ACS. Individual Homework 2

Александр Васюков | БПИ235 (239)

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.`

Вариант 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.`

Решение, претендующее на 10 баллов:
Считаем корень методом хорд по формуле
`c = left - (f(left) * (right - left)) / (f(right) - f(left))`,
где
`f(x) = x^5 - x - 0.2;`
`left - левая граница`;
`right - правая граница`.

Файлы хранятся на GitHub по ссылке: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/ACS/Homework/IHW_2
Есть 4 файла:

1. main.asm - взаимодействие с пользователем. Предлагается выбор автотестов или ввода собственного значения эпсилон. Также предусмотрен повторный ввод данных после выполнения программы.
2. solution.asm - макрос по вычислению приближённого корня. Сначала считываются значения границ (`left` и `right`), со стека считывается эпсилон. Дальше происходит расчёт по формуле, смена границ и проверка выхода из цикла. В конце выводится корень уравнения.
3. tests.asm - автоматические тесты при эпсилон, равных:
    1. 0.001
    2. 0.0001
    3. 0.00001
    4. 0.000001
    5. 0.0000001
    6. 0.00000001
4. macros.asm - библиотека с макросами:
    1. pow_double_int - возведение числа double в степень int.
    2. f - функция x^5 - x - 0.2.
    3. user - ввод эпсилон от пользователя.
    4. get_epsilon - получение эпсилона с клавиатуры.
    5. check_epsilon - проверка эпсилона на соответствие границам [1, 1.1].
    6. print - вывод числа double в консоль.
    7. print_eps - вывод на экран текущего эпсилона.

Пример работы программы

```
MENU
Enter '0': input your epsilon.
Enter other symbol: launch autotests.

Your choice: 0
Input the epsilon from 0.1 to 0.00000001: 0.00005
1.044752418928656
If you want to finish program, enter '0', otherwise press any other number to restart: 2

MENU
Enter '0': input your epsilon.
Enter other symbol: launch autotests.

Your choice: 0
Input the epsilon from 0.1 to 0.00000001: 1

You need input the number from 0.1 to 0.00000001. Try again!
Input the epsilon from 0.1 to 0.00000001: -123

You need input the number from 0.1 to 0.00000001. Try again!
Input the epsilon from 0.1 to 0.00000001: 0.0000342
1.0447606035444905
If you want to finish program, enter '0', otherwise press any other number to restart: 1

MENU
Enter '0': input your epsilon.
Enter other symbol: launch autotests.

Your choice: 1
Result for the test with epsilon = 0.001: 1.0446831501003284
Result for the test with epsilon = 1.0E-4: 1.044752418928656
Result for the test with epsilon = 1.0E-5: 1.0447606035444905
Result for the test with epsilon = 1.0E-6: 1.044761570525993
Result for the test with epsilon = 1.0E-7: 1.0447616847699501
Result for the test with epsilon = 1.0E-8: 1.044761698267276
If you want to finish program, enter '0', otherwise press any other number to restart: 0

-- program is finished running (0) --
```

Выполненные условия:

На 4-5

1. Есть решение на ассемблере. Есть ввод и вывод данных на экран.
2. Все изменяемые параметры программы вводятся с консоли.
3. Обработка данных, полученных из файла сформирована в виде отдельной подпрограммы.
4. В подкаталоге данных присутствуют файлы, используемые для тестирования.
5. Буфер для текста программы имеет фиксированный размер размером не менее 4096 байт, допускающий ввод без искажений только тексты, ограниченные этим размером.
6. При чтении файла размером, превышающим размер буфера, не происходит падения программы. Программа корректно обрабатывает введенный «урезанный» текст.
7. Отчёт с полным тестовым покрытием.
   На 6-7
8. Сохранение локальных переменных в свободных регистрах.
9. Программа читает файлы размером до 10 килобайт.
10. Есть подпрограммы.
11. Не нарушены условия конвенции.
12. Изменения в отчёте. Решение сразу написано.
    На 8
13. Есть возможность дополнительного вывода результатов на консоль.
14. Есть многократный вызов программы.
15. Добавлена информация в отчёт.
    На 9
16. Используются макросы.
17. Реализована дополнительная тестовая программа.
    На 10
18. Программа разбита на несколько файлов.
19. Макросы выделены в отдельную автономную библиотеку.

