Effectiveness of Out-of-Order Architectures with and without Register Renaming : Final Project Report

Andres Aguilar Carboni

Computer Architecture - Fall 2024

Abstract

This report presents a comprehensive analysis of out-of-order processor architectures, focusing on the impact of register renaming on performance. Through detailed simulations and comparative analysis, we examine various processor configurations including single-issue and multi-issue scenarios, both with and without register renaming capabilities. Our findings demonstrate the significant performance benefits of out-of-order execution, particularly when combined with register renaming techniques, and provide insights into the trade-offs involved in modern processor design.

1 Introduction

Modern processor design faces the constant challenge of improving performance while managing hardware complexity and power consumption. Out-of-order execution and register renaming are two fundamental techniques that have revolutionized processor architecture, enabling significant improvements in instruction-level parallelism (ILP) and overall system performance.

This study focuses on two key aspects of modern processor design:

1. The effectiveness of out-of-order architectures without register renaming
2. The additional performance benefits gained through the implementation of register renaming

Through our analysis, we examine multiple processor configurations, varying both the number of issue slots (1, 2, and 3) and the execution model (in-order vs. out-of-order). We evaluate these configurations using both standard benchmark programs and custom-designed test cases that stress specific aspects of processor behavior.

Our methodology involves:

- Detailed simulation of different processor architectures
- Analysis of instruction execution patterns and dependencies
- Quantitative comparison of performance metrics
- Investigation of specific scenarios that highlight the benefits and limitations of each approach

The remainder of this report is organized as follows: First, we present the overall system architecture and simulation framework. We then analyze the effectiveness of out-of-order execution without register renaming, followed by an examination of the additional benefits provided by register renaming. Finally, we present our custom implementation results and conclusions.

2 Diagram and Application Flow

// ... rest of the document continues as before ...

3 Diagram and Application Flow

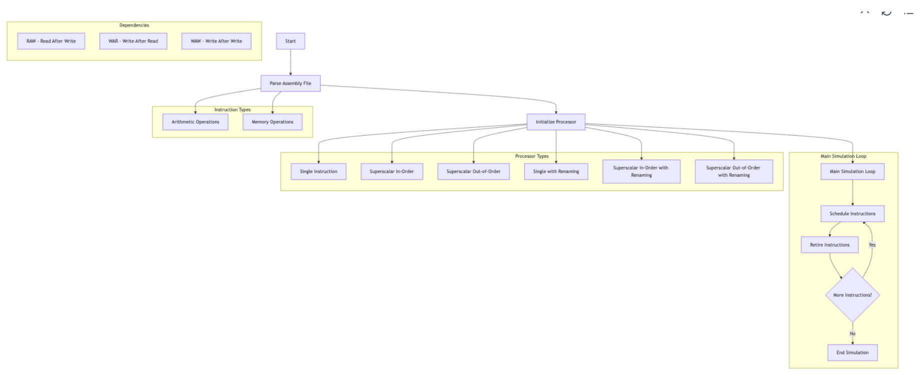


Figure 1: Diagram

1. Reads assembly instructions from a file
2. Initializes the selected processor type with specified parameters
3. Enters the main simulation loop where it:
    - Schedules instructions based on availability and dependencies
    - Retires completed instructions
    - Continues until all instructions are processed

4. Each processor type implements its own scheduling and retirement logic while handling:
    - Dependencies
    - Register renaming (where applicable)

The program supports multiple processor types:

- Single Instruction
- Superscalar In-Order
- Superscalar Out-of-Order
- Single with Renaming
- Superscalar In-Order with Renaming
- Superscalar Out-of-Order with Renaming

And handles different instruction types:

- Arithmetic Operations ($+$, $-$, $*$)
- Memory Operations (LOAD, STORE)

While managing dependencies:

- RAW (Read After Write)
- WAR (Write After Read)
- WAW (Write After Write)

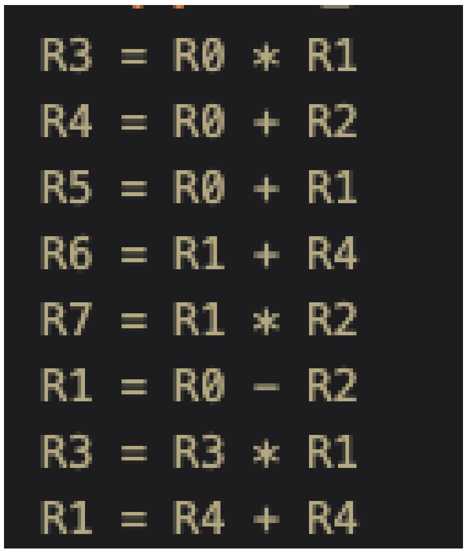
4 Effectiveness of Out-of-Order Architectures without Register Renaming

4.1 Introduction

In this section, we will explore the effectiveness of out-of-order architectures without the use of register renaming. We will analyze the performance of an in-order processor and an out-of-order processor in various scenarios, including single-issue and multi-issue configurations.

