Homework #2: Rational Number Arithmetic Using C++ and MIPS Assembly Language.

Zi Xuan Li

The Grove School of Engineering, The City College of New York

CSC 34200 E[19155]: Computer Organization

Professor Gertner

February 25th, 2024

Table of Contents

Objective:

This assignment aims to refresh our understandings of rational number arithmetic by implementing a rational number class in C++ and MIPS assembly, providing methods for addition, subtraction, multiplication, division, and a method for checking if a number is rational.

C++ Rational Number Arithmetic

The C++ program defines a class RATIONAL to perform arithmetic operations on rational numbers. In the main function, two instances of RATIONAL classes are created. R1 with the value ½ and R2 with the value ¾.

1. Addition result: The program first computes the addition of r1 and r2, which results in ½ + ¾. The result of which is 10/8, which is then displayed.
2. Subtraction result: Next, the program computes the subtraction of r1 and r2, which is ½ - ¾. The result of this subtraction is -2/8, which is then displayed.
3. Multiplication result: The program then calculates the multiplication of r1 and r2, which is ½ * ¾. The result of this multiplication is 3/8, which is also displayed.
4. Division result: Finally, the program performs the division of r1 and r2, which is ½ / ¾. The result of which is 4/6, which is then displayed.
5. Is rational?: Lastly, the program displays whether the result of each arithmetic operation done on r1 and r2 is rational or not.

```cpp
#include <iostream>

using namespace std;

class RATIONAL {
private:
    int numerator;
    int denominator;

public:
    RATIONAL(int num = 0, int denom = 1) : numerator(num), denominator(denom) {
        if (denominator == 0) {
            cout << "Error: Denominator cannot be zero." << endl;
            exit(1);
        }
    }

    RATIONAL Add_Rational(const RATIONAL& other) const {
        int num = numerator * other.denominator + other.numerator * denominator;
        int denom = denominator * other.denominator;
        return RATIONAL(num, denom);
    }

    RATIONAL Sub_Rational(const RATIONAL& other) const {
        int num = numerator * other.denominator - other.numerator * denominator;
        int denom = denominator * other.denominator;
        return RATIONAL(num, denom);
    }

    RATIONAL Mul_Rational(const RATIONAL& other) const {
        int num = numerator * other.numerator;
        int denom = denominator * other.denominator;
        return RATIONAL(num, denom);
    }

    RATIONAL Div_Rational(const RATIONAL& other) const {
        if (other.numerator == 0) {
            cout << "Error: Division by zero." << endl;
            exit(1);
        }
        int num = numerator * other.denominator;
        int denom = denominator * other.numerator;
        return RATIONAL(num, denom);
    }
```

```
    // Method to check if the number is rational
    bool Is_Rational() const {
        return denominator != 0;
    }

    // Method to display the rational number
    void Display() const {
        cout << numerator << "/" << denominator << endl;
    }

    friend ostream& operator<<(ostream& os, const RATIONAL& rational) {
        os << "(" << rational.numerator << "/" << rational.denominator << ")";
        return os;
    }
};

int main() {
    RATIONAL r1(1, 2);
    RATIONAL r2(3, 4);

    cout << "Original Rationals: r1 = " << r1 << ", r2 = " << r2 << endl << endl;
    cout << "Addition: " << r1 << " + " << r2 << " = " << r1.Add_Rational(r2) <<
endl;
    cout << "Subtraction: " << r1 << " - " << r2 << " = " << r1.Sub_Rational(r2)
<< endl;
    cout << "Multiplication: " << r1 << " * " << r2 << " = " << r1.Mul_Ra-
tional(r2) << endl;
    cout << "Division: " << r1 << " / " << r2 << " = " << r1.Div_Rational(r2) <<
endl;

    RATIONAL result_add = r1.Add_Rational(r2);
    cout << "\nAddition Result: ";
    result_add.Display();
    cout << "Is the additonal result rational? " << (result_add.Is_Rational() ?
"Yes" : "No") << endl;

    RATIONAL result_sub = r1.Sub_Rational(r2);
    cout << "Subtraction Result: ";
    result_sub.Display();
    cout << "Is the subtraction result rational? " << (result_sub.Is_Rational() ?
"Yes" : "No") << endl;

    RATIONAL result_mul = r1.Mul_Rational(r2);
    cout << "Multiplication Result: ";
    result_mul.Display();
```

```
    cout << "Is the multiplication result rational? " << (result_mul.Is_Ra-
tional() ? "Yes" : "No") << endl;

    RATIONAL result_div = r1.Div_Rational(r2);
    cout << "Division Result: ";
    result_div.Display();
    cout << "Is the division result rational? " << (result_div.Is_Rational() ?
"Yes" : "No") << endl;

    return 0;
}
```

