

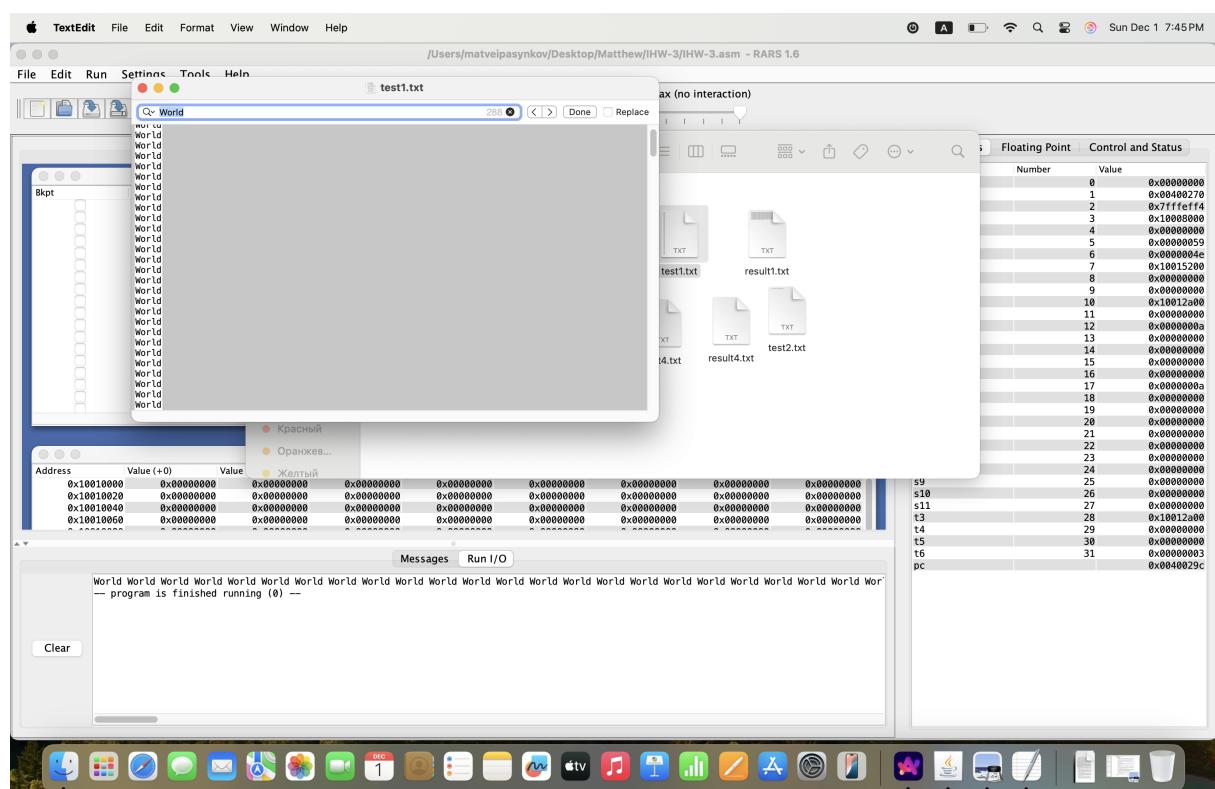
Отчёт по ИДЗ-3

Пасынков Матвей Евгеньевич, БПИ237, Вариант 30

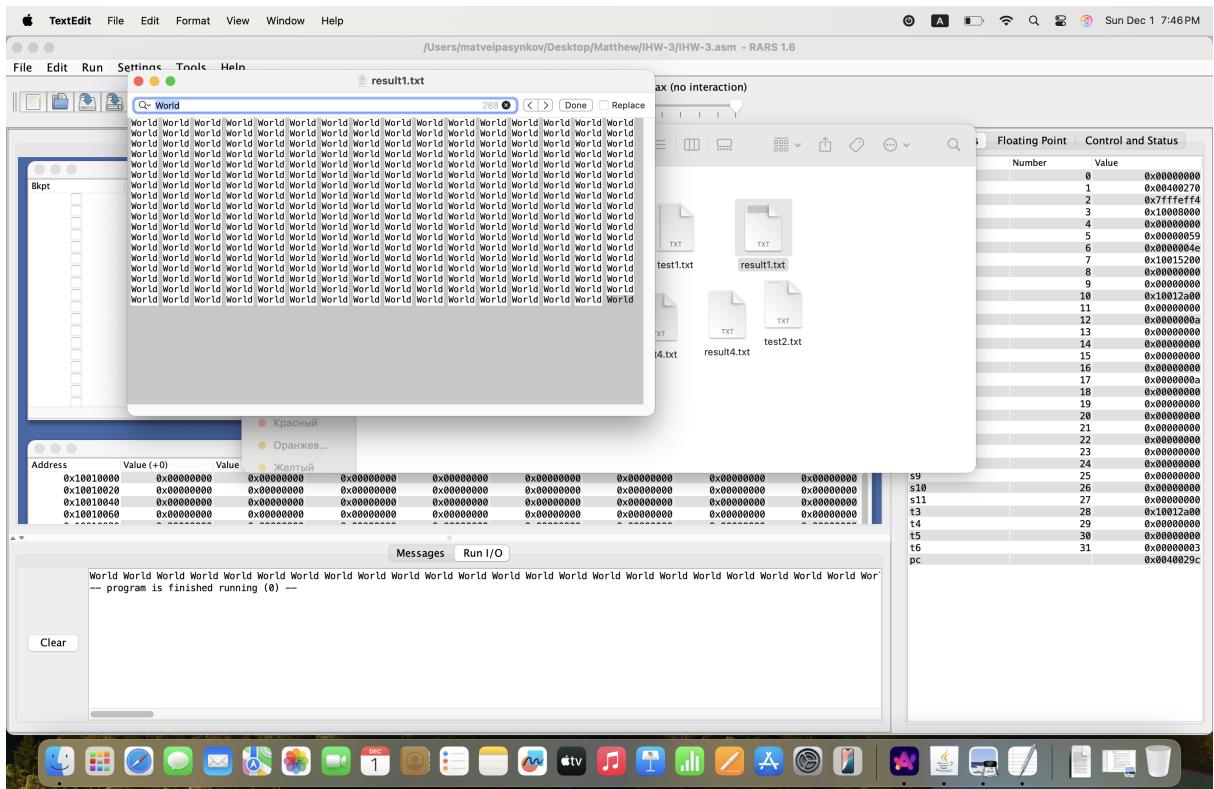
Осуществление тестовых прогонов, которое требуется для 4 - 5 баллов (ввод осуществляется через графические окна, а не консоль из-за критериев на 10, о которых я расскажу позднее).

Тест первый (test1.txt, результат будем сохранять в result1.txt)

test1.txt (файл состоит из одних слов World - их 288 штук).

