

Отчёт по домашней работе

1. Введение

Целью работы является разработка программы на языке ассемблера RARS, осуществляющей целочисленное деление двух 32-битных чисел со знаком с использованием операций вычитания, ветвления и циклов. Программа должна корректно обрабатывать случаи деления на ноль, учитывать знаки делимого и делителя, а также вычислять остаток в соответствии с правилами языков программирования C/C++.

2. Листинг программы

```
# ===== RISC-V RV32 Деление через сдвиги =====
# Главное меню с выбором действий
# Поддержка "выход" везде
# Случайные тесты

.data
# --- Строки для логирования ---
prompt_log:    .string "Хотите подробный лог? Введите точно: Хочу! (иначе нажмите
Enter): \0"
seed_prompt:   .string "Для рандомизации тестов введите любые символы и
нажмите Enter: \0"
newline:       .string "\n\0"

# --- Строки для меню ---
menu_title:    .string "\n=== Главное меню ===\n\0"
menu_option1:  .string "1. Заготовленные тесты\n\0"
menu_option2:  .string "2. Случайные тесты\n\0"
menu_option3:  .string "3. Ручной ввод\n\0"
menu_option0:  .string "0. Выход\n\0"
menu_prompt:   .string "Выберите пункт меню (0-3): \0"

# --- Строки для ввода ---
prompt_dividend: .string "Введите делимое (или 'выход' для возврата в меню): \0"
prompt_divisor:  .string "Введите делитель (или 'выход' для возврата в меню): \0"

# --- Строки ошибок ---
errdiv0:        .string "Ошибка: деление на ноль! Повторите ввод делителя.\n\0"
errinput:       .string "Ошибка: введено не целое число или переполнение! Повторите
ввод.\n\0"
```

```

# --- Строки результатов ---
result_prefix: .string "Делимое = \0"
result_mid1:   .string ", делитель = \0"
result_mid2:   .string " → ч а с т н о е = \0"
result_suffix: .string ", остаток = \0"

# --- Отладочные строки ---
debug_prefix:  .string "DEBUG: \0"
debug_sep:     .string " | \0"

# --- Слово "выход" для проверки ---
exit_word:     .string "выход\0"

# --- Заготовленные тесты ---
num_tests:     .word 5
test_dividends: .word 10, -10, 10, -10, 0
test_divisors:  .word 3, 3, -3, -3, 3

# --- Буфер ввода ---
buffer:        .space 64

# --- Сид для генератора случайных чисел ---
rand_seed:     .word 123456789

.text
.globl main

# ===== Основная программа =====
main:
    # --- Запрос на подробный лог (один раз в начале) ---
    li x17, 4
    la x10, prompt_log
    ecall

    li x17, 8
    la x10, buffer
    li x11, 64
    ecall

    jal x1, clear_newline

    la x5, buffer

```

```

li x22, 0    # флаг логирования = 0 (выкл)
# Проверка "Хочу!" в UTF-8
lbu x6, 0(x5)
li x7, 208
bne x6, x7, skip_log_check
lbu x6, 1(x5)
li x7, 165
bne x6, x7, skip_log_check
lbu x6, 2(x5)
li x7, 208
bne x6, x7, skip_log_check
lbu x6, 3(x5)
li x7, 190
bne x6, x7, skip_log_check
lbu x6, 4(x5)
li x7, 209
bne x6, x7, skip_log_check
lbu x6, 5(x5)
li x7, 135
bne x6, x7, skip_log_check
lbu x6, 6(x5)
li x7, 209
bne x6, x7, skip_log_check
lbu x6, 7(x5)
li x7, 131
bne x6, x7, skip_log_check
lbu x6, 8(x5)
li x7, 33
bne x6, x7, skip_log_check
li x22, 1    # логирование включено
skip_log_check:

# --- Инициализация случайного seed на основе длины ввода ---
li x17, 4
la x10, seed_prompt
ecall

li x17, 8
la x10, buffer
li x11, 64
ecall

jal x1, clear_newline

