow.

Отчет на 10

Разобьем программу по разным файлам.

main.asm

Файлик с запуском основной программы.

```
.include "macrolib.asm"

.globl  A_x0, B_x1, array_for_rounding
.data
        array_for_rounding: .word 1000, 10000, 100000, 1000000  # Массив для округления чисел
        A_x0: .double 1.0                                       # Левая граница интервала
        B_x1: .double 1.1                                       # Правая граница интервала

.text
main:
        la t0 input                                             # Ввод и проверка ввода, на соответствие точности вычислений.
        jalr t0
        la t0 algorithm                                         # Основной алгортим, приближения корня методом хорд(параметры:
                                                                # ft0 – левая граница интервала, ft1 – правая, ft3 – приближенный корень)
        jalr t0
        la t0 rounding                                          # Алгоритм округления полученного числа (параметры: t0 – адрес массива,
                                                                # ft11 – заданная точность, ft9 – корректоная размерность округления, ft2 – число необходимое округлить)
        jalr t0

        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
        exit

```

input.asm

Файлик с подпрограммой ввода точности.

```
.include "macrolib.asm"
.globl input
.data
        array_accuracy: .double 0.001, 0.0001, 0.00001, 0.000001 # Массив возможных точностей
.text
input:
        push(ra)
        print_str("Введите точность вычислений [0.001:0.0000001]: ")
        read_double(ft11)                # Считываем введённое значение в ft11

check_input:
        la t0, array_accuracy            # Загружаем адрес начала массива точностей в t0
        li t1, 4                         # Количество элементов в массиве
        li t2, 0                         # Счётчик для прохода по массиву

loop:
        beq t2, t1, repeat_input         # Если прошли все элементы и не нашли совпадения, запросить ввод повторно

        fld ft0, 0(t0)                   # Загружаем текущий элемент массива в ft0
        feq.d t3, ft0, ft11              # Сравниваем введённое значение с элементом массива

        beqz t3, next_element            # Если не совпадает, переход к следующему элементу

        j end                            # Если совпало, завершить проверку

next_element:
        addi t2, t2, 1                   # Увеличиваем счётчик
        addi t0, t0, 8                   # Переходим к следующему элементу массива (каждый double занимает 8 байт)
        j loop                           # Возвращаемся в цикл

repeat_input:
        print_str("Некорректный ввод! Повторите попытку.\n")
        read_double(ft11)                # Считываем введённое значение в ft11
        j check_input
end:
        pop(ra)
        ret
```

algorithm.asm

Файлик, который содержит основной алгоритм создания массива B.

```
.include "macrolib.asm"
.globl algorithm

.text
# Зададим, два значения x_0 и x_1, такие что значения функции от этих аргументов будут разных знаков.
# Следующий шаг алгоритма: обновления приближения корня с помощью уравнения хорды.
# Итерации происходят до тех пор, пока разность между последовательными приближениями не станет, меньше заданной точности.
algorithm:
        push(ra)                                        # Пусть x_0 и x_1 равны границам нашего интервала.
        fld ft0 A_x0 t0
        fld ft1 B_x1 t0

loop:
        fsub.d fs2, ft1, ft0                    # значение x_1-x_0
        my_func(ft0)                            # Значение f(x_0)
        fmv.d fs3, fa0                          # Переносим в fs3

        my_func(ft1)                            # Значение f(x_1)
        fmv.d fs4, fa0                          # Переносим в fs4

        fmul.d fs2, fs2, fs3                    # Умножаем данную дробь на f(x_0)
        fsub.d fs3, fs4, fs3                    # Находим f(x_1) - f(x_0)

        fdiv.d fs2, fs2, fs3                    # Делим разность аргументов на разность значений

        fsub.d ft2, ft0, fs2                    # Находим x_2

check:
        fcvt.d.w   fs5, zero

        fsub.d fs5, ft2, ft1
        fabs.d fs5, fs5                         # Нахожу модуль разности x_i и x_{i-1}

        flt.d   t1, fs5, ft11                   # Сравниваю его с заданной точностью
        bnez   t1, end_loop                     # Выхожу из цикла, если точность больше либо равна


        fmv.d ft0, ft1
        fmv.d ft1, ft2
        fcvt.d.w   ft2, zero
        j loop
end_loop:
        pop(ra)
        ret
```

