Table of Contents

Exercise 1: Compute_Integral Function

Problem Description:

Develop a function to calculate a definite integral for the quadratic function f(x) = px^2 + qx + r between the limits a and b. using RISC-V assembly language and the Creator simulator.

Pseudocode:

```
FUNCTION Compute_Integral(a: Integer, b: Integer, p: Integer, q: Integer, r: Integer,
                N: Integer) -> Float
   // Compute the width of each step
   h = (b - a) / N

   // Initialize the sum to 0.0
   sum = 0.0

   // Loop from n = 0 to n < N
   FOR n FROM 0 TO N - 1
      // Compute x = a + n * h
      x = a + n * h

      // Compute f(x) = p * x^2 + q * x + r
      fx = p * x * x + q * x + r

      // Add f(x) to the sum
      sum = sum + fx
   END FOR

   // Compute the integral as h * sum
   integral = h * sum

   RETURN integral
END FUNCTION
```

Design Decisions and Results

Step Size Calculation (h = (b - a) / N): Divides the range [a, b] into N equal segments for numerical integration.

Loop Structure: Uses a for loop (n = 0 to N - 1) to accumulate function values, making the implementation simple and straightforward.

Function Calculation (f(x) = p * x^2 + q * x + r): Integrates a quadratic polynomial, which balances simplicity and complexity for practical demonstration.

Area Approximation: Computes the integral as h * sum (similar to the Riemann sum). This gives an approximate value of the area under the curve.

Conclusions

Working with assembly taught us how important registers and the stack are for function calls and data Storage concepts that felt abstract in higher-level programming.
we realized that patience and attention to detail are essential.
And that we have to dive deep into debugging without the help of modern tools.

Problems Encountered

Every command felt like a puzzle. I kept checking notes and RISKV official documentation to avoid mistakes.

We had a hard time referencing memory correctly and sometimes overwrote values by mistake, especially with indirect addressing.

Managing registers was challenging too. Unlike higher-level languages like python.
We had to decide manually which register to use for each value.
We often destroyed them up, leading to unexpected results and trial and error,
but as we can see we spent 6 hours which was rewarding to us.

Estimation of Time Spent

6 Hours without managing test cases.

Test Plan

Test 1: Basic Quadratic
- Description: Validates the basic functionality with a simple quadratic function where the exact integral is known.
- Input: a = 0, b = 2, p = 1, q = 0, r = 0, N = 1000
- Expected: Integral ≈ 2.6667
- What Happened: Integral was calculated correctly.

Test 2: Linear Function
- Description: Tests a linear function where the exact integral is straightforward to compute.
- Input: a = 1, b = 3, p = 0, q = 2, r = 1, N = 1000
- Expected: Integral = 10, Integer Result = 10
- What Happened: Integral was computed accurately, and the integer result was as expected.

Test 3: Constant Function
- Description: Ensures the function correctly handles constant functions.
- Input: a = -5, b = 5, p = 0, q = 0, r = 3, N = 1000
- Expected: Integral = 30, Integer Result = 30
- What Happened: The function correctly returned the expected integral and integer result.

Test 4: Negative
- Description: Tests the function's ability to handle negative coefficients and evaluate integrals that result in negative values.
- Input: a = 2, b = 4, p = -1, q = 4, r = -3, N = 1000
- Expected: Integral ≈ -0.6667
- What Happened: The function handled negative coefficients correctly, returning the expected integral and integer result.

Test 5: Zero Interval
- Description: Validates that the function correctly handles cases where the lower and upper limits are the same.
- Input: a = 5, b = 5, p = 2, q = 3, r = 4, N = 1000
- Expected: Integral = 0
- What Happened: The function correctly returned an integral of zero and an integer result of zero.