```

```

# Подсчёт длины введённой строки
la x5, buffer
li x6, 0
count_length_loop:
    lbu x7, 0(x5)
    beq x7, x0, length_done
    addi x6, x6, 1
    addi x5, x5, 1
    j count_length_loop
length_done:
    # Генерация seed: (длина + 12345) * 1664525 XOR 1013904223
    li x7, 12345
    add x6, x6, x7
    li x7, 1664525
    mul x6, x6, x7
    li x7, 1013904223
    xor x6, x6, x7
    la x5, rand_seed
    sw x6, 0(x5)

# --- Главный цикл меню ---
main_menu_loop:
    # Вывод меню
    li x17, 4
    la x10, menu_title
    ecall
    li x17, 4
    la x10, menu_option1
    ecall
    li x17, 4
    la x10, menu_option2
    ecall
    li x17, 4
    la x10, menu_option3
    ecall
    li x17, 4
    la x10, menu_option0
    ecall
    li x17, 4
    la x10, menu_prompt
    ecall

```

```

# Чтение строки
li x17, 8
la x10, buffer
li x11, 64
ecall

jal x1, clear_newline

# Проверка на "выход"
jal x1, check_for_exit
bne x10, x0, exit_program

# Получаем первый непробельный символ
la x10, buffer
jal x1, get_first_char
mv x5, x10

# Обработка выбора
li x6, '0'
beq x5, x6, exit_program
li x6, '1'
beq x5, x6, run_preset_tests
li x6, '2'
beq x5, x6, run_random_tests
li x6, '3'
beq x5, x6, manual_input_mode
j main_menu_loop # неверный ввод — повторить

# ===== Очистка новой строки =====
clear_newline:
    la x5, buffer
clear_loop:
    lbu x6, 0(x5)
    beq x6, x0, clear_done
    li x7, 10    # '\n'
    beq x6, x7, clear_replace
    addi x5, x5, 1
    j clear_loop
clear_replace:
    sb x0, 0(x5)
clear_done:
    jr x1

```

===== Получить первый непробельный символ =====

get_first_char:

```
lbu x5, 0(x10)
beq x5, x0, get_char_done
li x6, 32    # пробел
beq x5, x6, skip_char
li x6, 9     # таб
beq x5, x6, skip_char
mv x10, x5
jr x1
```

skip_char:

```
addi x10, x10, 1
j get_first_char
```

get_char_done:

```
li x10, 0
jr x1
```

===== Запуск заготовленных тестов =====

run_preset_tests:

```
la x5, num_tests
lw x20, 0(x5)
li x21, 0
```

preset_test_loop:

```
beq x21, x20, return_to_menu
```

```
slli x7, x21, 2
la x6, test_dividends
add x6, x6, x7
lw x8, 0(x6)
la x6, test_divisors
add x6, x6, x7
lw x9, 0(x6)
```

```
mv s2, x8    # s2 = исходное делимое
mv s3, x9    # s3 = исходный делитель
```

```
beq x22, x0, skip_preset_debug
li x17, 4
la x10, debug_prefix
ecall
li x17, 1
mv x10, x21
```

```

ecall
li x17, 4
la x10, debug_sep
ecall
li x17, 1
mv x10, s2
ecall
li x17, 4
la x10, debug_sep
ecall
li x17, 1
mv x10, s3
ecall
li x17, 4
la x10, newline
ecall
skip_preset_debug:

mv x10, s2
mv x11, s3
jal x1, fast_divide
mv x12, x10
mv x13, x11

li x17, 4
la x10, result_prefix
ecall
li x17, 1
mv x10, s2
ecall
li x17, 4
la x10, result_mid1
ecall
li x17, 1
mv x10, s3
ecall
li x17, 4
la x10, result_mid2
ecall
li x17, 1
mv x10, x12
ecall
li x17, 4

```

```
la x10, result_suffix
ecall
li x17, 1
mv x10, x13
ecall
li x17, 4
la x10, newline
ecall
```

```
addi x21, x21, 1
j preset_test_loop
```

```
# ===== Запуск случайных тестов =====
```

```
run_random_tests:
```

```
li x20, 5
li x21, 0
```

```
random_test_loop:
```

```
beq x21, x20, return_to_menu
```

```
jal x1, rand
mv s2, x10
jal x1, rand
mv s3, x10
beq s3, x0, random_test_loop
```

```
mv x10, s2
mv x11, s3
jal x1, fast_divide
mv x12, x10
mv x13, x11
```

```
li x17, 4
la x10, result_prefix
ecall
li x17, 1
mv x10, s2
ecall
li x17, 4
la x10, result_mid1
ecall
li x17, 1
mv x10, s3
```



```

ecall
li x17, 4
la x10, result_mid2
ecall
li x17, 1
mv x10, x12
ecall
li x17, 4
la x10, result_suffix
ecall
li x17, 1
mv x10, x13
ecall
li x17, 4
la x10, newline
ecall