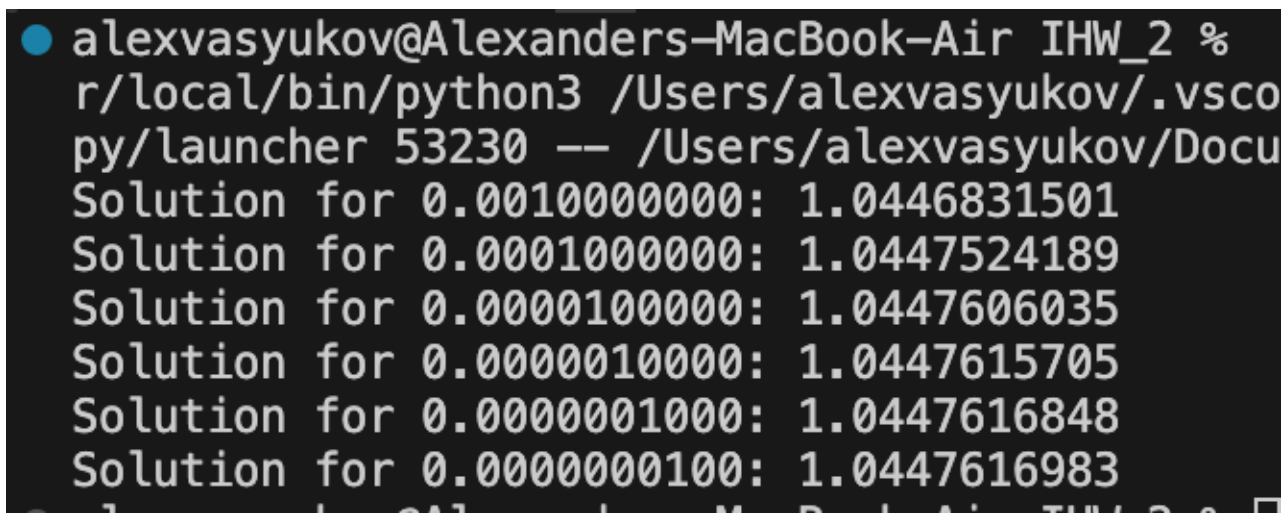
20. Используются дополнительные графические диалоговые окна для ввода и отображения диалогов, предоставляемые симулятором RARS.
21. Всё добавлено в отчёт.

Аналогичная программа на Python для проверки точности значений

secant.py

```python
def f(x):
        return x**5 - x - 0.2

def secant(start_left, start_right, eps):
        left = start_left
        right = start_right
        while abs(right - left) > eps:
                c = left - (f(left) * (right - left)) / (f(right) - f(left))
                f_c = f(c)

                if abs(f_c) < eps:
                        return c

                if f_c > 0:
                        right = c
                else:
                        left = c
        return c

def main():
        eps = 0.001
        left = 1.0
        right = 1.1

        if f(left) * f(right) >= 0:
                print(f"There is not the solution between {left} and {right}")
                return

        for i in range(6):
                res = secant(left, right, eps)
                print(f"Solution for {eps:.10f}: {res:.10f}")
                eps /= 10

if __name__ == "__main__":
        main()
```

Результат:



Код

main.asm

```
.data
prompt_start:   .asciz "\nMENU\nEnter '0': input your epsilon.\nEnter other symbol: launch autotests.\n\nYour
choice: "
prompt_next:    .asciz "If you want to finish program, enter '0', otherwise press any other number to
restart: "
sep:            .asciz " "
newline:        .asciz "\n"

.include "macro.asm"
.include "testing.asm"
.include "tests.asm"
```

```
.text
.global main
main:
        li      s0 0
        fcvt.d.w ft0 s0

        # Output message for choice own array or autotests
        li      a7 4
        la      a0 prompt_start
        ecall
        # Read number of choice
        li      a7 5
        ecall
        beqz    a0 your_eps
        j       to_tests
your_eps:
        # User iunputs the epsilon
        user()

        # Solution
        secant()

what_is_next:
        # Message with an offer to continue
        li      a7 4
        la      a0 prompt_next
        ecall
        # Read number of choice
        li      a7 5
        ecall
        beqz    a0 end
        j       main
to_tests:
        # Launch autotests
        call    autotests
        j       what_is_next
end:
        # Stop
        li      a7 10
        ecall
```

solution.asm

```
.macro secant()
.data
start_left:     .double 1.0
start_right:    .double 1.1

.text
        fld     ft0 start_left t0
        fld     ft1 start_right t0
        fld     ft2 (sp)
        addi    sp sp 8
        fsub.d  ft3 ft0 ft0     # double zero

work:
        fsub.d  ft4 ft1 ft0     # ft4 = right - left
        fabs.d  ft5 ft4         # ft5 = |right - left|

        fgt.d   t1 ft5 ft2      # If ft5 > eps, t1 = 1
        beqz    t1 end_secant

        f(ft0, fa0)     # fa0 = f(left)
        f(ft1, fa1)     # fa1 = f(right)

        fmul.d  ft6 fa0 ft4
        fsub.d  ft7 fa1 fa0
        fdiv.d  ft6 ft6 ft7
        fsub.d  fa2 ft0 ft6     # c = left - ft6

        f(fa2, fa3)             # fa3 = f(c)

        fgt.d   t1 fa3 ft3      # if fa0 > ft3 (f > 0), then t1 = 1, else t1 = 0
        beqz    t1 change_left  # if t1 == 1, then change right
```

```
change_right:
        fmv.d   ft1 fa2
        j       next
change_left:
        fmv.d   ft0 fa2

next:
        fabs.d  fa4 fa3         # fa4 = |f(c)|
        flt.d   t1 fa4 ft2      # if fa0 < ft4 (f < eps), then t1 = 1, else t1 = 0
        bnez    t1 end_secant   # if t1 != 0, the go out
        j work

end_secant:
        print(fa2)
.end_macro
```