Figure 1: C++ code with a class rational and methods add_rational, sub_rational, mul_rational, div_rational, is_rational, and main function to the test the class.

```
Original Rationals: r1 = (1/2), r2 = (3/4)

Addition: (1/2) + (3/4) = (10/8)
Subtraction: (1/2) - (3/4) = (-2/8)
Multiplication: (1/2) * (3/4) = (3/8)
Division: (1/2) / (3/4) = (4/6)

Addition Result: 10/8
Is the additonal result rational? Yes
Subtraction Result: -2/8
Is the subtraction result rational? Yes
Multiplication Result: 3/8
Is the multiplication result rational? Yes
Division Result: 4/6
Is the division result rational? Yes
```

Figure 2: This is the output of code from Figure 1.

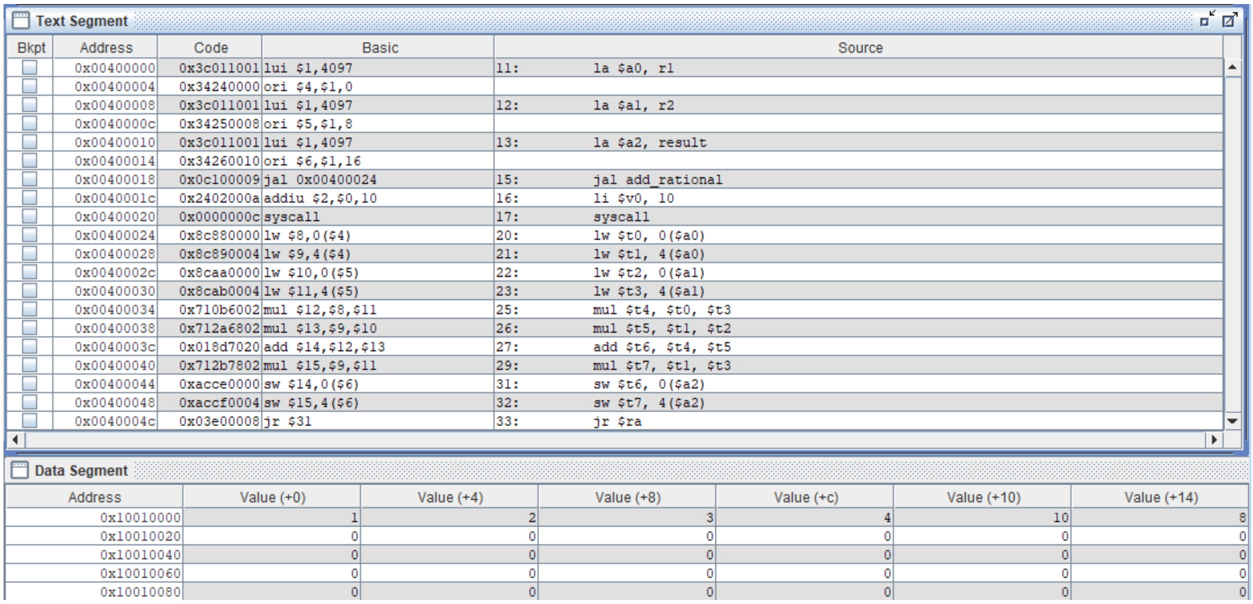
MIPS

Addition for Rational Numbers

The MIPS assembly code first loads the numerator and denominator of both **r1** and **r2** into registers. It then performs cross-multiplication to calculate the new numerator and denominator of the result: the numerator of **r1** multiplied by the denominator of **r2** is stored in **$t4**, and the denominator of **r1** multiplied by the numerator of **r2** is stored in **$t5**. The sum of **$t4** and **$t5** gives the new numerator of the result, stored in **$t6**. The new denominator of the result is the product of the denominators of **r1** and **r2**, stored in **$t7**.

```
.data

r1: .word 1, 2
r2: .word 3, 4
result: .word 0, 0

.text
.globl main

main:
    la $a0, r1
    la $a1, r2
    la $a2, result

    jal add_rational
    li $v0, 10
    syscall

