Отчёт по ИДЗ-2

Пасынков Матвей Евгеньевич, БПИ237, Вариант 26.

Условие задания.

> Разработать программу вычисления корня пятой степени согласно
> быстро сходящемуся итерационному алгоритму определения
> корня n-той степени с точностью не хуже 0,1%.

Описание метода решения задания.

Мы реализуем быстро сходящийся итерационный алгоритм, который работает до тех пор, пока точность будет не хуже 0,1%, то есть модуль разности между предыдущим и нынешним результатом будет менее 0,1% = 0,001 (но в моём коде точность будет 0,000001, то есть гораздо лучше, чем нужно).

Суть быстро сходящегося итерационного алгоритма следующая (согласно Википедии[1]):

1. Сделать начальное предположение x0.
2. Задать

$$x_{k+1} = \frac{1}{n}\left((n-1)x_k + \frac{A}{x_k^{n-1}}\right)$$

3. Повторять шаг 2, пока не будет достигнута необходимая точность.

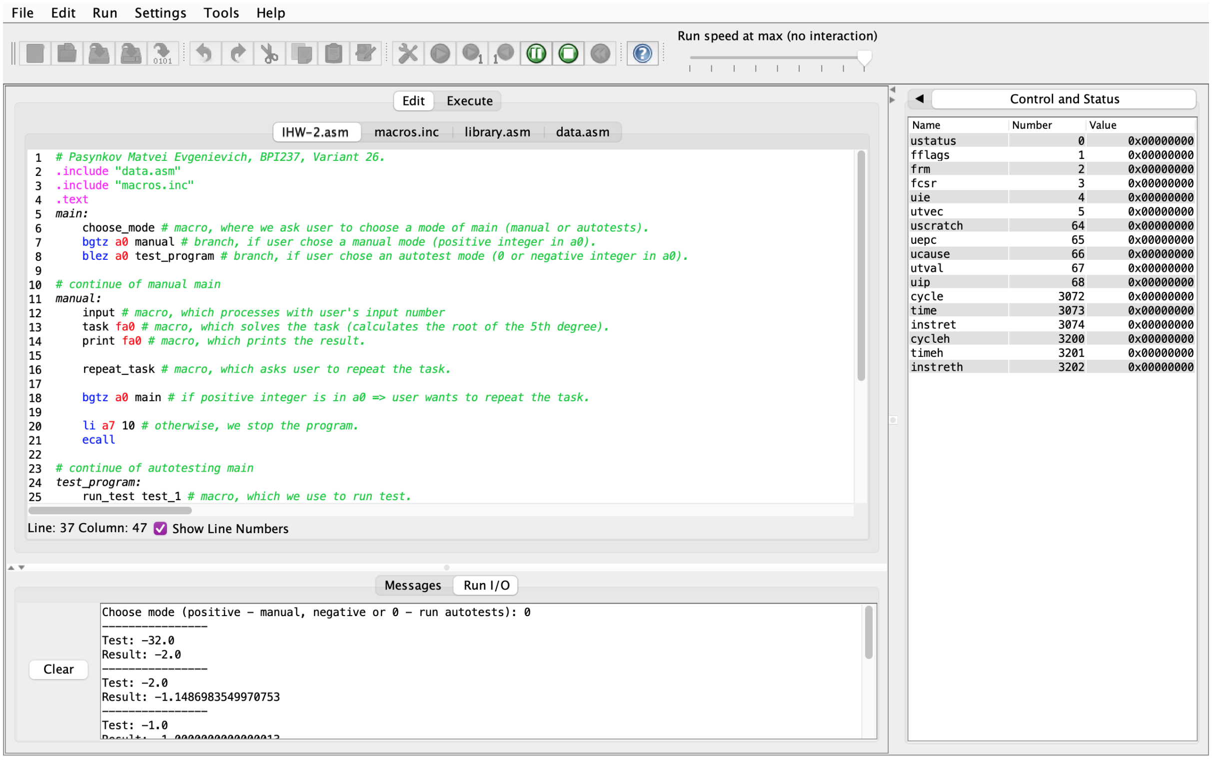
Источники информации

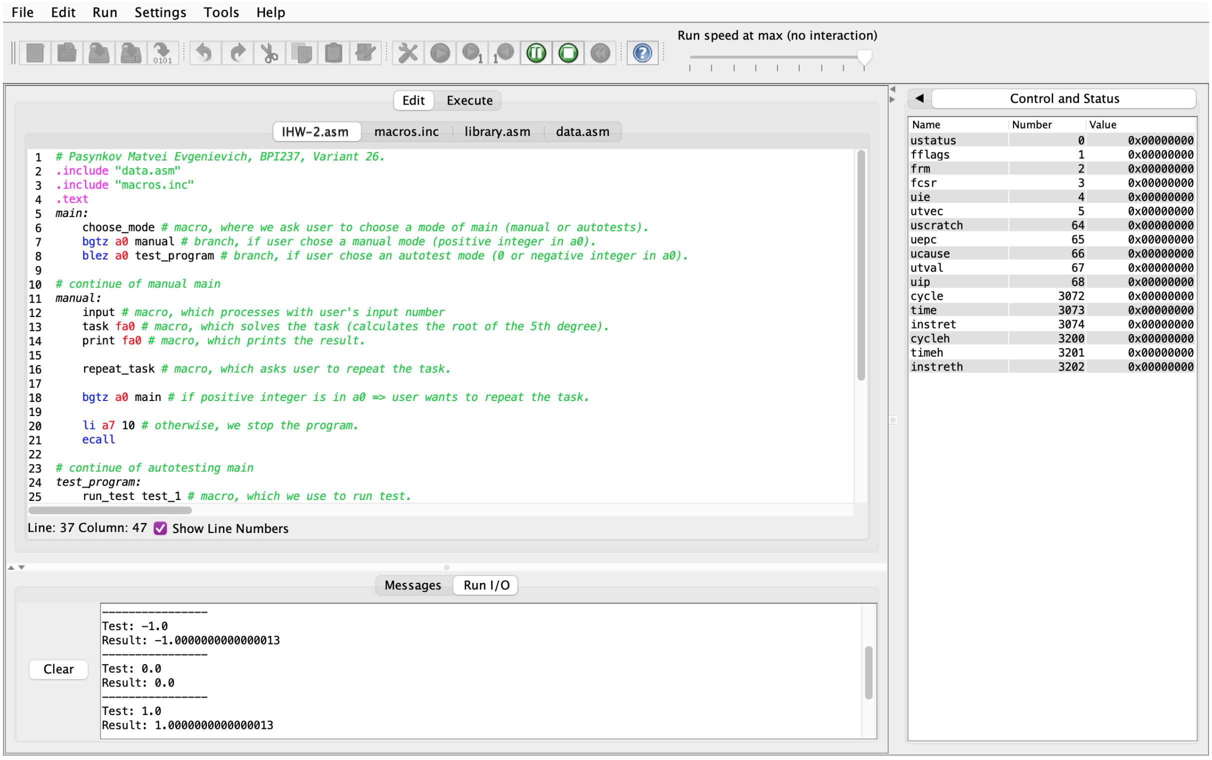
1. https://ru.wikipedia.org/wiki/Алгоритм_нахождения_корня_n-ной_степени