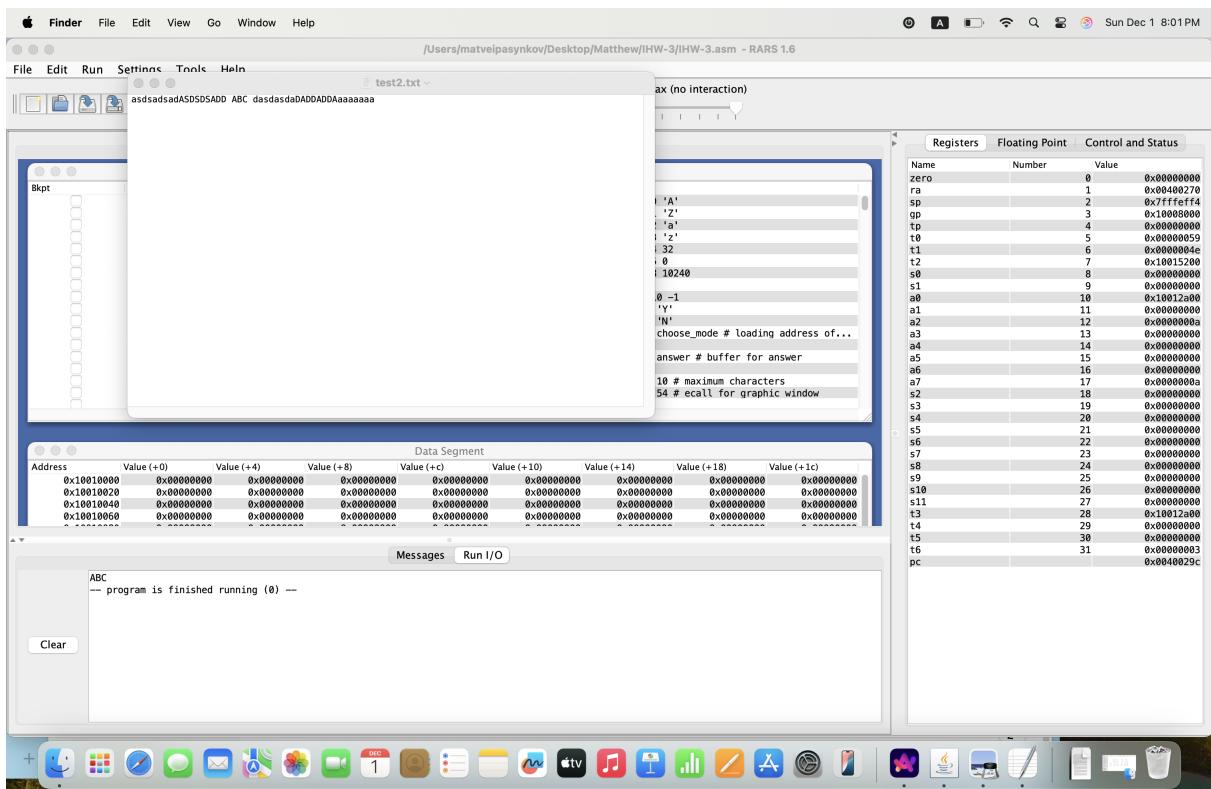


result1.txt (получаем 288 слов World)

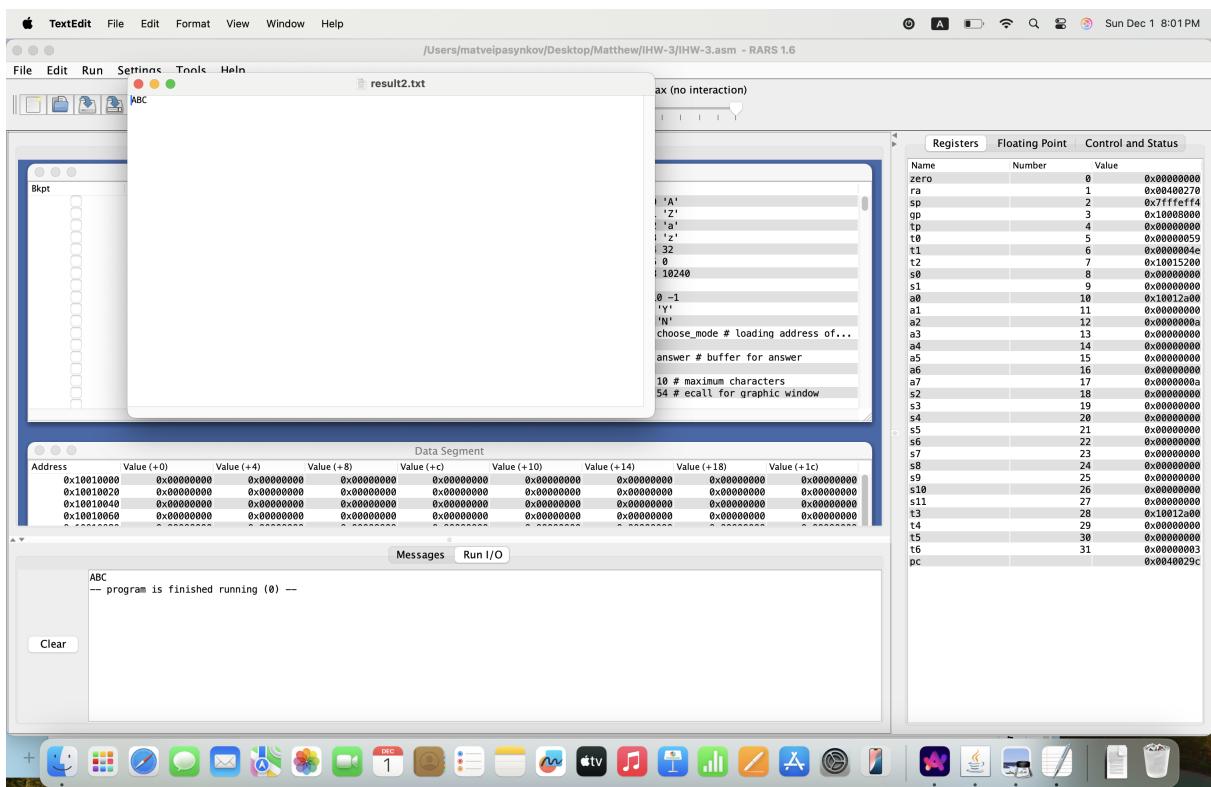


Тест второй (test2.txt, результат будем сохранять в result2.txt)

test2.txt (файл состоит трёх слов, где только одно из них начинается с большой буквы)