add_rational:
    lw $t0, 0($a0)
    lw $t1, 4($a0)
    lw $t2, 0($a1)
    lw $t3, 4($a1)

    mul $t4, $t0, $t3
    mul $t5, $t1, $t2
    add $t6, $t4, $t5

    mul $t7, $t1, $t3

    sw $t6, 0($a2)
    sw $t7, 4($a2)
    jr $ra
```

Figure 3: MIPS assembly code to compute the addition of two rational numbers, r1 = ½ and r2 = ¾.



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	11: la $a0, r1
	0x00400004	0x34240000	ori $4,$1,0	
	0x00400008	0x3c011001	lui $1,4097	12: la $a1, r2
	0x0040000c	0x34250008	ori $5,$1,8	
	0x00400010	0x3c011001	lui $1,4097	13: la $a2, result
	0x00400014	0x34260010	ori $6,$1,16	
	0x00400018	0x0c100009	jal 0x00400024	15: jal add_rational
	0x0040001c	0x2402000a	addiu $2,$0,10	16: li $v0, 10
	0x00400020	0x0000000c	syscall	17: syscall
	0x00400024	0x8c880000	lw $8,0($4)	20: lw $t0, 0($a0)
	0x00400028	0x8c890004	lw $9,4($4)	21: lw $t1, 4($a0)
	0x0040002c	0x8caa0000	lw $10,0($5)	22: lw $t2, 0($a1)
	0x00400030	0x8cab0004	lw $11,4($5)	23: lw $t3, 4($a1)
	0x00400034	0x710b6002	mul $12,$8,$11	25: mul $t4, $t0, $t3
	0x00400038	0x712a6802	mul $13,$9,$10	26: mul $t5, $t1, $t2
	0x0040003c	0x018d7020	add $14,$12,$13	27: add $t6, $t4, $t5
	0x00400040	0x712b7802	mul $15,$9,$11	29: mul $t7, $t1, $t3
	0x00400044	0xacce0000	sw $14,0($6)	31: sw $t6, 0($a2)
	0x00400048	0xaccf0004	sw $15,4($6)	32: sw $t7, 4($a2)
	0x0040004c	0x03e00008	jr $31	33: jr $ra

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	1	2	3	4	10	8
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0