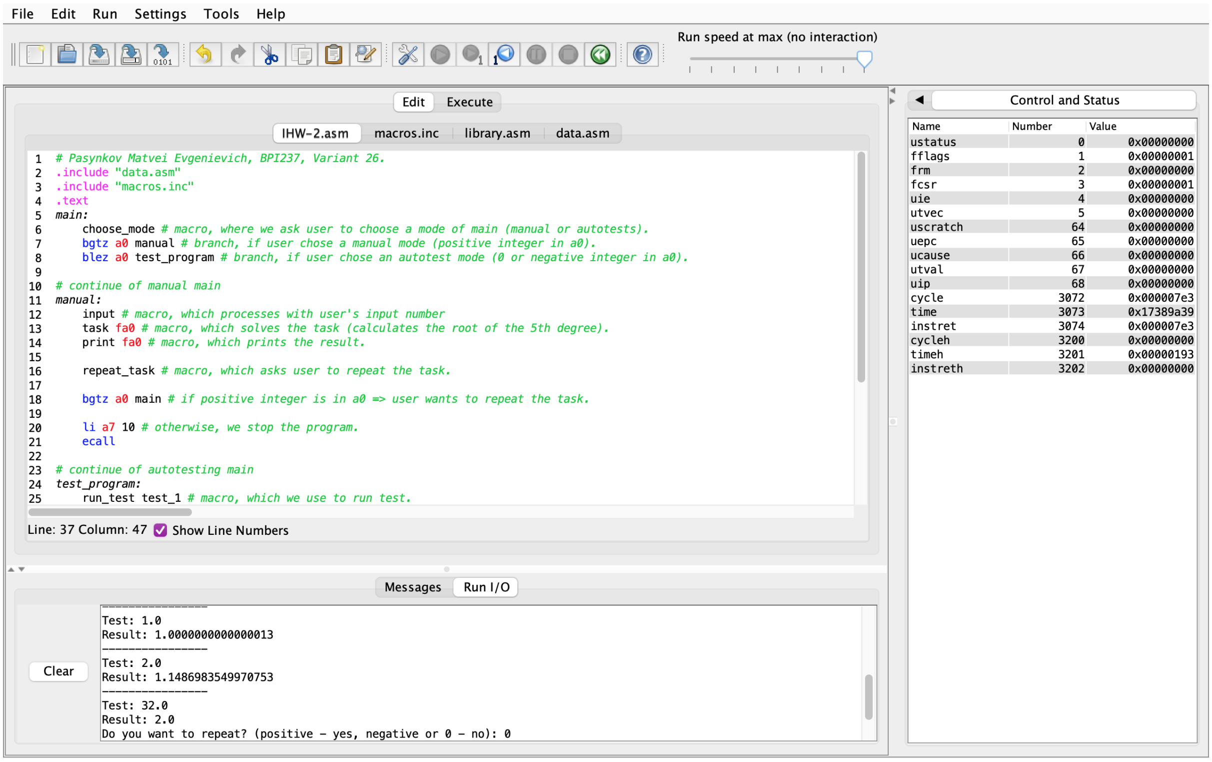
Критерии на 4-5 баллов.

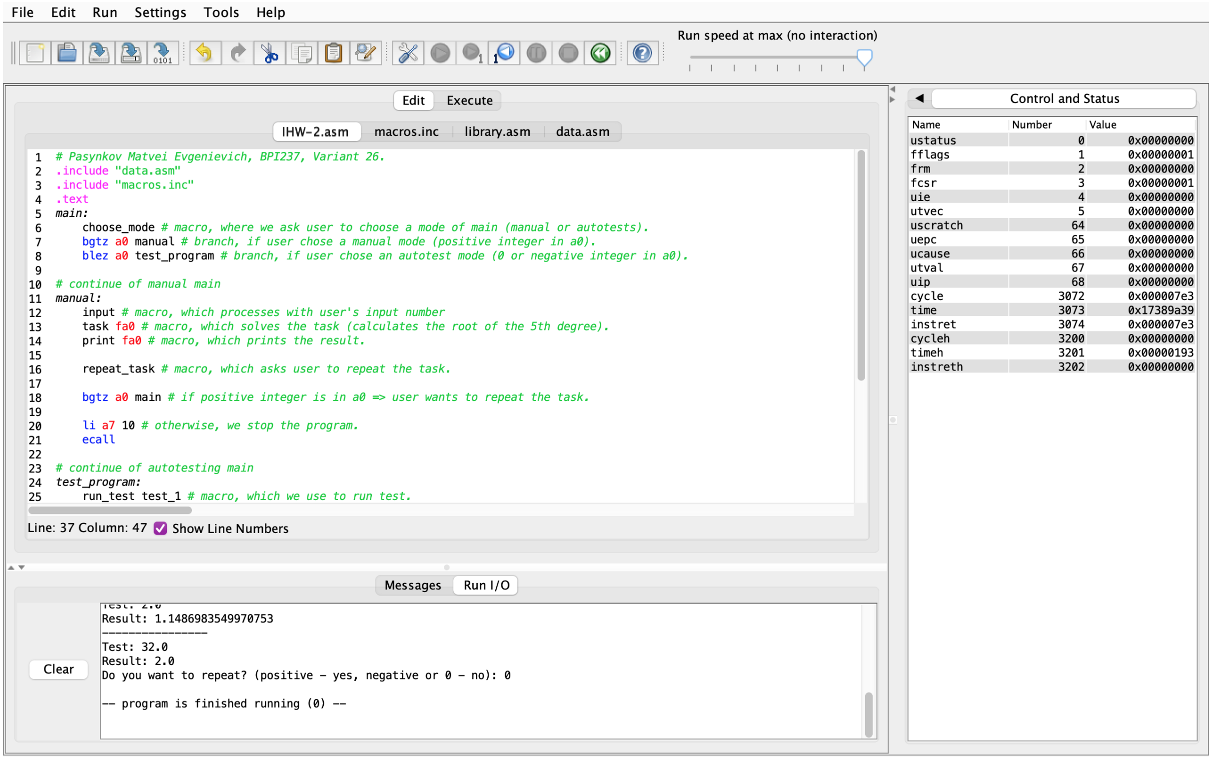
Перед тем, как продемонстрирую результаты тестового покрытия, скажу, что так как я делал работу по критериям до 10 баллов, то согласно критериям на 8 баллов у меня реализовано автоматическое тестирование, поэтому покажу его результаты 🙂, так как в нём реализовано тестовое покрытие.