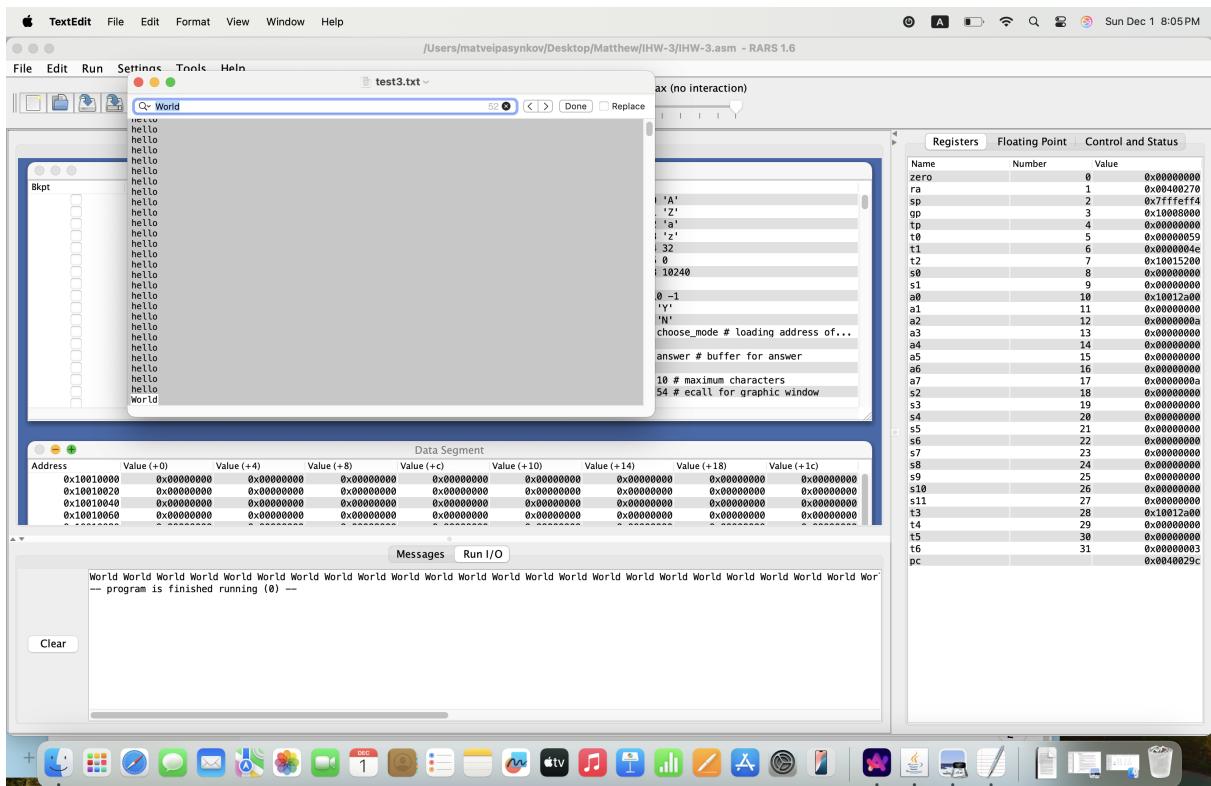


result2.txt (как раз получаем нужное одно слово)

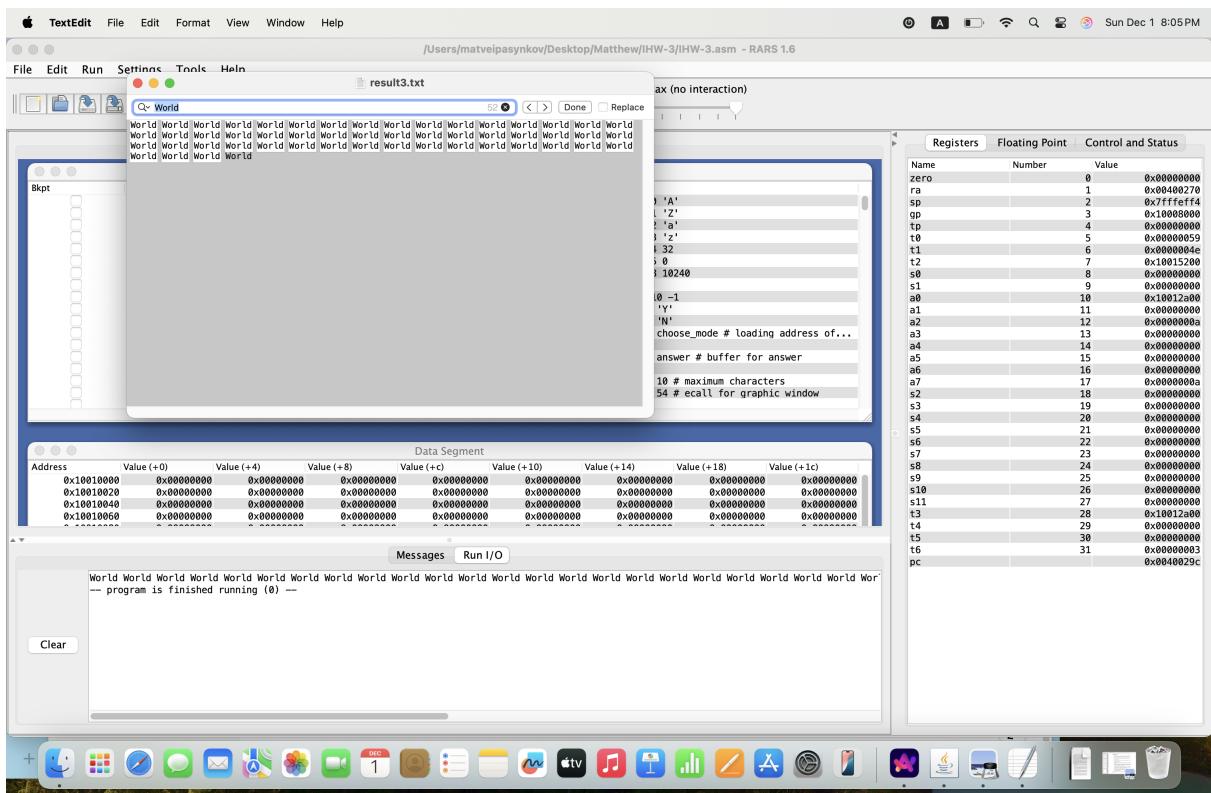


Тест третий (test3.txt, результат будем сохранять в result3.txt)

test3.txt (файл состоит в основном из двух слов: World (52 штуки) и hello)

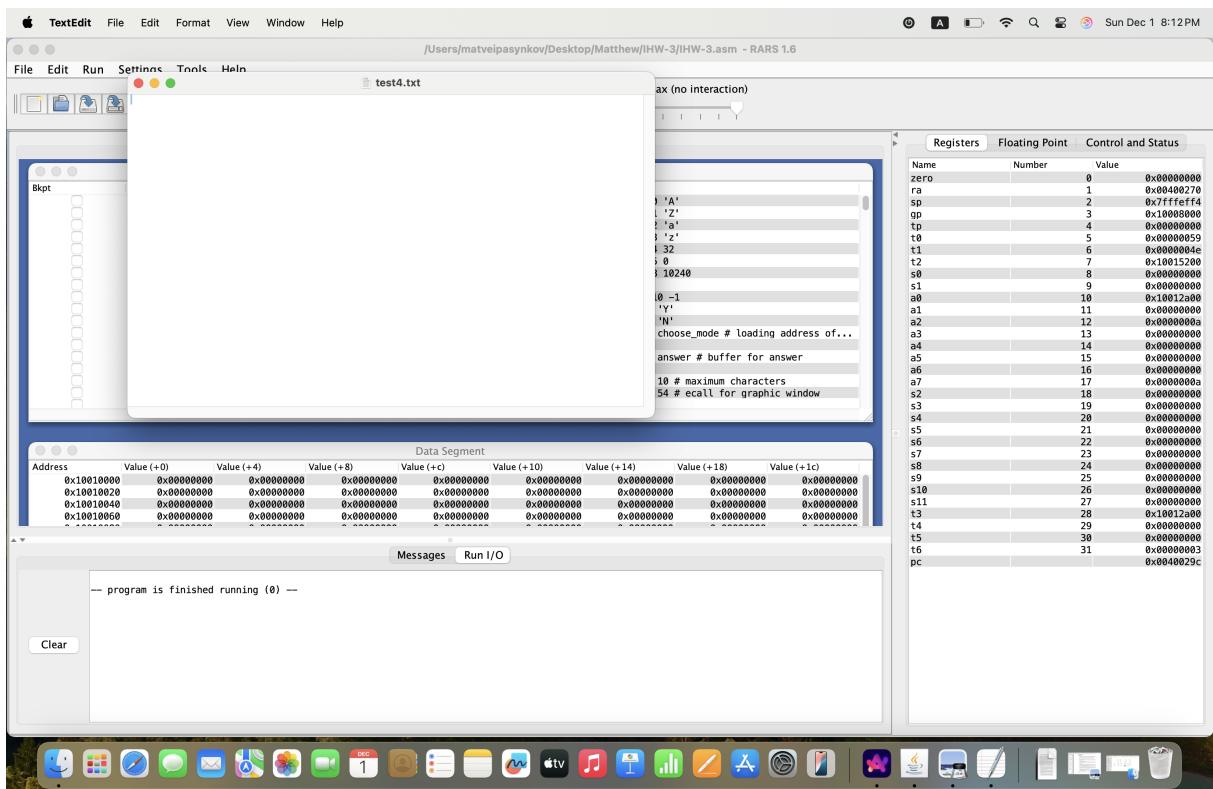


result3.txt (получаем 52 слова World)