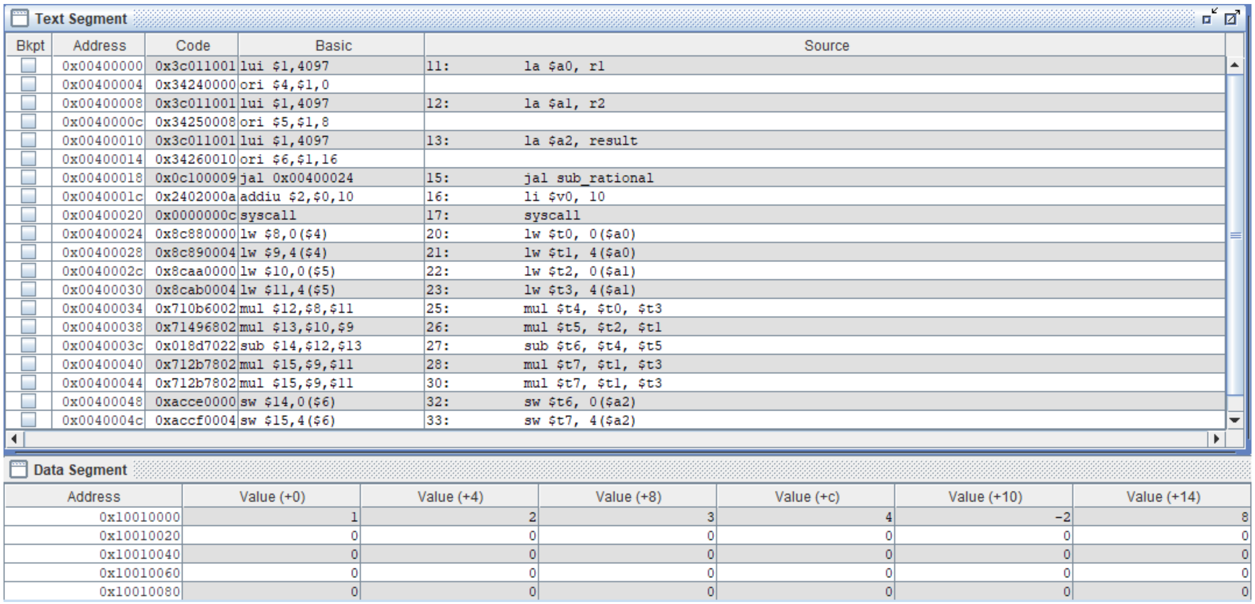
Figure 4: Output of the code from Figure 3.

Subtraction for Rational Numbers

The code begins by loading the numerator and denominator of **r1** and **r2** into registers **$t0**, **$t1**, **$t2**, and **$t3** respectively. It then performs cross-multiplication to calculate the new numerator and denominator of the result: the product of the numerator of **r1** and the denominator of **r2** is stored in **$t4**, and the product of the numerator of **r2** and the denominator of **r1** is stored in **$t5**. The subtraction of **$t5** from **$t4** gives the new numerator of the result, stored in **$t6**. The new denominator of the result is the product of the denominators of **r1** and **r2**, which is stored in **$t7**.

```
.data

r1: .word 1, 2
r2: .word 3, 4
result: .word 0, 0

.text
.globl main

main:
    la $a0, r1
    la $a1, r2
    la $a2, result

    jal sub_rational
    li $v0, 10
    syscall

sub_rational:
    lw $t0, 0($a0)
    lw $t1, 4($a0)
    lw $t2, 0($a1)
    lw $t3, 4($a1)

    mul $t4, $t0, $t3
    mul $t5, $t2, $t1
    sub $t6, $t4, $t5
    mul $t7, $t1, $t3

    mul $t7, $t1, $t3

    sw $t6, 0($a2)
    sw $t7, 4($a2)
    jr $ra
```

Figure 5: MIPS assembly code to compute the subtraction of two rational numbers, r1 = ½ and r2 = ¾.



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	11: la $a0, r1
	0x00400004	0x34240000	ori $4,$1,0	
	0x00400008	0x3c011001	lui $1,4097	12: la $a1, r2
	0x0040000c	0x34250008	ori $5,$1,8	
	0x00400010	0x3c011001	lui $1,4097	13: la $a2, result
	0x00400014	0x34260010	ori $6,$1,16	
	0x00400018	0x0c100009	jal 0x00400024	15: jal sub_rational
	0x0040001c	0x2402000a	addiu $2,$0,10	16: li $v0, 10
	0x00400020	0x0000000c	syscall	17: syscall
	0x00400024	0x8c880000	lw $8,0($4)	20: lw $t0, 0($a0)
	0x00400028	0x8c890004	lw $9,4($4)	21: lw $t1, 4($a0)
	0x0040002c	0x8caa0000	lw $10,0($5)	22: lw $t2, 0($a1)
	0x00400030	0x8cab0004	lw $11,4($5)	23: lw $t3, 4($a1)
	0x00400034	0x710b6002	mul $12,$8,$11	25: mul $t4, $t0, $t3
	0x00400038	0x71496802	mul $13,$10,$9	26: mul $t5, $t2, $t1
	0x0040003c	0x018d7022	sub $14,$12,$13	27: sub $t6, $t4, $t5
	0x00400040	0x712b7802	mul $15,$9,$11	28: mul $t7, $t1, $t3
	0x00400044	0x712b7802	mul $15,$9,$11	30: mul $t7, $t1, $t3
	0x00400048	0xacce0000	sw $14,0($6)	32: sw $t6, 0($a2)
	0x0040004c	0xaccf0004	sw $15,4($6)	33: sw $t7, 4($a2)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	1	2	3	4	-2	8
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0