Результаты тестов:




File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
IHW-2.asm macros.inc library.asm data.asm
Pasynkov Matvei Evgenievich, BPI237, Variant 26.
.include "data.asm"
.include "macros.inc"
.text
main:
choose_mode # macro, where we ask user to choose a mode of main (manual or autotests).
bgtz a0 manual # branch, if user chose a manual mode (positive integer in a0).
blez a0 test_program # branch, if user chose an autotest mode (0 or negative integer in a0).
continue of manual main
manual:
input # macro, which processes with user's input number
task fa0 # macro, which solves the task (calculates the root of the 5th degree).
print fa0 # macro, which prints the result.
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
continue of autotesting main
test_program:
run_test test_1 # macro, which we use to run test.
Line: 37 Column: 47 Show Line Numbers
Messages Run I/O
Clear
Test: -1.0
Result: -1.0000000000000013
Test: 0.0
Result: 0.0
Test: 1.0
Result: 1.0000000000000013
Control and Status


File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
IHW-2.asm macros.inc library.asm data.asm
Pasynkov Matvei Evgenievich, BPI237, Variant 26.
.include "data.asm"
.include "macros.inc"
.text
main:
choose_mode # macro, where we ask user to choose a mode of main (manual or autotests).
bgtz a0 manual # branch, if user chose a manual mode (positive integer in a0).
blez a0 test_program # branch, if user chose an autotest mode (0 or negative integer in a0).
continue of manual main
manual:
input # macro, which processes with user's input number
task fa0 # macro, which solves the task (calculates the root of the 5th degree).
print fa0 # macro, which prints the result.
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
continue of autotesting main
test_program:
run_test test_1 # macro, which we use to run test.
Line: 37 Column: 47 Show Line Numbers
Messages Run I/O
Clear
Test: 1.0
Result: 1.0000000000000013
Test: 2.0
Result: 1.1486983549970753
Test: 32.0
Result: 2.0
Do you want to repeat? (positive - yes, negative or 0 - no): 0
Control and Status



Критерии на 6-7 баллов.

Все реализованы, в качестве подтверждения прикреплю код library.asm, IHW-2.asm (основного файла), macros.inc, data.asm (хранит весь нужный Data Segment).

IHW-2.asm

```
# Pasynkov Matvei Evgenievich, BPI237, Variant 26.
.include "data.asm"
.include "macros.inc"
.text
main:
    choose_mode # macro, where we ask user to choose a mode o
    bgtz a0 manual # branch, if user chose a manual mode (pos
    blez a0 test_program # branch, if user chose an autotest 

# continue of manual main
manual:
    input # macro, which processes with user's input number
    task fa0 # macro, which solves the task (calculates the r
    print fa0 # macro, which prints the result.
```

```
    repeat_task # macro, which asks user to repeat the task.

    bgtz a0 main # if positive integer is in a0 => user wants

    li a7 10 # otherwise, we stop the program.
    ecall

# continue of autotesting main
test_program:
    run_test test_1 # macro, which we use to run test.
    run_test test_2
    run_test test_3
    run_test test_4
    run_test test_5
    run_test test_6
    run_test test_7

    repeat_task # macro, which asks user to repeat the task.

    bgtz a0 main # if positive integer is in a0 => user wants

    li a7 10 # otherwise, we stop the program.
    ecall
```