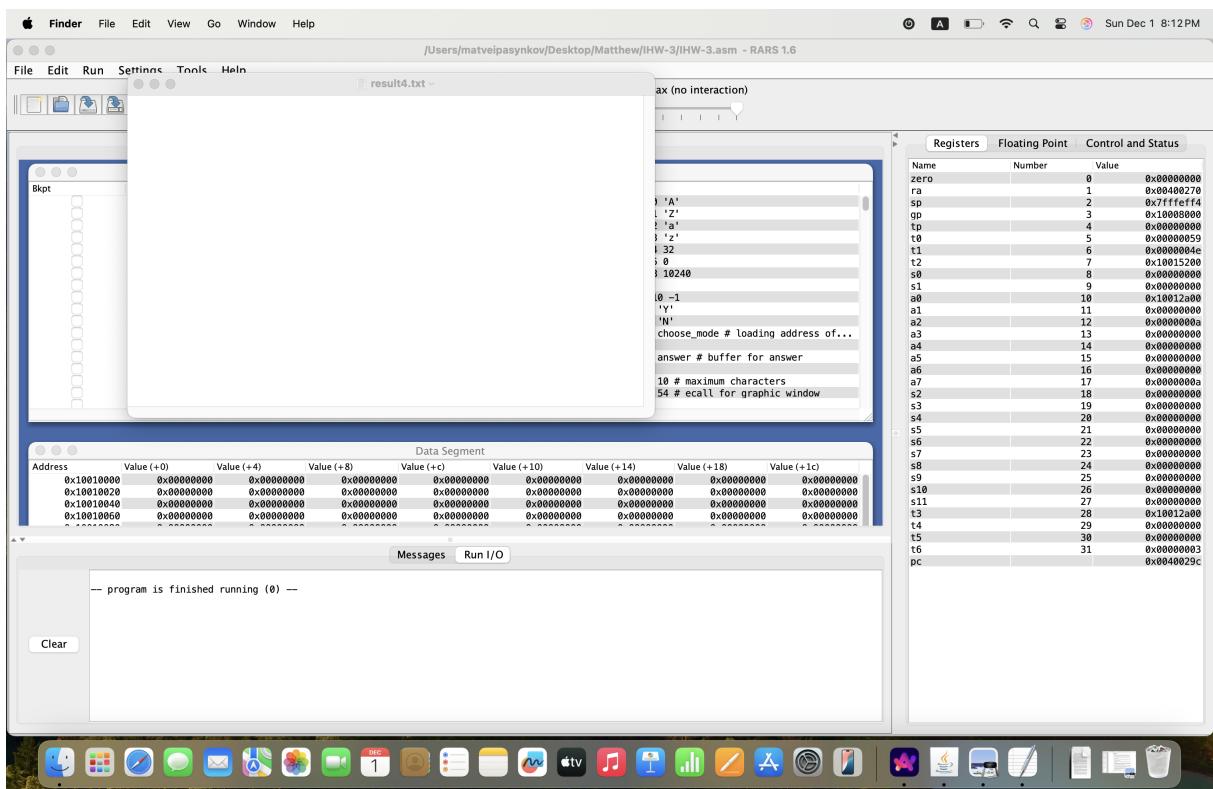


Тест четвёртый (test4.txt, результат будем сохранять в result3.txt)

test4.txt (пустой файл)



result4.txt (пустой файл)



Дополнение программы критериями на 6 - 7 баллов

Все перечисленные критерии выполнены в данных файлах:

IHW-3.asm

```
# Pasynkov Matvei Evgenievich, Variant 30
.include "macros.inc"
.include "data.asm"

.text
.globl prog_start, manual

prog_start:
    load_constants # loading necessary constants.
    li t0 'Y'
    li t1 'N'

    la a0 choose_mode # loading address of message with question
    la a1 answer # buffer for answer
    li a2 10 # maximum characters
    li a7 54 # ecall for graphic window
    ecall
    li a0 0

    lb a0 answer # loading first character from answer.

    beq a0 t0 manual # if answer is yes => we are going to make auto tests
    # otherwise, we are going to autotests.

auto_tests:
    la a0 test_path_1 # loading first test
    la a1 data
    run_test_read a0 a1 # program for reading test file

    la a0 data
    la a1 output_data
    process_string a0 a1 # program for solving task.

    la a0 output_test_path1
```

```
la a1 output_data
run_test_write a0 a1 # program for writing test file

la a0 data
la a1 output_data
clear_data a0 a1 # program for cleating buffers

la a1 data
la a0 test_path_2 # loading second test
run_test_read a0 a1

la a0 data
la a1 output_data
process_string a0 a1

la a0 output_test_path2
la a1 output_data
run_test_write a0 a1

la a0 data
la a1 output_data
clear_data a0 a1

la a0 test_path_3 # loading third test
la a1 data
run_test_read a0 a1

la a0 data
la a1 output_data
process_string a0 a1

la a0 output_test_path3
la a1 output_data
run_test_write a0 a1

la a0 data
la a1 output_data
clear_data a0 a1
```

```
la a0 test_path_4 # loading fourth test
la a1 data
run_test_read a0 a1

la a0 data
la a1 output_data
process_string a0 a1

la a0 output_test_path4
la a1 output_data
run_test_write a0 a1

la a0 data
la a1 output_data
clear_data a0 a1

li a7 10 # stopping program.
ecall

manual:
    la a0 data # loading buffer for saving input data
    input a0 # reading file

    bgtz a0 end_main # if the filename is incorrect, stop the

    la a0, data
    la a1, output_data
    process_string a0 a1 # program for solving task.

    la a0 output_data
    print a0 # saving result

    la a0 output_data
    print_in_console a0 # program for printing in console.

    bgtz a0 end_main
```

```
end_main:  
    # Завершение программы  
    clear_constants # clearing constants.  
    li a7, 10          # syscall for stopping program.  
    ecall
```