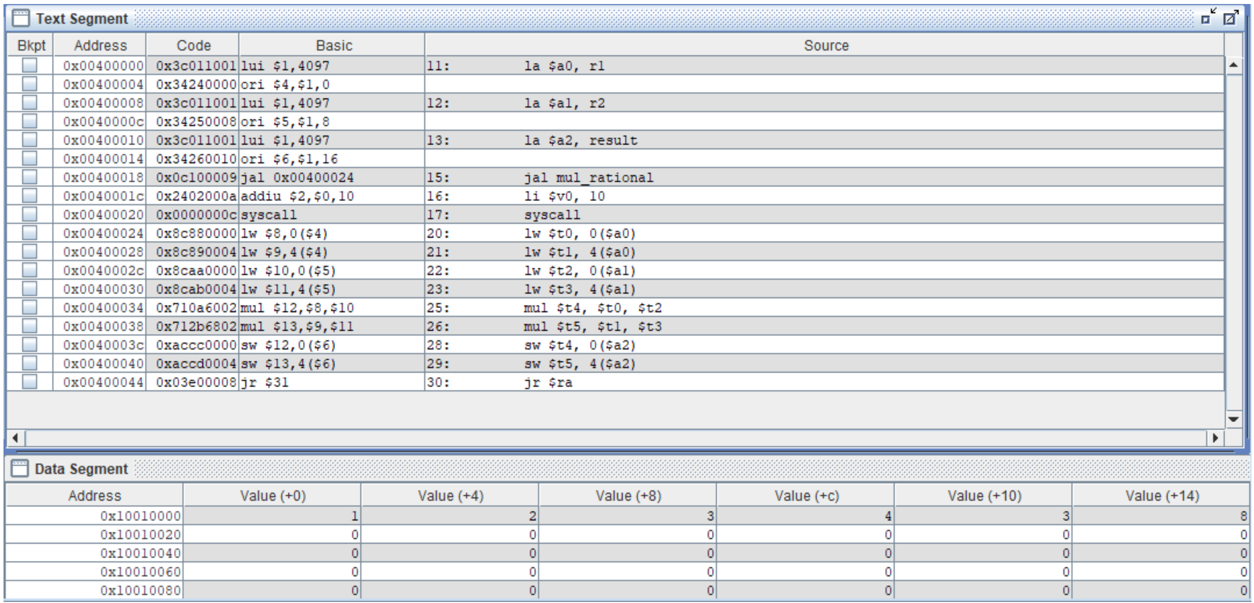
Figure 6: Output of the code from Figure 5.

Multiplication for Rational Numbers

The code begins by loading the numerator and denominator of **r1** and **r2** into registers **$t0**, **$t1**, **$t2**, and **$t3** respectively. It then performs multiplication to calculate the new numerator and denominator of the result: the product of the numerator of **r1** and the numerator of **r2** is stored in **$t4**, and the product of the denominator of **r1** and the denominator of **r2** is stored in **$t5**.

```
.data

r1: .word 1, 2
r2: .word 3, 4
result: .word 0, 0

.text
.globl main

main:
    la $a0, r1
    la $a1, r2
    la $a2, result

    jal mul_rational
    li $v0, 10
    syscall

mul_rational:
    lw $t0, 0($a0)
    lw $t1, 4($a0)
    lw $t2, 0($a1)
    lw $t3, 4($a1)

    mul $t4, $t0, $t2
    mul $t5, $t1, $t3

    sw $t4, 0($a2)
    sw $t5, 4($a2)
    jr $ra
```

Figure 7: MIPS assembly code to compute the multiplication of two rational numbers, r1 = ½ and r2 = ¾.