library.asm

```
.include "macros.inc"
.include "data.asm"

.globl _input _task _print _choose_mode _repeat_task _run_tes

.text
# Programs.
# input: nothing.
# output: fa0 - double number from user (after using syscall
_input:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)
```

```
    la a0 ask_x # asking user for input
    li a7 4 # syscall to print string
    ecall # syscall

    li a7 7 # syscall to get double in a0
    ecall # syscall

    lw ra (sp) # getting return address from stack and return
    addi sp sp 4

    ret

# input: fa0 - double number from user.
# output: fa0 - double number of result (after moving result
_task:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)

    fmv.d ft0 fa0 # moving number from fa0 to ft0

    fld ft1 n t0 # loading n = 5 (because we are calculating
    fld ft2 eps t0 # loading necessary accuracy (epsilon)

    fdiv.d ft3 ft0 ft1 # saving current result in ft3 (we nee
    fmv.d ft4 ft0 # saving previous result in ft4 (we need it
loop:
    fsub.d ft5 ft3 ft4 # checking condition that we have got
    fabs.d ft5 ft5
    flt.d t0 ft5 ft2
    bnez t0 end_loop

    fmv.d ft4 ft3 # implementation of the fast converging ite
    fmul.d ft3 ft4 ft4 # saving in ft3: (x_k) ^ 2
    fmul.d ft3 ft3 ft3 # saving in ft3: (x_k) ^ 4
    fdiv.d ft3 ft0 ft3 # saving in ft3: x / (x_k) ^ 4, where
    fadd.d ft3 ft3 ft4 # saving in ft3: x_k + x / (x_k) ^ 4
    fadd.d ft3 ft3 ft4 # saving in ft3: 2 * x_k + x / (x_k) ^
```

```
    fadd.d ft3 ft3 ft4 # saving in ft3: 3 * x_k + x / (x_k) ^
    fadd.d ft3 ft3 ft4 # saving in ft3: 4 * x_k + x / (x_k) ^
    fdiv.d ft3 ft3 ft1 # saving in ft3: (4 * x_k + x / (x_k)

    fmv.d fa0 ft3 # saving current result (x_(k + 1)) in fa0
    j loop # jumping to the start of loop.
end_loop:
    lw ra (sp) # loading return address from stack.
    addi sp sp 4

    ret # returning.

# input: fa0 - double number of result.
# output: nothing.
_print:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)

    la a0 result # loading address of string "Result: " in a0
    li a7 4 # syscall to print a string
    ecall # syscall

    li a7 3 # syscall to print a double number of result.
    ecall # syscall

    la a0 ln # loading address of string "\n" in a0
    li a7 4 # syscall to print string.
    ecall # syscall.

    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.

# input: nothing
# output: a0 - positive integer - yes, otherwise - no (after
_repeat_task:
    addi sp sp -4 # saving return address.
```

```
    sw ra (sp)

    la a0 ask_repeat # asking user to repeat.
    li a7 4 # syscall to print a string.
    ecall # syscall

    li a7 5 # syscall to read an integer.
    ecall # syscall.

    lw ra (sp) # loading return address
    addi sp sp 4

    ret # returning

# input: nothing
# output: a0 -- positive - manual, negative or 0 - run autote
_choose_mode:
    addi sp sp -4 # saving return address.
    sw ra (sp)

    la a0 ask_mode # asking user about mode.
    li a7 4 # syscall to print string
    ecall # syscall

    li a7 5 # syscall to read integer from user.
    ecall # syscall

    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.

# input: fa0: test_number.
# output: nothing.
_run_test:
    addi sp sp -4 # saving return address.
    sw ra (sp)
```

```
    la a0 test_message # saving test message's address.
    li a7 4 # syscall to write string.
    ecall # syscall

    li a7 3 # syscall to write double number.
    ecall # syscall

    la a0 ln # saving '\n's address.
    li a7 4 # syscall to write string.
    ecall # syscall.

    task fa0 # calling task function.
    print fa0 # calling print function.

    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.
```

macros.inc

```
# Macros

# Macro, which we use to get double from user.
# Input: nothing
# Output: fa0 - double number from user.
.macro input
    jal _input
.end_macro

# Macro, which we use to solve the task.
# Input: FPU register, which stores double number from user.
# Output: fa0 - double number of result.
.macro task %reg
    fmv.d fa0 %reg
    jal _task
.end_macro
```

```
# Macro, which we use to print result's double number.
# Input: FPU register, which stores double number with result
# Output: result on display.
.macro print %reg
    fmv.d fa0 %reg
    jal _print
.end_macro

# Macro, which we use to repeat main.
# Input: nothing.
# Output: a0 - positive integer - yes, otherwise - no.
.macro repeat_task
    jal _repeat_task
.end_macro

# Macro, which we use to ask the user to choose a mode.
# Input: nothing.
# Output: a0 -- positive - manual, negative or 0 - run autote
.macro choose_mode
    jal _choose_mode
.end_macro

# Macro, which we use to run test.
# Input: label of test in Data Segment.
# Output: result of test on display.
.macro run_test %label_test
    fld fa0 %label_test t0
    jal _run_test
.end_macro
```

data.asm

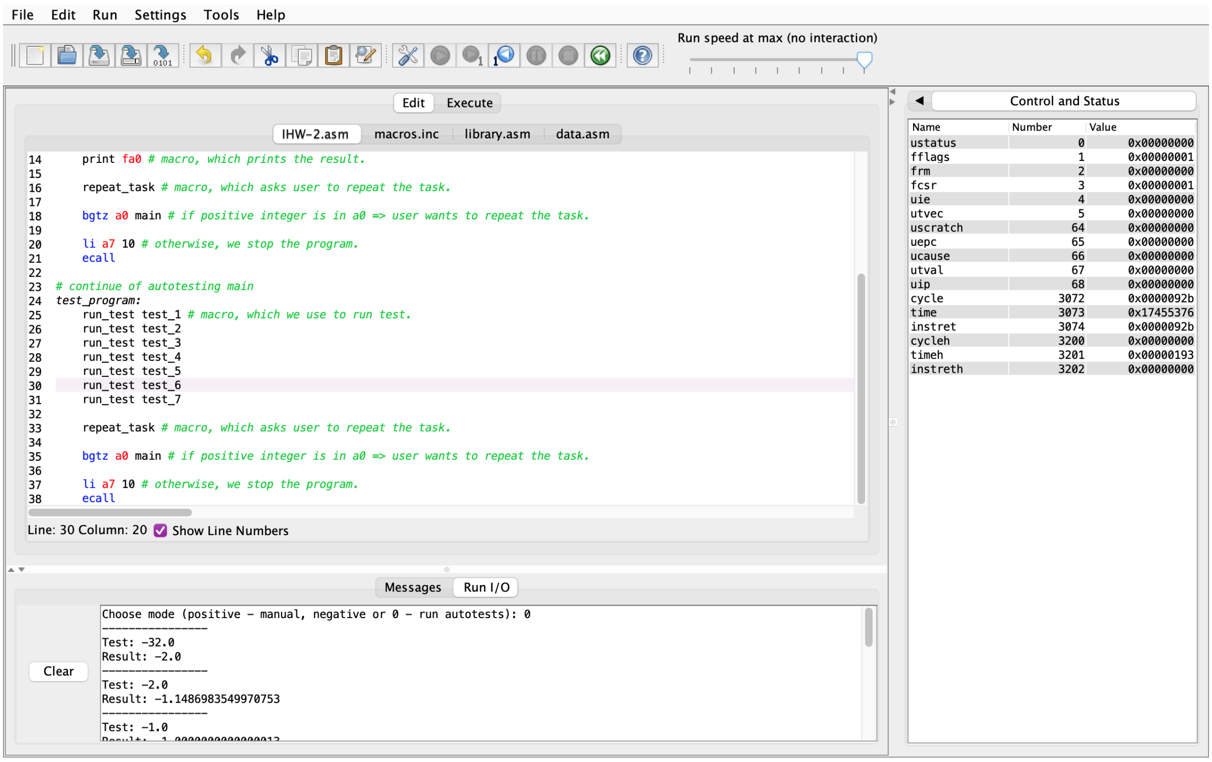
```
#Data Segment, for IHW-2.asm, library.asm.
.data
ask_x: .asciz "Enter x: "
result: .asciz "Result: "
ask_repeat: .asciz "Do you want to repeat? (positive - yes, n
ask_mode: .asciz "Choose mode (positive - manual, negative or
```