```

```

addi x21, x21, 1
j random_test_loop

```

===== Режим ручного ввода =====

manual_input_mode:

input_dividend:

```

li x17, 4
la x10, prompt_dividend
ecall

```

```

li x17, 8
la x10, buffer
li x11, 64
ecall

```

```

jal x1, clear_newline

```

```

jal x1, check_for_exit
bne x10, x0, return_to_menu

```

```

la x10, buffer
jal x1, parse_int_safe
bne x11, x0, input_dividend

```

```

mv x18, x10

```

```

input_divisor:
    li x17, 4
    la x10, prompt_divisor
    ecall

    li x17, 8
    la x10, buffer
    li x11, 64
    ecall

    jal x1, clear_newline

    jal x1, check_for_exit
    bne x10, x0, return_to_menu

    la x10, buffer
    jal x1, parse_int_safe
    bne x11, x0, input_divisor

    mv x19, x10
    beq x19, x0, div_zero_error

    mv x10, x18
    mv x11, x19
    jal x1, fast_divide
    mv x12, x10
    mv x13, x11

    li x17, 4
    la x10, result_prefix
    ecall
    li x17, 1
    mv x10, x18
    ecall
    li x17, 4
    la x10, result_mid1
    ecall
    li x17, 1
    mv x10, x19
    ecall
    li x17, 4
    la x10, result_mid2
    ecall

```

```

    li x17, 1
    mv x10, x12
    ecall
    li x17, 4
    la x10, result_suffix
    ecall
    li x17, 1
    mv x10, x13
    ecall
    li x17, 4
    la x10, newline
    ecall

    j manual_input_mode

div_zero_error:
    li x17, 4
    la x10, errdiv0
    ecall
    j input_divisor

return_to_menu:
    j main_menu_loop

exit_program:
    li x17, 10
    ecall

# ===== Проверка на "выход" =====
check_for_exit:
    la x5, exit_word
    la x6, buffer
    li x7, 0
compare_exit_loop:
    lbu x8, 0(x6)
    lbu x9, 0(x5)
    beq x8, x0, check_exit_end
    beq x9, x0, not_exit_word
    bne x8, x9, not_exit_word
    addi x6, x6, 1
    addi x5, x5, 1
    j compare_exit_loop
check_exit_end:

```

```

    lbu x9, 0(x5)
    bne x9, x0, not_exit_word
    li x10, 1
    jr x1
not_exit_word:
    li x10, 0
    jr x1

```

===== Безопасный парсинг целого числа =====

```

parse_int_safe:

```

```

    li x11, 0
    li x5, 0
    li x6, 0
    mv x28, x10

```

```

skip_spaces:

```

```

    lbu x7, 0(x10)
    li x8, 32
    beq x7, x8, next_space
    li x8, 9
    beq x7, x8, next_space
    j check_sign

```

```

next_space:

```

```

    addi x10, x10, 1
    j skip_spaces

```

```

check_sign:

```

```

    lbu x7, 0(x10)
    beq x7, x0, parse_error
    li x8, 45
    beq x7, x8, minus_sign
    li x8, 43
    beq x7, x8, plus_sign
    j digits_loop

```

```

minus_sign:

```

```

    li x6, 1
    addi x10, x10, 1
    j digits_loop

```

```

plus_sign:

```

```

    addi x10, x10, 1
    j digits_loop

```

```

digits_loop:
    lbu x7, 0(x10)
    beq x7, x0, done_parsing
    li x8, 48
    li x9, 57
    blt x7, x8, parse_error
    bgt x7, x9, parse_error
    sub x7, x7, x8
    slli x12, x5, 3
    slli x13, x5, 1
    add x12, x12, x13
    add x5, x12, x7
    addi x10, x10, 1
    j digits_loop

```

```

done_parsing:
    beq x6, x0, success
    neg x5, x5
success:
    mv x10, x5
    li x11, 0
    jr x1