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	11: la $a0, r1
	0x00400004	0x34240000	ori $4,$1,0	
	0x00400008	0x3c011001	lui $1,4097	12: la $a1, r2
	0x0040000c	0x34250008	ori $5,$1,8	
	0x00400010	0x3c011001	lui $1,4097	13: la $a2, result
	0x00400014	0x34260010	ori $6,$1,16	
	0x00400018	0x0c100009	jal 0x00400024	15: jal mul_rational
	0x0040001c	0x2402000a	addiu $2,$0,10	16: li $v0, 10
	0x00400020	0x0000000c	syscall	17: syscall
	0x00400024	0x8c880000	lw $8,0($4)	20: lw $t0, 0($a0)
	0x00400028	0x8c890004	lw $9,4($4)	21: lw $t1, 4($a0)
	0x0040002c	0x8caa0000	lw $10,0($5)	22: lw $t2, 0($a1)
	0x00400030	0x8cab0004	lw $11,4($5)	23: lw $t3, 4($a1)
	0x00400034	0x710a6002	mul $12,$8,$10	25: mul $t4, $t0, $t2
	0x00400038	0x712b6802	mul $13,$9,$11	26: mul $t5, $t1, $t3
	0x0040003c	0xaccc0000	sw $12,0($6)	28: sw $t4, 0($a2)
	0x00400040	0xaccd0004	sw $13,4($6)	29: sw $t5, 4($a2)
	0x00400044	0x03e00008	jr $31	30: jr $ra

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	1	2	3	4	3	8
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0

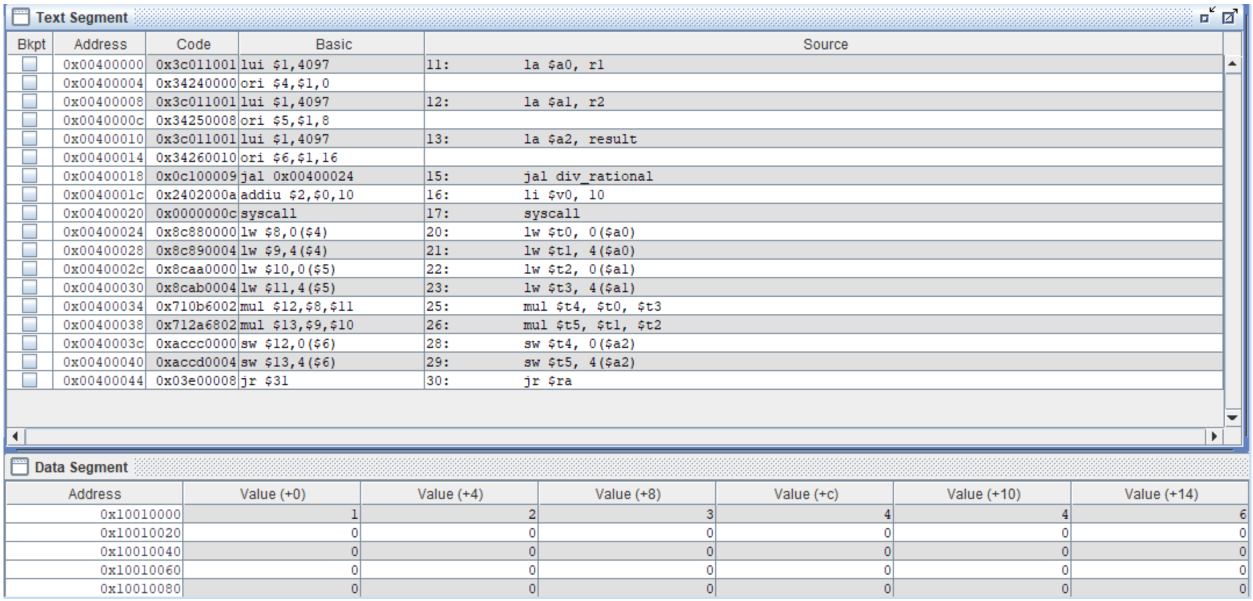
Figure 8: Output of the code from Figure 7.