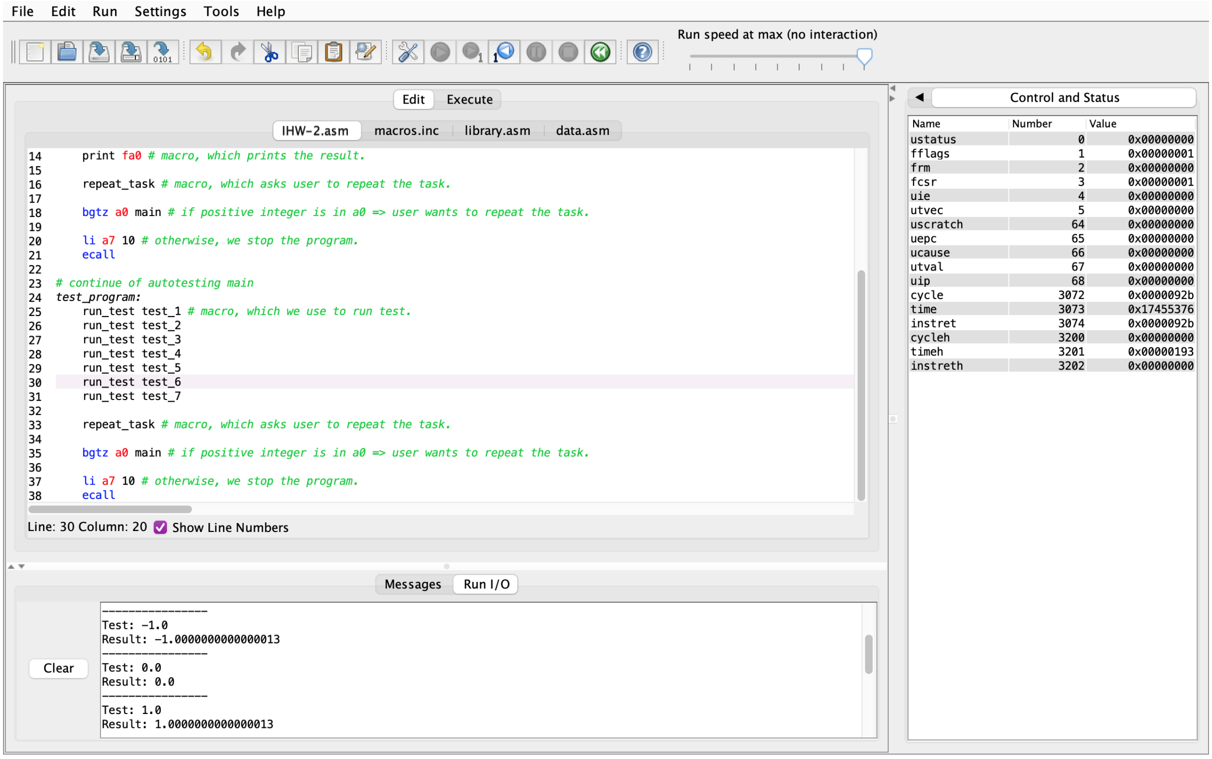
```
ln: .asciz "\n"
n: .double 5.0
eps: .double 0.000001

test_1: .double -32
test_2: .double -2
test_3: .double -1
test_4: .double 0
test_5: .double 1
test_6: .double 2
test_7: .double 32
test_message: .asciz "---------------\nTest: "
```

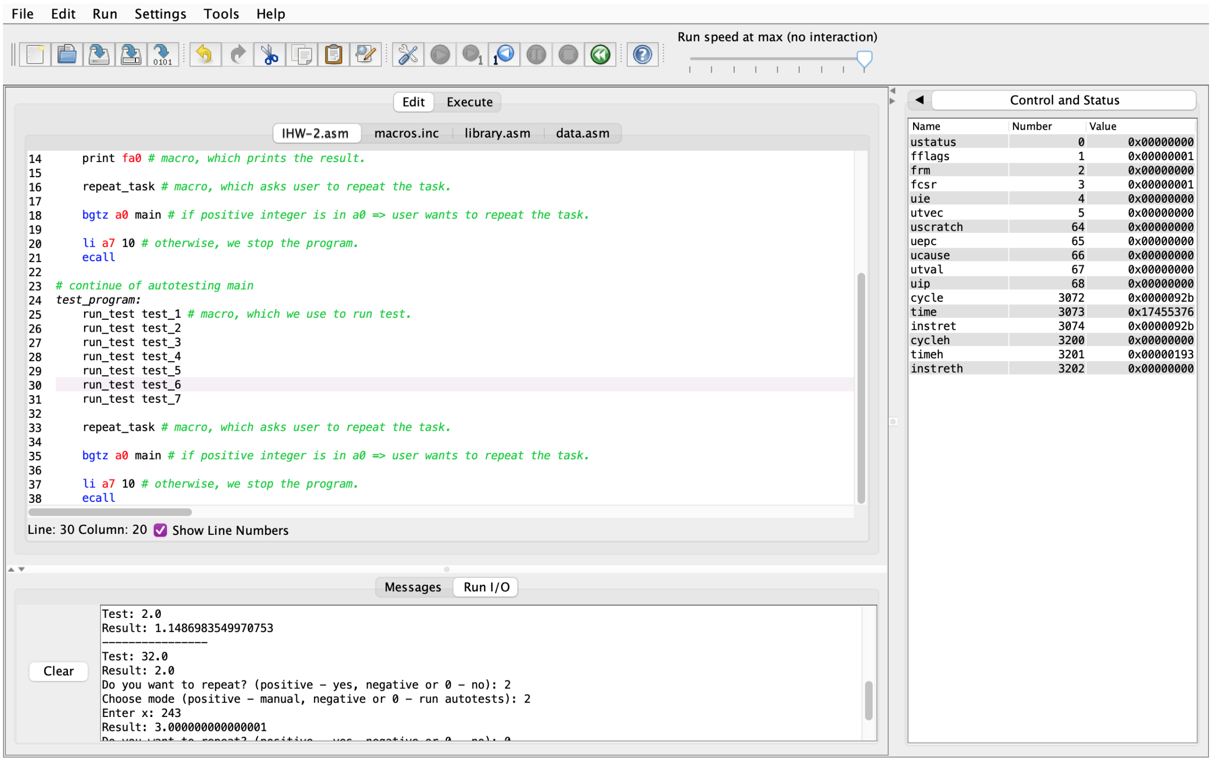
Критерии на 8 баллов.