lib.asm

```
.include "data.asm"  
.include "macros.inc"  
  
.text  
.globl _input, _print, _process_string, _remove_newline_prog,  
# Program for taking input data from user's file.  
# input: a0 - address of data, which stores input from user.  
# output: nothing.  
_input:  
    mv t2 a0  
    addi sp sp -4  
    sw ra (sp)  
  
    la a0 write_input_path  
    la a1 input_path  
    li a2 512  
    li a7 54  
    ecall  
  
    la a0, input_path  
    lb t0, 0(a0)  
    li t1, 10  
  
    la a0 input_path  
    li a1 0  
    remove_newline_prog a0 a1  
  
    li a7, 1024  
    la a0, input_path
```

```
    li    a1, 0
    ecall
    mv    t6, a0

    beq a0 s10 error_reading

    li t0 0

read_loop:
    mv a0 t6
    la a1 buffer
    li a2 512
    li a7 63
    ecall

    bltz a0 error_reading
    beqz a0 end_loop

    add t0 t0 a0
    mv t1 a0

    la t3 buffer
    bge t0 s8 end_loop
saving_data_loop:
    beqz t1 read_loop
    lb t4 (t3)
    sb t4 (t2)

    addi t3 t3 1
    addi t2 t2 1
    addi t1 t1 -1
    j saving_data_loop
error_reading: # if error occurred.
    la a0 error_path
    li a1 0
    li a7 55
    ecall
```

```
lw ra (sp)
addi sp sp 4

li a0 1

ret
end_loop:
    li    a7, 57
    mv    a0, t6
    ecall

lw ra (sp)
addi sp sp 4

li a0 0

ret
# Program for printing result data and saving in user's file.
# input: a0 - address of data, which stores result.
# output: nothing.
_print:
    mv t2 a0
    addi sp sp -4
    sw ra (sp)

    la a0 write_output_path
    la a1 output_path
    li a2 512
    li a7 54
    ecall

    la a0 output_path
    li a1 1
    remove_newline_prog a0 a1

    li    a7, 1024
    la    a0, output_path
    li    a1, 1
```

```
ecall
mv t6, a0

beq a0 s10 error_writing

li t0 0
li t1 512
la t3 buffer
mv t5 s8
loading_data_loop:
beqz t1 write_loop
beqz t5 write_loop

lb t4 (t2)
sb t4 (t3)

addi t3 t3 1
addi t2 t2 1
addi t1 t1 -1
addi t5 t5 -1
j loading_data_loop
write_loop:
mv a0 t6
la a1 buffer
li a2 512
li a7 64
ecall

bltz a0 error_writing
beqz a0 end_write_loop

li t1 512
la t3 buffer

blez t5 end_write_loop

j loading_data_loop
error_writing: # if error occurred.
```

```
la a0 error_path
li a1 0
li a7 55
ecall

lw ra (sp)
addi sp sp 4

li a0 1

ret
end_write_loop:
li a7 57
mv a0 t6
ecall

lw ra (sp)
addi sp sp 4

li a0 0

ret

# Program for solving task.
# input: a0 - address of data, which stores input from user,
# output: nothing.
_process_string:
addi sp sp -4
sw ra (sp)

li t3 0
li t4 0
process_loop:
addi t3 t3 1
lb t1, 0(a0)
beq t1, zero, finish

blt t1 s2 else
```

```

ble t1 s3 if_word_started
j else
if_word_started:
    li t4 1
    j continue_loop
else:
    blt t1 s0 continue_else
    ble t1 s1 continue_loop
continue_else:
    li t4 0
continue_loop:
    # checking the letter
    blt t1, s0, not_letter    # if symbol is less than 'A' - i
    bgt t1, s1, not_letter    # if symbol is greater than 'Z'
    bgtz t4 not_letter # if word started -> skip

    # copy word
    beqz zero, copy_word
    bge t3 s8 finish

    j next_char

not_letter:
    j next_char

copy_word:
copy_loop:
    addi t3 t3 1

    sb t1, 0(a1)
    addi a1, a1, 1
    addi a0, a0, 1
    lb t1, 0(a0)

    blt t1, s0, word_end
    ble t1, s1, copy_loop
    bgt t1, s3, word_end
    blt t1, s2, word_end

```

```

    bge t3 s8 finish
    j copy_loop

word_end:
    sb s4, 0(a1)
    addi a1, a1, 1
    j process_loop

next_char:
    addi a0, a0, 1
    j process_loop

finish:
    addi a1, a1, -1
    sb s5, 0(a1)

    lw ra (sp)
    addi sp sp 4

    ret
# Program for removing '\n' from user's name of file
# input: a0 - user's path, a1 - type of file (0 - input, 1 -
# output: nothing.
_remove_newline_prog:
    addi sp sp -4
    sw ra (sp)
remove_newline:
    beq t0, t1, end_remove_newline
    addi a0, a0, 1
    lb t0, 0(a0)
    bnez t0, remove_newline
end_remove_newline:
    beqz a1 remove_newline_from_input
    sb zero -1(a0)
    j return_from_remove_newline
remove_newline_from_input:
    sb zero, (a0)
return_from_remove_newline:

```

```
lw ra (sp)
addi sp sp 4

ret
# Program for printing result in console.
# input: a0 - result's data.
# output: nothing.
_print_in_console:
    mv t3 a0
    addi sp sp -4
    sw ra (sp)

    li t0 'Y'
    li t1 'N'

    la a0 ask_to_display_results
    la a1 answer
    li a2 10
    li a7 54
    ecall

    li a0 0

    lb a0 answer

    beq a0 t0 agree_to_print
    beq a0 t1 end_display_on_console

    la a0 incorrect_answer
    li a7 4
    ecall
    j end_display_on_console

agree_to_print:
    mv a0 t3
    li a7 4
    ecall
```

```
end_display_on_console:
    lw ra (sp)
    addi sp sp -4
    ret
# Program for reading test file.
# input: a0 - test's path, a1 - address of data, which will s
# output: nothing.
_run_test_read:
    mv t2 a1

    addi sp sp -4
    sw ra (sp)

    li a7, 1024
    li a1, 0
    ecall
    mv t6, a0

    beq a0 s10 run_test_error_reading

    li t0 0

run_test_read_loop:
    mv a0 t6
    la a1 buffer
    li a2 512
    li a7 63
    ecall

    bltz a0 run_test_error_reading
    beqz a0 run_test_end_loop

    add t0 t0 a0
    mv t1 a0

    la t3 buffer
    bge t0 s8 run_test_end_loop
run_test_saving_data_loop:
```

```

beqz t1 run_test_read_loop
lb t4 (t3)
sb t4 (t2)

addi t3 t3 1
addi t2 t2 1
addi t1 t1 -1
j run_test_saving_data_loop
run_test_error_reading:
    la a0 error_path
    li a7 4
    ecall

    lw ra (sp)
    addi sp sp 4

    li a0 1

    ret
run_test_end_loop:
    li a7, 57
    mv a0, t6
    ecall

    lw ra (sp)
    addi sp sp 4

    li a0 0

    ret
# Program for solving task.
# input: a0 - path for saving data, a1 - address of data, which
# output: nothing.
_run_test_write:
    mv t2 a1

    addi sp sp -4
    sw ra (sp)

```

```
    li    a7, 1024
    li    a1, 1
    ecall
    mv    t6, a0

    beq a0 s10 run_test_error_writing

    li t0 0
    li t1 512
    la t3 buffer
    mv t5 s8
run_test_loading_data_loop:
    beqz t1 run_test_write_loop
    beqz t5 run_test_write_loop

    lb t4 (t2)
    sb t4 (t3)

    addi t3 t3 1
    addi t2 t2 1
    addi t1 t1 -1
    addi t5 t5 -1
    j run_test_loading_data_loop
run_test_write_loop:
    mv a0 t6
    la a1 buffer
    li a2 512
    li a7 64
    ecall

    blitz a0 run_test_error_writing
    beqz a0 run_test_end_write_loop

    li t1 512
    la t3 buffer

    blez t5 run_test_end_write_loop
```

```
j run_test_loading_data_loop
run_test_error_writing:
    la a0 error_path
    li a7 4
    ecall

    lw ra (sp)
    addi sp sp 4

    li a0 1

    ret
run_test_end_write_loop:
    li a7 57
    mv a0 t6
    ecall

    lw ra (sp)
    addi sp sp 4

    li a0 0

    ret
# Program for clearing data.
# input: a0 - address of data, which stores input from user,
# output: nothing.
_clear_data:
    addi sp sp -4
    sw ra (sp)

    mv t0 s8
    mv t1 a0
    mv t2 a1
clear_loop:
    beqz t0 end_clear_loop

    sb zero (t1)
```

```
    sb zero (t2)

    addi t1 t1 1
    addi t2 t2 1
    addi t0 t0 -1

    j clear_loop
end_clear_loop:

    lw ra (sp)
    addi sp sp 4

    ret
```