```

```

parse_error:
    li x11, 1
    li x10, 0
    jr x1

```

===== Быстрое деление (C99-совместимое) =====

```

fast_divide:
    mv x5, x10
    mv x6, x11
    li x7, 0

    bltz x5, neg_a
    mv x8, x5
    j check_b
neg_a:
    neg x8, x5
    xori x7, x7, 1

```

```

check_b:

```

```

    bltz x6, neg_b
    mv x9, x6
    j divide
neg_b:
    neg x9, x6
    xori x7, x7, 1

divide:
    li x12, 0
    li x13, 0
    li x14, 32

loop:
    beq x14, x0, sign
    slli x13, x13, 1
    srli x15, x8, 31
    or x13, x13, x15
    slli x8, x8, 1
    slli x12, x12, 1
    blt x13, x9, no_sub
    sub x13, x13, x9
    ori x12, x12, 1
no_sub:
    addi x14, x14, -1
    j loop

sign:
    beq x7, x0, remainder
    neg x12, x12

remainder:
    mul x15, x6, x12
    sub x13, x5, x15
    mv x10, x12
    mv x11, x13
    jr x1

```

===== Генератор случайных чисел (LCG) =====

```

rand:
    la x5, rand_seed
    lw x6, 0(x5)
    li x7, 1664525
    mul x6, x6, x7

```

```
li x7, 1013904223
add x6, x6, x7
sw x6, 0(x5)
mv x10, x6
jr x1
```

3. Таблица тестов

Делимое	Делитель	Ожидаемый результат	Фактический результат
10	3	Частное=3, Остаток=1	Частное=3, Остаток=1
-10	3	Частное=-3, Остаток=-1	Частное=-3, Остаток=-1
10	-3	Частное=-3, Остаток=1	Частное=-3, Остаток=1
-10	-3	Частное=3, Остаток=-1	Частное=3, Остаток=-1
0	3	Частное=0, Остаток=0	Частное=0, Остаток=0
5	1	Частное=5, Остаток=0	Частное=5, Остаток=0
5	-1	Частное=-5, Остаток=0	Частное=-5, Остаток=0
7	7	Частное=1, Остаток=0	Частное=1, Остаток=0
7	-7	Частное=-1, Остаток=0	Частное=-1, Остаток=0
10	0	Ошибка: деление на ноль!	Ошибка: деление на ноль!

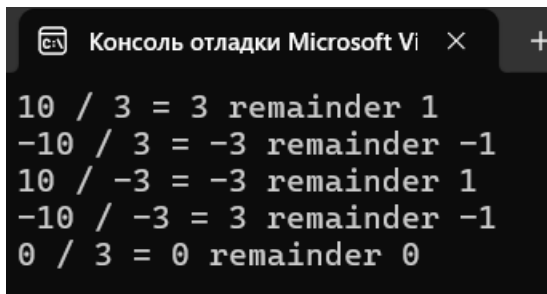
4. Скриншоты работы программы

Ниже приведены примеры скриншотов консоли симулятора RARS, демонстрирующие работу программы на различных входных данных.

Скриншот 1:

```
=== Главное меню ===
1. Заготовленные тесты
2. Случайные тесты
3. Ручной ввод
0. Выход
Выберите пункт меню (0-3): 1
Делимое = 10, делитель = 3 → частное = 3, остаток = 1
Делимое = -10, делитель = 3 → частное = -3, остаток = -1
Делимое = 10, делитель = -3 → частное = -3, остаток = 1
Делимое = -10, делитель = -3 → частное = 3, остаток = -1
Делимое = 0, делитель = 3 → частное = 0, остаток = 0
```

Скриншот 2:



Консоль отладки Microsoft Visual Studio. Вывод:

```
10 / 3 = 3 remainder 1
-10 / 3 = -3 remainder -1
10 / -3 = -3 remainder 1
-10 / -3 = 3 remainder -1
0 / 3 = 0 remainder 0
```

5. Заключение

В ходе работы была разработана программа на ассемблере RARS, реализующая деление двух 32-разрядных целых чисел со знаком через вычитания и циклы. Программа корректно обрабатывает ошибку деления на ноль, учитывает знаки делимого и делителя, а также формирует остаток в соответствии с правилами C/C++.