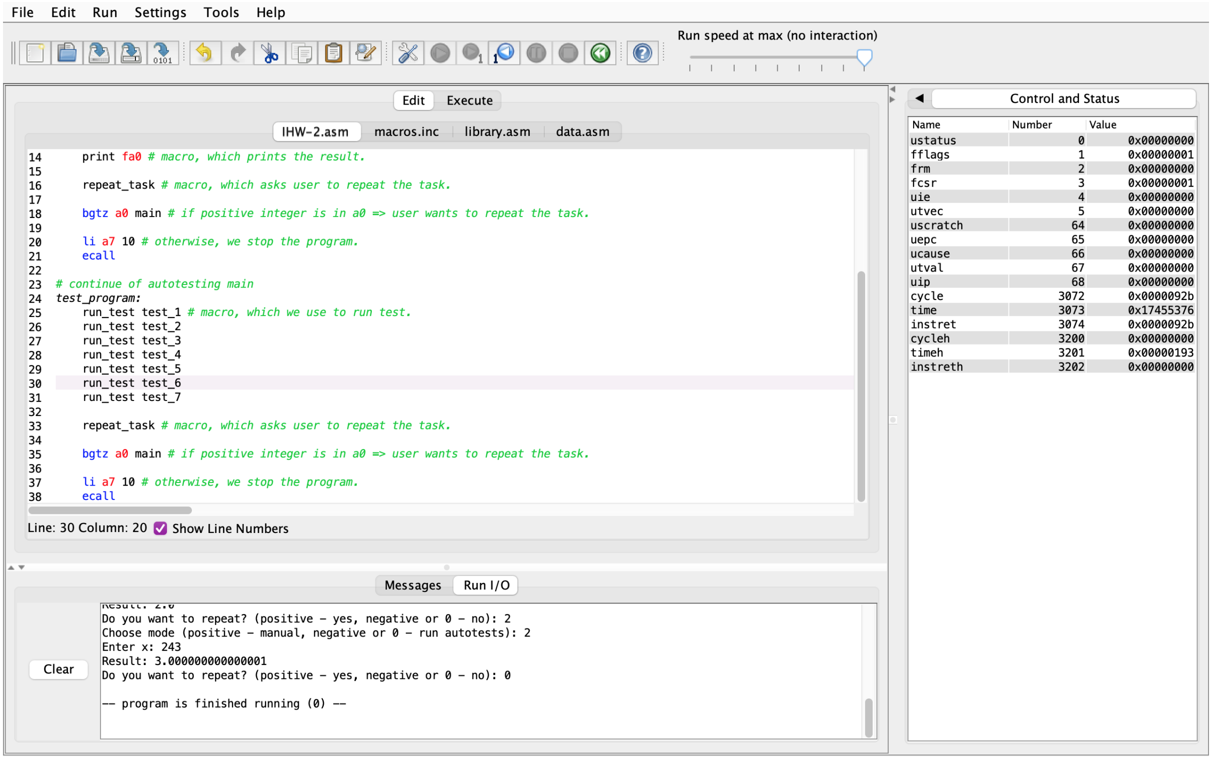
Автоматическое тестирование, многократное использование программы, а также код для проверки работы на высокоуровневом языке есть.

Их реализация представлена в следующих скриншотах:




File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
IHW-2.asm macros.inc library.asm data.asm
print fa0 # macro, which prints the result.
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
continue of autotesting main
test_program:
run_test test_1 # macro, which we use to run test.
run_test test_2
run_test test_3
run_test test_4
run_test test_5
run_test test_6
run_test test_7
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
Line: 30 Column: 20 Show Line Numbers
Messages Run I/O
Clear
Test: -1.0
Result: -1.0000000000000013
Test: 0.0
Result: 0.0
Test: 1.0
Result: 1.0000000000000013
Control and Status
Name Number Value
ustatus 0 0x00000000
fflags 1 0x00000001
frm 2 0x00000000
fcsr 3 0x00000001
uie 4 0x00000000
utvec 5 0x00000000
uscratch 64 0x00000000
uepc 65 0x00000000
ucause 66 0x00000000
utval 67 0x00000000
uip 68 0x00000000
cycle 3072 0x0000092b
time 3073 0x17455376
instret 3074 0x0000092b
cycleh 3200 0x00000000
timeh 3201 0x00000193
instreth 3202 0x00000000