Test 6: Large Number of Steps for High Accuracy
- Description: Tests the function's accuracy and performance with a very large number of steps.
- Input: a = 0, b = 1, p = 3, q = 0, r = 0, N = 1000000
- Expected: Integral = 1, Integer Result = 1
- What Happened: The function kept running for hours but we saw that it was giving 0 so we stopped it

Test 7: Small Number of Steps for Lower Accuracy
- Description: Evaluates the impact of a low number of steps on the accuracy of the integral.
- Input: a = 0, b = 1, p = 3, q = 0, r = 0, N = 10
- Expected: Integral Approximation ≈ 0.93
- What Happened: The integral approximation was close to the expected value it was around 0.81.

Test 8: Integration Over Negative Range
- Description: Ensures the function correctly handles integration over negative ranges.
- Input: a = -3, b = -1, p = 2, q = -1, r = 5, N = 1000
- Expected: Integral ≈ 31.3333
- What Happened: The function successfully computed the integral over a negative range and returned the correct integer result.

Exercise 2: Integral_Matrix Function

Problem Description:

Develop a function called `Integral_Matrix` that stores values into an MxM matrix using the previously developed `Compute_Integral` function.

Pseudocode:

```
function Integral_Matrix(matrix, M, p, q, r)
    # Constants
    N = 100  # Number of steps for accuracy

    # Initialize loop counter i = 0
    i = 0

    # Matrix processing loop
    while i < M * M do
        # Call Compute_Integral for each element
        result = Compute_Integral(p, q, r, N)

        # Store the result back into the matrix at index i
        matrix[i] = result

        # Increment the counter
        i = i + 1

    # End of matrix loop
    return matrix
```

Design Decisions

Simple Matrix Structure: We used a single list instead of something complex, making it easier and faster to work with.

Modular and Flexible: The integral calculation is kept separate, which makes it easier to handle. The function can work with all kinds of numbers—positive, negative, or decimals.

Handles Edge Cases Well: If the matrix is empty (M = 0), the function won't crash; it just returns an empty result, making it more reliable.

Conclusions

This assignment really taught us a lot about working with matrix operations and calculating integrals. we learned that simplifying complex problems, like using one list for a matrix instead of making it too complicated idea was inspired by stackoverflow comment section.

Breaking the work into smaller parts, like handling the integral separately, made it easier to manage. we also saw how important it is to think about all the different cases,
like when M = 0, so the function doesn't fail.

Problems Encountered

Looping through the matrix as a single array was also tricky. we made mistakes with indexing, which messed up my updates and required lots of debugging.
passing all the parameters was sometimes confusing.

Finally, handling edge cases like an empty matrix (M = 0) was a challenge. These issues taught us the importance of stack management, clear indexing, and handling all types of inputs.

Estimation of Time Spent

5-6.5 Hours without test cases

Test Plan

Test 1: Basic Test
- Description: Small 2x2 matrix to test functionality.
- Input: M = 2, p = 1, q = 1, r = 1
- Expected: 2x2 matrix with the same value everywhere.
- What Happened: Matrix had all correct values.

Test 2: Empty Matrix (Edge Case)
- Description: Test with M = 0 to see if it handles empty input.
- Input: M = 0, p = 2, q = 2, r = 2
- Expected: Should be empty.
- What Happened: Returned empty matrix, no issues.

Test 3: Large Matrix Test
- Description: Big test with a 1000x1000 matrix.
- Input: M = 1000, p = 3, q = 5, r = 7
- Expected: Large matrix filled with computed values.
- What Happened: Kept running for hours honestly idk if the test is failure or no

Test 4: Negative Values
- Description: Test with negative parameters.
- Input: M = 3, p = -2, q = -3, r = -4
- Expected: A 3x3 matrix with computed negative values.
- What Happened: Handled negative values just fine.

Test 5: Single Element Test
- Description: One element to test small input.
- Input: M = 1, p = 10, q = 20, r = 30
- Expected: A 1x1 matrix with computed result.
- What Happened: Got the correct value.

Test 6: Extreme Values
- Description: Test with large numbers in parameters.
- Input: M = 5, p = 1000, q = 2000, r = 3000
- Expected: A 5x5 matrix filled with large computed values.
- What Happened: Worked as expected but for a long time so we stopped the process and somehow final answer was good