Division for Rational Numbers

The code begins by loading the numerator and denominator of **r1** and **r2** into registers **$t0**, **$t1**, **$t2**, and **$t3** respectively. It then performs multiplication to calculate the new numerator and denominator of the result: the product of the numerator of **r1** and the denominator of **r2** is stored in **$t4**, and the product of the denominator of **r1** and the numerator of **r2** is stored in **$t5**.

```
.data

r1: .word 1, 2
r2: .word 3, 4
result: .word 0, 0

.text
.globl main

main:
    la $a0, r1
    la $a1, r2
    la $a2, result

    jal div_rational
    li $v0, 10
    syscall

div_rational:
    lw $t0, 0($a0)
    lw $t1, 4($a0)
    lw $t2, 0($a1)
    lw $t3, 4($a1)

    mul $t4, $t0, $t3
    mul $t5, $t1, $t2

    sw $t4, 0($a2)
    sw $t5, 4($a2)
    jr $ra
```

Figure 9: MIPS assembly code to compute the division of two rational numbers, r1 = ½ and r2 = ¾.



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	11: la $a0, r1
	0x00400004	0x34240000	ori $4,$1,0	
	0x00400008	0x3c011001	lui $1,4097	12: la $a1, r2
	0x0040000c	0x34250008	ori $5,$1,8	
	0x00400010	0x3c011001	lui $1,4097	13: la $a2, result
	0x00400014	0x34260010	ori $6,$1,16	
	0x00400018	0x0c100009	jal 0x00400024	15: jal div_rational
	0x0040001c	0x2402000a	addiu $2,$0,10	16: li $v0, 10
	0x00400020	0x0000000c	syscall	17: syscall
	0x00400024	0x8c880000	lw $8,0($4)	20: lw $t0, 0($a0)
	0x00400028	0x8c890004	lw $9,4($4)	21: lw $t1, 4($a0)
	0x0040002c	0x8caa0000	lw $10,0($5)	22: lw $t2, 0($a1)
	0x00400030	0x8cab0004	lw $11,4($5)	23: lw $t3, 4($a1)
	0x00400034	0x710b6002	mul $12,$8,$11	25: mul $t4, $t0, $t3
	0x00400038	0x712a6802	mul $13,$9,$10	26: mul $t5, $t1, $t2
	0x0040003c	0xaccc0000	sw $12,0($6)	28: sw $t4, 0($a2)
	0x00400040	0xaccd0004	sw $13,4($6)	29: sw $t5, 4($a2)
	0x00400044	0x03e00008	jr $31	30: jr $ra

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	1	2	3	4	4	6
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0

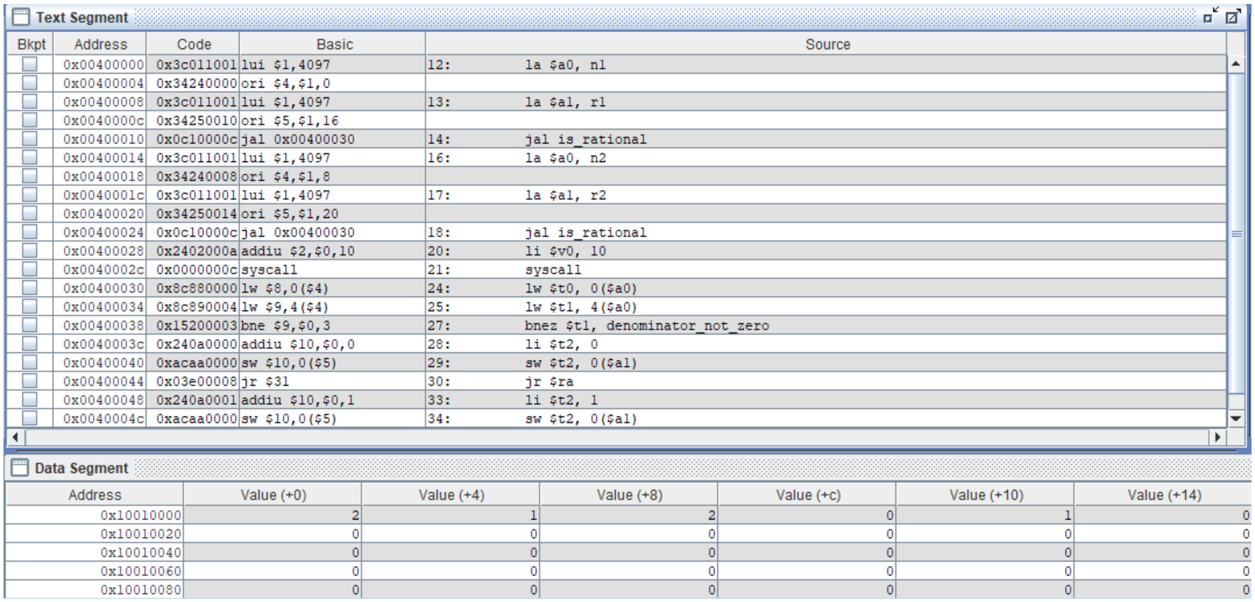
Figure 10: Output of the code from Figure 9.