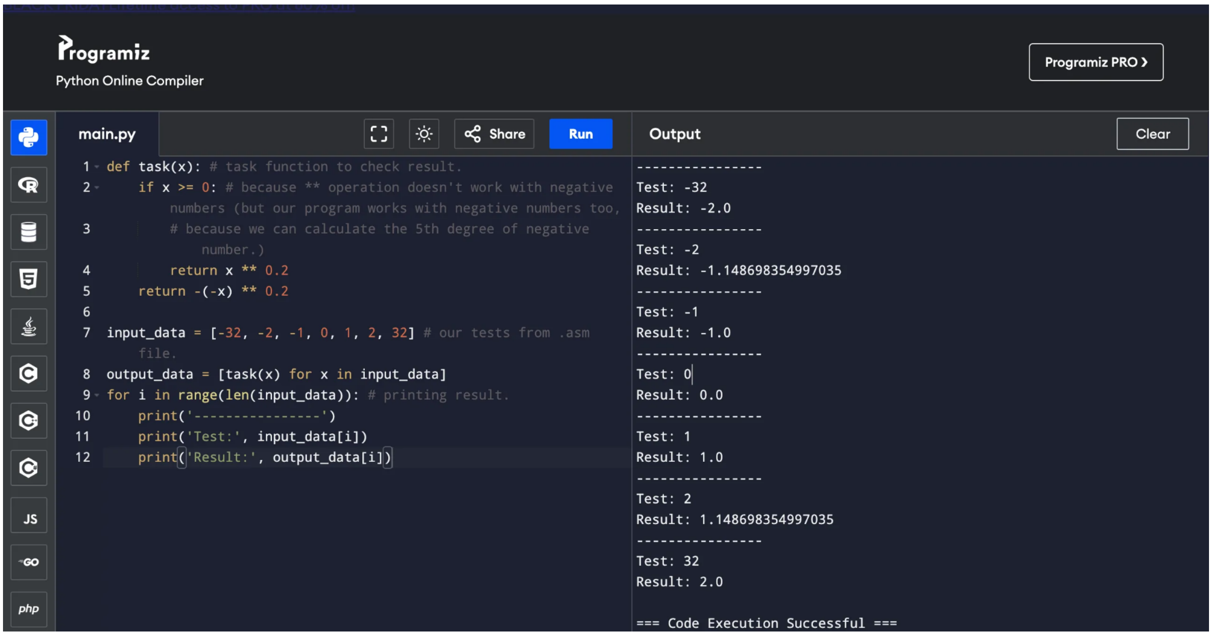

File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
IHW-2.asm macros.inc library.asm data.asm
print fa0 # macro, which prints the result.
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
continue of autotesting main
test_program:
run_test test_1 # macro, which we use to run test.
run_test test_2
run_test test_3
run_test test_4
run_test test_5
run_test test_6
run_test test_7
repeat_task # macro, which asks user to repeat the task.
bgtz a0 main # if positive integer is in a0 => user wants to repeat the task.
li a7 10 # otherwise, we stop the program.
ecall
Line: 30 Column: 20 Show Line Numbers
Messages Run I/O
Clear
Test: 2.0
Result: 1.1486983549970753
Test: 32.0
Result: 2.0
Do you want to repeat? (positive - yes, negative or 0 - no): 2
Choose mode (positive - manual, negative or 0 - run autotests): 2
Enter x: 243
Result: 3.000000000000001
Control and Status
Name Number Value
ustatus 0 0x00000000
fflags 1 0x00000001
frm 2 0x00000000
fcsr 3 0x00000001
uie 4 0x00000000
utvec 5 0x00000000
uscratch 64 0x00000000
uepc 65 0x00000000
ucause 66 0x00000000
utval 67 0x00000000
uip 68 0x00000000
cycle 3072 0x0000092b
time 3073 0x17455376
instret 3074 0x0000092b
cycleh 3200 0x00000000
timeh 3201 0x00000193
instreth 3202 0x00000000



А также код высокоуровневый код:

```
def task(x): # task function to check result.
    if x >= 0: # because ** operation doesn't work with negat
        # because we can calculate the 5th degree of negative
        return x ** 0.2
    return -(-x) ** 0.2

input_data = [-32, -2, -1, 0, 1, 2, 32] # our tests from .asm
output_data = [task(x) for x in input_data]
for i in range(len(input_data)): # printing result.
    print('----------------')
    print('Test:', input_data[i])
    print('Result:', output_data[i])
```

Результаты:



Как мы видим, результаты нашего кода не хуже результатов на Python, чем на 0.001, что от нас и требовалось в условии.

Критерии на 9 баллов.

Макросы реализованы в macros.inc.

```
# Macros

# Macro, which we use to get double from user.
# Input: nothing
# Output: fa0 - double number from user.
.macro input
    jal _input
.end_macro

# Macro, which we use to solve the task.
# Input: FPU register, which stores double number from user.
# Output: fa0 - double number of result.
.macro task %reg
    fmv.d fa0 %reg
    jal _task
.end_macro

# Macro, which we use to print result's double number.
# Input: FPU register, which stores double number with result
```

```
# Output: result on display.
.macro print %reg
    fmv.d fa0 %reg
    jal _print
.end_macro

# Macro, which we use to repeat main.
# Input: nothing.
# Output: a0 - positive integer - yes, otherwise - no.
.macro repeat_task
    jal _repeat_task
.end_macro

# Macro, which we use to ask the user to choose a mode.
# Input: nothing.
# Output: a0 -- positive - manual, negative or 0 - run autote
.macro choose_mode
    jal _choose_mode
.end_macro

# Macro, which we use to run test.
# Input: label of test in Data Segment.
# Output: result of test on display.
.macro run_test %label_test
    fld fa0 %label_test t0
    jal _run_test
.end_macro
```

Критерии на 10 баллов.

Декомпозиция на несколько файлов реализована согласно критериям.