tests.asm

Файлик, содержащий реализацию тестов.

```
.include "macrolib.asm"
# Загружаем возможные значения точности для тестов
.data
        test1: .double 0.001
        test2: .double 0.0001
        test3: .double 0.00001
        test4: .double 0.000001
.text
tests:
# Тестирование программы
test_1:
        fld ft11 test1 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
test_2:
        fld ft11 test2 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)

test_3:
        fld ft11 test3 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
test_4:
        fld ft11 test4 t0
        la t0 algorithm
        jalr t0
        la t0 rounding
        jalr t0
        print_str("Вот значение корня с заданной вами точностью :")
        print_double(ft2)
        exit
```

macrolib.asm

Файлик, содержащий библиотеку макросов.

```
1
2  # Печать содержимого регистра, если там храниться double значение.
3  .macro print_double(%x)
4          li a7, 3
5          fmv.d fa0, %x
6          ecall
7          newline
8  .end_macro
9
10 # Ввод дробного числа(double) числа с консоли в указанный регистр,
11 # исключая регистр a0
12 .macro read_double(%x)
13    push_d(fa0)
14    li a7, 7
15    ecall
16    fmv.d %x, fa0
17    pop_d(fa0)
18
19 .end_macro
20 # Печать строки передаваемой в макро.
21 .macro print_str (%x)
22    .data
23 str:
24    .asciz %x
25    .text
26    push (a0)
27    li a7, 4
28    la a0, str
29    ecall
30    pop  (a0)
31    .end_macro
32
33 # Печать символа передаваемой в макро.
34 .macro print_char(%x)
35    push (a0)
36    li a7, 11
37    li a0, %x
38    ecall
39    pop  (a0)
40    .end_macro
41
```

```
42  # Перевод строки.
43  .macro newline
44     print_char('\n')
45     .end_macro
46
47  # Завершение программы
48  .macro exit
49      li a7, 10
50      ecall
51  .end_macro
52
53  # Сохранение заданного регистра на стеке
54  .macro push(%x)
55          addi    sp, sp, -4
56          sw      %x, (sp)
57  .end_macro
58
59  # Выталкивание значения с вершины стека в регистр
60  .macro pop(%x)
61          lw      %x, (sp)
62          addi    sp, sp, 4
63  .end_macro
64
65  # Сохранение заданного регистра на стеке
66  .macro push_d(%x)
67          addi    sp, sp, -4
68          fsd     %x, (sp)
69  .end_macro
70
71  # Выталкивание значения с вершины стека в регистр
72  .macro pop_d(%x)
73          fld     %x, (sp)
74          addi    sp, sp, 4
75  .end_macro
```

```
78  # Возведение дробного числа в целочисленную степень %pow
79  .macro pow_d(%x ,%pow)
80  .data
81          null: .double 0,0
82  .text
83          push(ra)
84          push(t1)
85          push_d(ft0)
86          push_d(ft1)
87          li t1, 1                            # Итератор
88          beqz %pow, if_pow0
89          fcvt.d.w  fa0, t1
90          addi %pow, %pow, 1
91  loop:
92          bge t1 %pow end
93          fmul.d fa0, fa0, %x
94          addi t1 t1 1
95          j loop
96  if_pow0:
97          fcvt.d.w ft1, t1
98          fld ft0 null t0
99          fadd.d fa0, ft1, ft0
100 end:
101         pop_d(ft1)
102         pop_d(ft0)
103         pop(t1)
104         pop(ra)
105 .end_macro
106
```

```
106
107 # Макрос, который считает значение моей x^5-x-0.2=0
108 .macro my_func(%x)
109 .data
110         const: .double 0.2
111         null: .double 0.0
112 .text
113         push_d(ft5)
114         push_d(ft6)
115         push(t0)
116
117         fld ft5 null t0
118         fld ft6 const t0
119         fsub.d ft5, ft5, ft6
120         fsub.d ft5, ft5, %x
121         li t0 5
122         pow_d(%x, t0)
123         fadd.d fa0, fa0, ft5
124
125         pop(t0)
126         pop_d(ft6)
127         pop_d(ft5)
128 .end_macro
129
```