Is this number rational?

The code defines two sets of data: **n1** with values **2** and **1**, and **n2** with values **2** and **0**. The program then checks if the denominator of a given rational number is zero or not.

```
.data
n1: .word 2, 1
n2: .word 2, 0

r1: .word 0
r2: .word 0

.text
.globl main

main:
    la $a0, n1
    la $a1, r1
    jal is_rational

    la $a0, n2
    la $a1, r2
    jal is_rational

    li $v0, 10
    syscall

is_rational:
    lw $t0, 0($a0)
    lw $t1, 4($a0)

    bnez $t1, denominator_not_zero
    li $t2, 0
    sw $t2, 0($a1)
    jr $ra

denominator_not_zero:
    li $t2, 1
    sw $t2, 0($a1)
    jr $ra
```

Figure 11: MIPS assembly code to check whether two numbers are rational or not, r1 = 2/1 and r2 = 2/0.



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	12: la $a0, n1
	0x00400004	0x34240000	ori $4,$1,0	
	0x00400008	0x3c011001	lui $1,4097	13: la $a1, r1
	0x0040000c	0x34250010	ori $5,$1,16	
	0x00400010	0x0c10000c	jal 0x00400030	14: jal is_rational
	0x00400014	0x3c011001	lui $1,4097	16: la $a0, n2
	0x00400018	0x34240008	ori $4,$1,8	
	0x0040001c	0x3c011001	lui $1,4097	17: la $a1, r2
	0x00400020	0x34250014	ori $5,$1,20	
	0x00400024	0x0c10000c	jal 0x00400030	18: jal is_rational
	0x00400028	0x2402000a	addiu $2,$0,10	20: li $v0, 10
	0x0040002c	0x0000000c	syscall	21: syscall
	0x00400030	0x8c880000	lw $8,0($4)	24: lw $t0, 0($a0)
	0x00400034	0x8c890004	lw $9,4($4)	25: lw $t1, 4($a0)
	0x00400038	0x15200003	bne $9,$0,3	27: bnez $t1, denominator_not_zero
	0x0040003c	0x240a0000	addiu $10,$0,0	28: li $t2, 0
	0x00400040	0xacaa0000	sw $10,0($5)	29: sw $t2, 0($a1)
	0x00400044	0x03e00008	jr $31	30: jr $ra
	0x00400048	0x240a0001	addiu $10,$0,1	33: li $t2, 1
	0x0040004c	0xacaa0000	sw $10,0($5)	34: sw $t2, 0($a1)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	2	1	2	0	1	0
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0

Figure 12: Output of the code from Figure 11.

Conclusion:

In this assignment, we gained a comprehensive understanding of rational number arithmetic and its implementations in programming. We jogged our memories on how to create a class in C++ for the purpose of doing operations such as addition and subtraction on rational numbers. Additionally, we also translated these operations into MIPS instructions which deepened our insight into assembly programing.