Файлы:

IHW-2.asm

```
# Pasynkov Matvei Evgenievich, BPI237, Variant 26.
.include "data.asm"
.include "macros.inc"
.text
```

```
main:
    choose_mode # macro, where we ask user to choose a mode o
    bgtz a0 manual # branch, if user chose a manual mode (pos
    blez a0 test_program # branch, if user chose an autotest r

# continue of manual main
manual:
    input # macro, which processes with user's input number
    task fa0 # macro, which solves the task (calculates the r
    print fa0 # macro, which prints the result.

    repeat_task # macro, which asks user to repeat the task.

    bgtz a0 main # if positive integer is in a0 => user wants

    li a7 10 # otherwise, we stop the program.
    ecall

# continue of autotesting main
test_program:
    run_test test_1 # macro, which we use to run test.
    run_test test_2
    run_test test_3
    run_test test_4
    run_test test_5
    run_test test_6
    run_test test_7

    repeat_task # macro, which asks user to repeat the task.

    bgtz a0 main # if positive integer is in a0 => user wants

    li a7 10 # otherwise, we stop the program.
    ecall
```

library.asm

```
.include "macros.inc"
.include "data.asm"

.globl _input _task _print _choose_mode _repeat_task _run_tes

.text
# Programs.
# input: nothing.
# output: fa0 - double number from user (after using syscall
_input:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)

    la a0 ask_x # asking user for input
    li a7 4 # syscall to print string
    ecall # syscall

    li a7 7 # syscall to get double in a0
    ecall # syscall

    lw ra (sp) # getting return address from stack and return
    addi sp sp 4

    ret

# input: fa0 - double number from user.
# output: fa0 - double number of result (after moving result
_task:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)

    fmv.d ft0 fa0 # moving number from fa0 to ft0

    fld ft1 n t0 # loading n = 5 (because we are calculating
    fld ft2 eps t0 # loading necessary accuracy (epsilon)

    fdiv.d ft3 ft0 ft1 # saving current result in ft3 (we nee
    fmv.d ft4 ft0 # saving previous result in ft4 (we need it
```

```
loop:
    fsub.d ft5 ft3 ft4 # checking condition that we have got
    fabs.d ft5 ft5
    flt.d t0 ft5 ft2
    bnez t0 end_loop

    fmv.d ft4 ft3 # implementation of the fast converging ite
    fmul.d ft3 ft4 ft4 # saving in ft3: (x_k) ^ 2
    fmul.d ft3 ft3 ft3 # saving in ft3: (x_k) ^ 4
    fdiv.d ft3 ft0 ft3 # saving in ft3: x / (x_k) ^ 4, where
    fadd.d ft3 ft3 ft4 # saving in ft3: x_k + x / (x_k) ^ 4
    fadd.d ft3 ft3 ft4 # saving in ft3: 2 * x_k + x / (x_k) ^
    fadd.d ft3 ft3 ft4 # saving in ft3: 3 * x_k + x / (x_k) ^
    fadd.d ft3 ft3 ft4 # saving in ft3: 4 * x_k + x / (x_k) ^
    fdiv.d ft3 ft3 ft1 # saving in ft3: (4 * x_k + x / (x_k)

    fmv.d fa0 ft3 # saving current result (x_(k + 1)) in fa0
    j loop # jumping to the start of loop.
end_loop:
    lw ra (sp) # loading return address from stack.
    addi sp sp 4

    ret # returning.

# input: fa0 - double number of result.
# output: nothing.
_print:
    addi sp sp -4 # saving return address on the stack.
    sw ra (sp)

    la a0 result # loading address of string "Result: " in a0
    li a7 4 # syscall to print a string
    ecall # syscall

    li a7 3 # syscall to print a double number of result.
    ecall # syscall

    la a0 ln # loading address of string "\n" in a0
```

```
    li a7 4 # syscall to print string.
    ecall # syscall.

    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.

# input: nothing
# output: a0 - positive integer - yes, otherwise - no (after
_repeat_task:
    addi sp sp -4 # saving return address.
    sw ra (sp)

    la a0 ask_repeat # asking user to repeat.
    li a7 4 # syscall to print a string.
    ecall # syscall

    li a7 5 # syscall to read an integer.
    ecall # syscall.

    lw ra (sp) # loading return address
    addi sp sp 4

    ret # returning

# input: nothing
# output: a0 -- positive - manual, negative or 0 - run autote
_choose_mode:
    addi sp sp -4 # saving return address.
    sw ra (sp)

    la a0 ask_mode # asking user about mode.
    li a7 4 # syscall to print string
    ecall # syscall

    li a7 5 # syscall to read integer from user.
    ecall # syscall
```

```
    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.

# input: fa0: test_number.
# output: nothing.
_run_test:
    addi sp sp -4 # saving return address.
    sw ra (sp)

    la a0 test_message # saving test message's address.
    li a7 4 # syscall to write string.
    ecall # syscall

    li a7 3 # syscall to write double number.
    ecall # syscall

    la a0 ln # saving '\n's address.
    li a7 4 # syscall to write string.
    ecall # syscall.

    task fa0 # calling task function.
    print fa0 # calling print function.

    lw ra (sp) # loading return address.
    addi sp sp 4

    ret # returning.
```

Макросы в macros.inc:

```
# Macros

# Macro, which we use to get double from user.
# Input: nothing
```

```
# Output: fa0 - double number from user.
.macro input
    jal _input
.end_macro

# Macro, which we use to solve the task.
# Input: FPU register, which stores double number from user.
# Output: fa0 - double number of result.
.macro task %reg
    fmv.d fa0 %reg
    jal _task
.end_macro

# Macro, which we use to print result's double number.
# Input: FPU register, which stores double number with result
# Output: result on display.
.macro print %reg
    fmv.d fa0 %reg
    jal _print
.end_macro

# Macro, which we use to repeat main.
# Input: nothing.
# Output: a0 - positive integer - yes, otherwise - no.
.macro repeat_task
    jal _repeat_task
.end_macro

# Macro, which we use to ask the user to choose a mode.
# Input: nothing.
# Output: a0 -- positive - manual, negative or 0 - run autote
.macro choose_mode
    jal _choose_mode
.end_macro

# Macro, which we use to run test.
# Input: label of test in Data Segment.
# Output: result of test on display.
```

```
.macro run_test %label_test
    fld fa0 %label_test t0
    jal _run_test
.end_macro
```

data.asm

```
#Data Segment, for IHW-2.asm, library.asm.
.data
ask_x: .asciz "Enter x: "
result: .asciz "Result: "
ask_repeat: .asciz "Do you want to repeat? (positive - yes, n
ask_mode: .asciz "Choose mode (positive - manual, negative or
ln: .asciz "\n"
n: .double 5.0
eps: .double 0.000001

test_1: .double -32
test_2: .double -2
test_3: .double -1
test_4: .double 0
test_5: .double 1
test_6: .double 2
test_7: .double 32
test_message: .asciz "----------------\nTest: "
```