macros.inc

```
# Macro for _input
# input - register, which stores data's address.
# output - nothing.
.macro input %reg_data
    mv a0 %reg_data
    jal _input
.end_macro

# Macro for _print
# input - register, which stores result data's address
# output - nothing.
.macro print %reg_data
    mv a0 %reg_data
    jal _print
.end_macro

# Macro for _process_string
# input - two registers: first stores data's address,
# second stores output_data's address.
# output - nothing.
.macro process_string %data_reg %output_data_reg
```

```

        mv a0 %data_reg
        mv a1 %output_data_reg
        jal _process_string
.end_macro

# Macro for _remove_newline_prog
# input - two registers: first stores data's address,
# second stores type of data.
# output = nothing.
.macro remove_newline_prog %data_reg %type
        mv a0 %data_reg
        mv a1 %type
        jal _remove_newline_prog
.end_macro

# Macro for _print_in_console
# input - register, which stores result data's address.
# output - nothing.
.macro print_in_console %data_reg
        mv a0 %data_reg
        jal _print_in_console
.end_macro

# Macro for _run_test_read
# input - register, which stores path for test data and
# second register, which stores data's address.
# output - nothing.
.macro run_test_read %reg_path %data_reg
        mv a0 %reg_path
        mv a1 %data_reg
        jal _run_test_read
.end_macro

# Macro for _run_test_write
# input - register, which stores path for result of test and
# second register, which stores result data's address.
# output - nothing.
.macro run_test_write %reg_path %data_reg

```

```

    mv a0 %reg_path
    mv a1 %data_reg
    jal _run_test_write
.end_macro

# Macro for _clear_data
# input - register, which stores data's address and
# second register, which stores output data's address.
.macro clear_data %reg_data %reg_output_data
    mv a0 %reg_data
    mv a1 %reg_output_data
    jal _clear_data
.end_macro

# Macro, which loads necessary constants.
# input - nothing.
# output - nothing.
.macro load_constants
    li s0 'A'
    li s1 'Z'
    li s2 'a'
    li s3 'z'
    li s4 32
    li s5 0
    li s8 10240
    li s10 -1
.end_macro

# Macro, which clears necessary constants.
# input - nothing.
# output - nothing.
.macro clear_constants
    li s0 0
    li s1 0
    li s2 0
    li s3 0
    li s4 0
    li s5 0

```

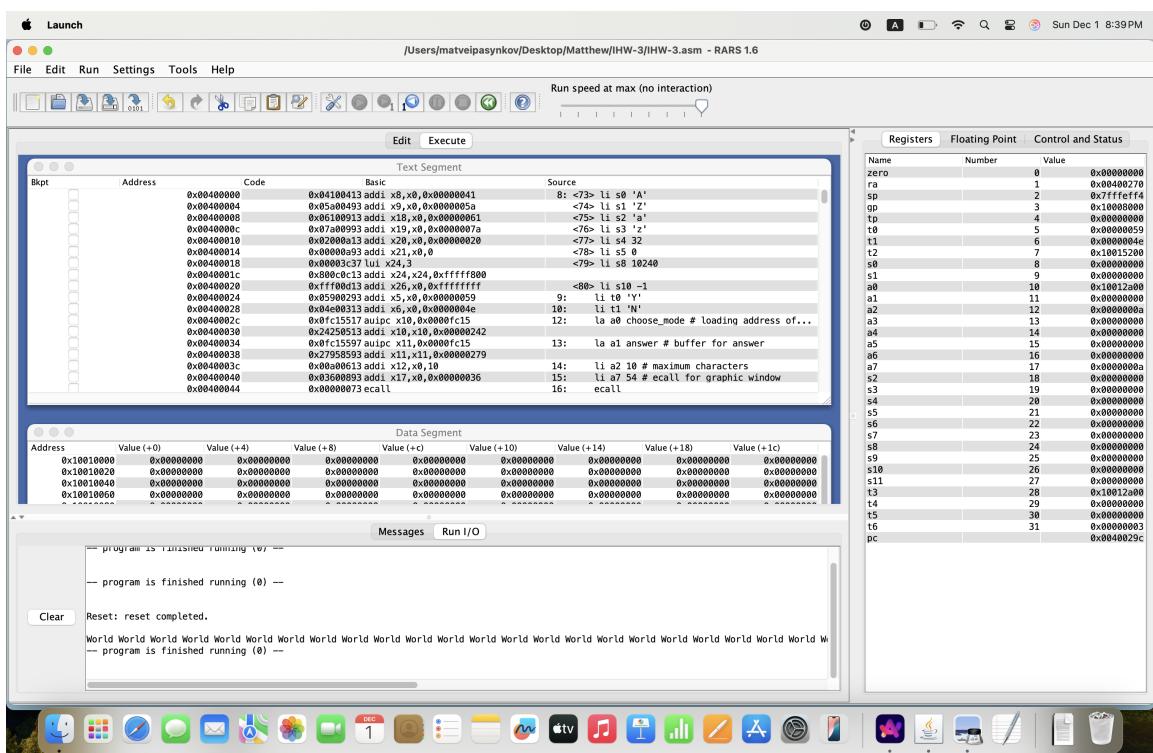
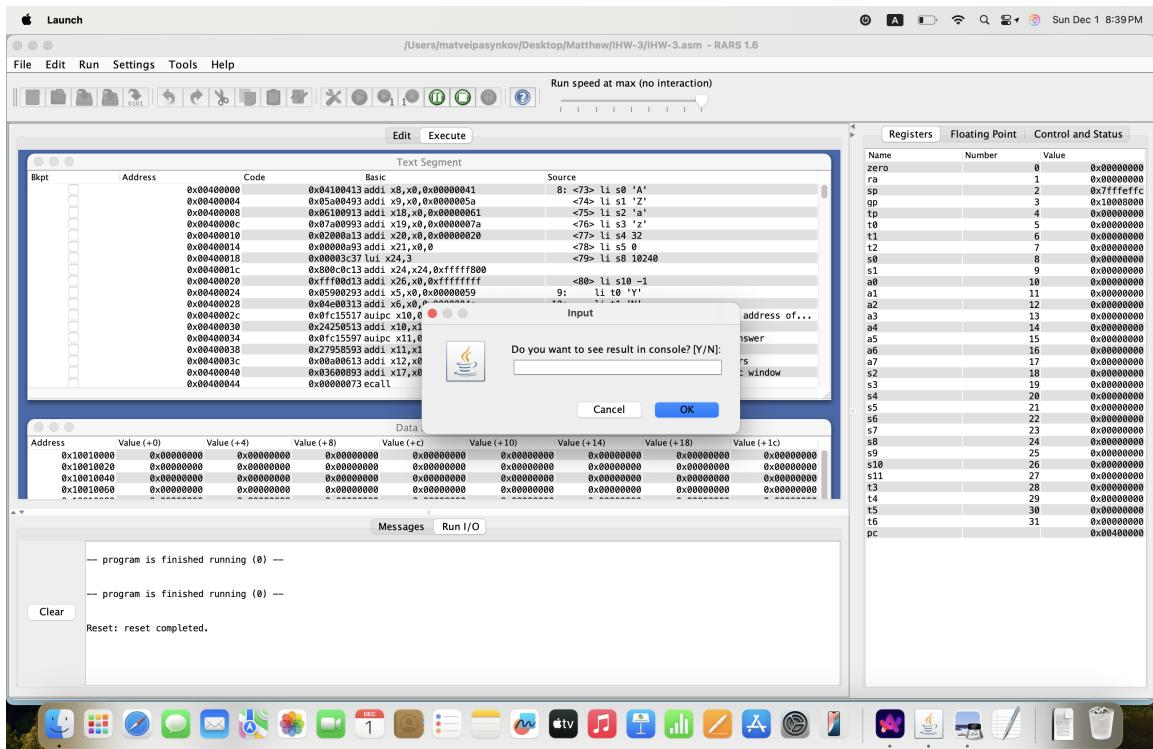
```
    li s8 0
    li s10 0
.end_macro
```