tests.asm

```
.data
test_1: .double 0.001
test_2: .double 0.0001
test_3: .double 0.00001
test_4: .double 0.000001
test_5: .double 0.0000001
test_6: .double 0.00000001

.text
autotests:
        # Save return address
        addi    sp sp -4
        sw      ra (sp)

        # Test 1
        fld     ft0 test_1 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Test 2
        fld     ft0 test_2 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Test 3
        fld     ft0 test_3 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Test 4
        fld     ft0 test_4 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Test 5
        fld     ft0 test_5 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Test 6
        fld     ft0 test_6 t0
        addi    sp sp -8
        fsd     ft0 (sp)
        print_eps(ft0)
        secant()

        # Load return address
        lw      ra (sp)
```

```
        addi    sp sp 4
        ret
```

macro.asm

```
.macro pow_double_int(%x, %y, %res)
.text
        # Set default meanings
        fmv.d   fa3 %x
        li      t3 1
        fcvt.d.w fa4 t3
        mv      t2 %y

        # The loop for counting of time of power
loop:   beqz    t2 end
        addi    t2 t2 -1
        fmul.d  fa4 fa4 fa3
        j       loop

end:    fmv.d   %res fa4
.end_macro


.macro print(%x)
.data
sep:    .asciz "\n"
.text
        # Read input meaning to registr fa0
        fcvt.d.w ft0 s0
        fadd.d  ft0 %x ft0
        fmv.d   fa0 ft0

        li      a7 3
        ecall
        # Output separation
        li      a7 4
        la      a0 sep
        ecall
.end_macro


.macro f(%x, %res)
# f(x) = x^5 - x - 0.2
.text
        fmv.d   ft8 %x
        li      t5 5
        pow_double_int(ft8, t5, fa4) # x^5
        fsub.d  fa4 fa4 ft8 # x^5 - x

        li      t3 2
        li      t4 10
        fcvt.d.w ft8 t3
        fcvt.d.w ft9 t4
        fdiv.d  ft8 ft8 ft9 # 0.2

        fsub.d  fa4 fa4 ft8 # x^5 - x - 0.2
        fmv.d   %res fa4
.end_macro


.macro user()
.text
user_eps:
        # Getting epsilon
        get_epsilon(ft0)
        # Checking epsilon
        check_epsilon(ft0, t1)
        beqz    t1 user_eps
.end_macro


.macro get_epsilon(%x)
.data
prompt_eps:     .asciz "Input the epsilon from 0.1 to 0.00000001: "
.text
        # Message for input epsilon
```

```
        li      a7 4
        la      a0 prompt_eps
        ecall
        # Read number
        li      a7 7
        ecall
        # Set number to register %x
        fadd.d  %x fa0 ft0
.end_macro


.macro check_epsilon(%eps, %x)
.data
eps_min:        .double 0.00000001
eps_max:        .double 0.001
prompt_er_eps:  .asciz "\nYou need input the number from 0.1 to 0.00000001. Try again!\n"
.text
        fmv.d   fa0 %eps
        # Boundary values of epsilon
        fld     ft0 eps_min t0
        fld     ft1 eps_max t0
        # Checking
        fge.d   t1 fa0 ft0
        beqz    t1 error_eps

        fle.d   t1 fa0 ft1
        beqz    t1 error_eps

        # Set 1 (true), if epsilon is correct
        li      %x 1
        j       end_check
error_eps:
        # Message that the array size is incorrect
        li      a7 4
        la      a0 prompt_er_eps
        ecall
        # Set 0 (false), if size is incorrect
        li      %x 0
end_check:
        # Save epsilon on the stack
        addi    sp sp -8
        fsd     fa0 (sp)
.end_macro

.macro print_eps(%eps)
.data
prompt_eps:     .asciz "Result for the test with epsilon = "
prompt_eps_2:   .asciz ": "
.text
        li      a7 4
        la      a0 prompt_eps
        ecall

        # Output epsilon
        fmv.d   fa0 %eps
        li      a7 3
        ecall

        li      a7 4
        la      a0 prompt_eps_2
        ecall
.end_macro
```