data.asm

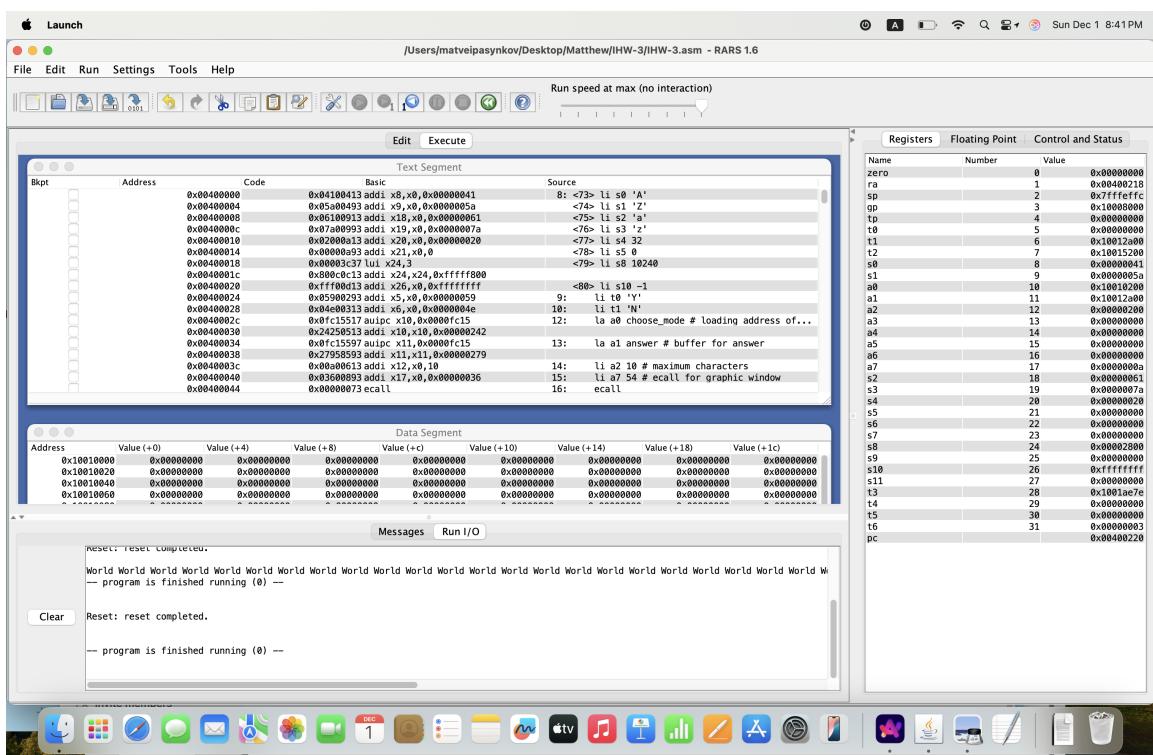
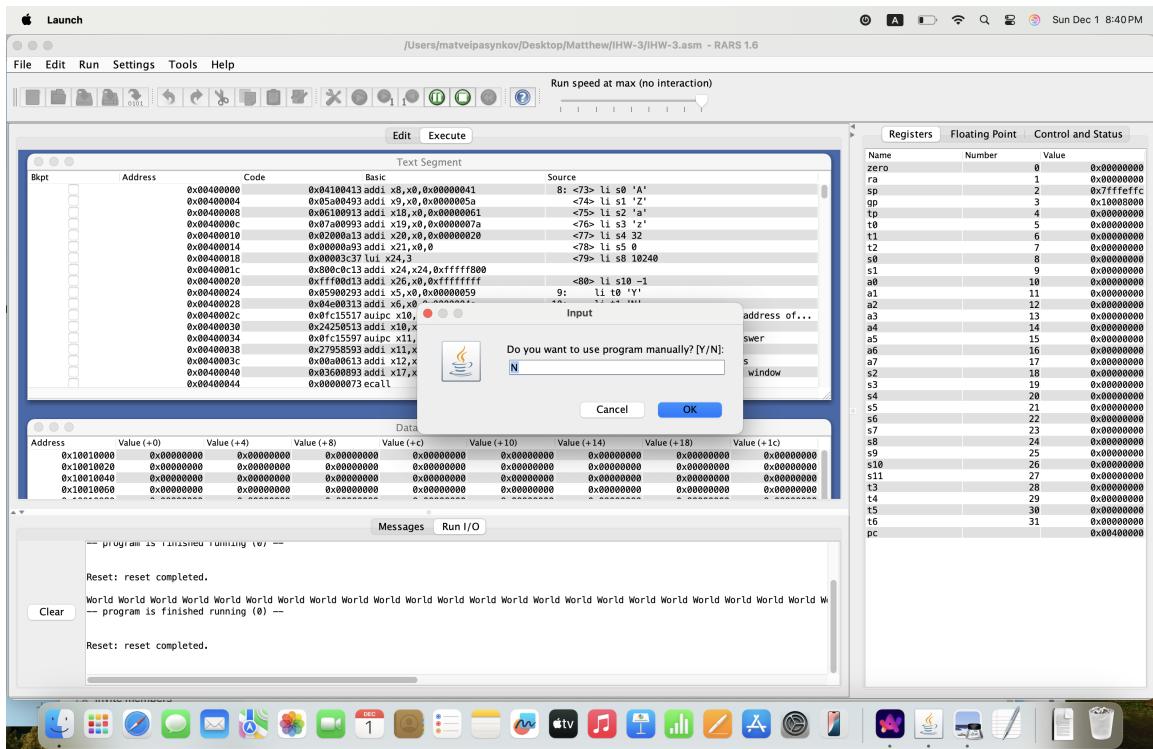
```
# Data Segment for program
.data
buffer: .space 512
data: .space 10240
output_data: .space 10240
space: .byte 32
null: .byte 0
error_path: .asciz "Error. Incorrect path."
write_input_path: "Write input path: "
write_output_path: "Write output path: "
ask_to_display_results: "Do you want to see result in console? [Y/N]: "
choose_mode: "Do you want to use program manually? [Y/N]: "
incorrect_answer: "Incorrect answer."
answer: .space 10
input_path: .space 512
output_path: .space 512
test_path_1: .asciz "files/test1.txt"
output_test_path1: .asciz "files/result1.txt"
test_path_2: .asciz "files/test2.txt"
output_test_path2: .asciz "files/result2.txt"
test_path_3: .asciz "files/test3.txt"
output_test_path3: .asciz "files/result3.txt"
test_path_4: .asciz "files/test4.txt"
output_test_path4: .asciz "files/result4.txt"
```

Дополнение программы критериями на 8 баллов

1. Возможность вывести результат в консоль с ответом Y/N



2. Тестовая программа.



Дополнение программы критериями на 9 баллов

1. Макросы

macros.inc

```

# Macro for _input
# input - register, which stores data's address.
# output - nothing.
.macro input %reg_data
    mv a0 %reg_data
    jal _input
.end_macro

# Macro for _print
# input - register, which stores result data's address
# output - nothing.
.macro print %reg_data
    mv a0 %reg_data
    jal _print
.end_macro

# Macro for _process_string
# input - two registers: first stores data's address,
# second stores output_data's address.
# output - nothing.
.macro process_string %data_reg %output_data_reg
    mv a0 %data_reg
    mv a1 %output_data_reg
    jal _process_string
.end_macro

# Macro for _remove_newline_prog
# input - two registers: first stores data's address,
# second stores type of data.
# output = nothing.
.macro remove_newline_prog %data_reg %type
    mv a0 %data_reg
    mv a1 %type
    jal _remove_newline_prog
.end_macro

# Macro for _print_in_console
# input - register, which stores result data's address.

```

```

# output - nothing.
.macro print_in_console %data_reg
    mv a0 %data_reg
    jal _print_in_console
.end_macro

# Macro for _run_test_read
# input - register, which stores path for test data and
# second register, which stores data's address.
# output - nothing.
.macro run_test_read %reg_path %data_reg
    mv a0 %reg_path
    mv a1 %data_reg
    jal _run_test_read
.end_macro

# Macro for _run_test_write
# input - register, which stores path for result of test and
# second register, which stores result data's address.
# output - nothing.
.macro run_test_write %reg_path %data_reg
    mv a0 %reg_path
    mv a1 %data_reg
    jal _run_test_write
.end_macro

# Macro for _clear_data
# input - register, which stores data's address and
# second register, which stores output data's address.
.macro clear_data %reg_data %reg_output_data
    mv a0 %reg_data
    mv a1 %reg_output_data
    jal _clear_data
.end_macro

# Macro, which loads necessary constants.
# input - nothing.
# output - nothing.

```

```

.macro load_constants
    li s0 'A'
    li s1 'Z'
    li s2 'a'
    li s3 'z'
    li s4 32
    li s5 0
    li s8 10240
    li s10 -1
.end_macro

# Macro, which clears necessary constants.
# input - nothing.
# output - nothing.
.macro clear_constants
    li s0 0
    li s1 0
    li s2 0
    li s3 0
    li s4 0
    li s5 0
    li s8 0
    li s10 0
.end_macro

```

2. Тестовая программа использующая макросы.

IHW-3.asm (run_autotests)

```

# Pasynkov Matvei Evgenievich, Variant 30
.include "macros.inc"
.include "data.asm"

.text
.globl prog_start, manual

prog_start:
    load_constants # loading necessary constants.
    li t0 'Y'

```

```
    li t1 'N'

    la a0 choose_mode # loading address of message with quest.
    la a1 answer # buffer for answer
    li a2 10 # maximum characters
    li a7 54 # ecall for graphic window
    ecall
    li a0 0

    lb a0 answer # loading first character from answer.

    beq a0 t0 manual # if answer is yes => we are going to ma
    # otherwise, we are going to autotests.

auto_tests:
    la a0 test_path_1 # loading first test
    la a1 data
    run_test_read a0 a1 # program for reading test file

    la a0 data
    la a1 output_data
    process_string a0 a1 # program for solving task.

    la a0 output_test_path1
    la a1 output_data
    run_test_write a0 a1 # program for writing test file

    la a0 data
    la a1 output_data
    clear_data a0 a1 # program for cleating buffers

    la a1 data
    la a0 test_path_2 # loading second test
    run_test_read a0 a1

    la a0 data
    la a1 output_data
    process_string a0 a1
```

```
la a0 output_test_path2
la a1 output_data
run_test_write a0 a1

la a0 data
la a1 output_data
clear_data a0 a1

la a0 test_path_3 # loading third test
la a1 data
run_test_read a0 a1

la a0 data
la a1 output_data
process_string a0 a1

la a0 output_test_path3
la a1 output_data
run_test_write a0 a1

la a0 data
la a1 output_data
clear_data a0 a1

la a0 test_path_4 # loading fourth test
la a1 data
run_test_read a0 a1

la a0 data
la a1 output_data
process_string a0 a1

la a0 output_test_path4
la a1 output_data
run_test_write a0 a1

la a0 data
```

```
la a1 output_data
clear_data a0 a1

li a7 10 # stopping program.
ecall

manual:
la a0 data # loading buffer for saving input data
input a0 # reading file

bgtz a0 end_main # if the filename is incorrect, stop the

la a0, data
la a1, output_data
process_string a0 a1 # program for solving task.

la a0 output_data
print a0 # saving result

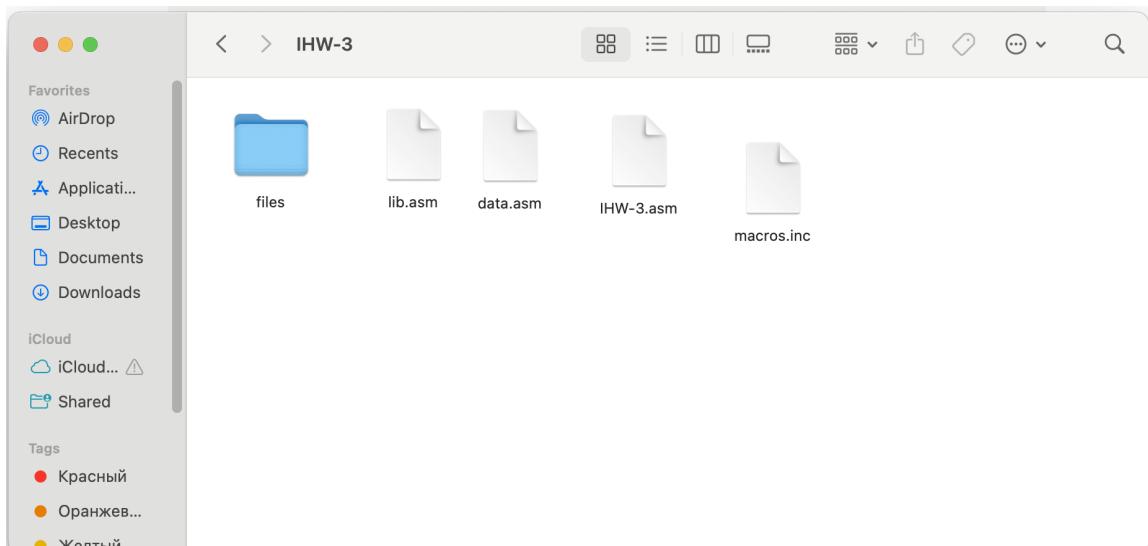
la a0 output_data
print_in_console a0 # program for printing in console.

bgtz a0 end_main

end_main:
# Завершение программы
clear_constants # clearing constants.
li a7, 10          # syscall for stopping program.
ecall
```

Дополнение программы критериями на 10 баллов

1. Программа разбита на несколько единиц компиляции



2. Макросы выделены в отдельную библиотеку (файл macros.inc)

3. Использование графических диалоговых окон.