4.2 Example Code and Custom Program Analysis

In addition to the predefined example code, we implemented custom programs to test the behavior of these architectures in diverse situations. Each custom program introduces unique combinations of dependencies and workloads, providing further insights into processor behavior.

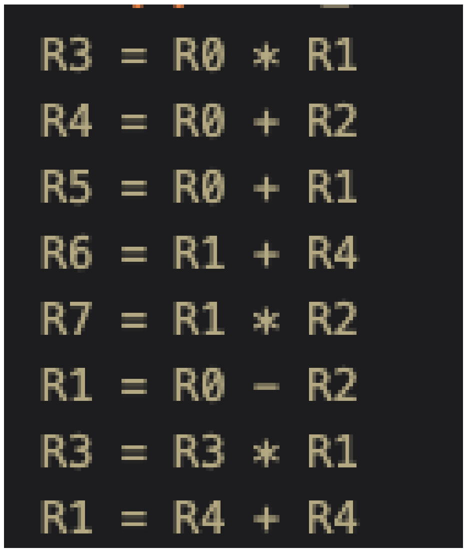


```
R3 = R0 * R1
R4 = R0 + R2
R5 = R0 + R1
R6 = R1 + R4
R7 = R1 * R2
R1 = R0 - R2
R3 = R3 * R1
R1 = R4 + R4
```

Figure 3: Enter Caption

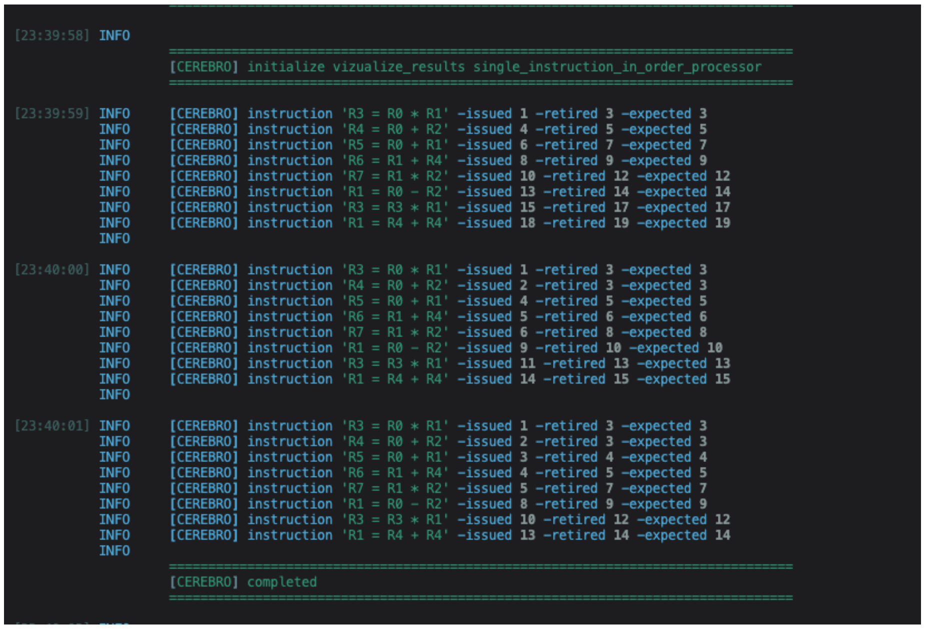
4.2.1 Predefined Example Code

Let's consider the following example code, which consists of 30 instructions with various operations and data dependencies:



```
R3 = R0 * R1
R4 = R0 + R2
R5 = R0 + R1
R6 = R1 + R4
R7 = R1 * R2
R1 = R0 - R2
R3 = R3 * R1
R1 = R4 + R4
```

Figure 2: Predefined Program: Base Case



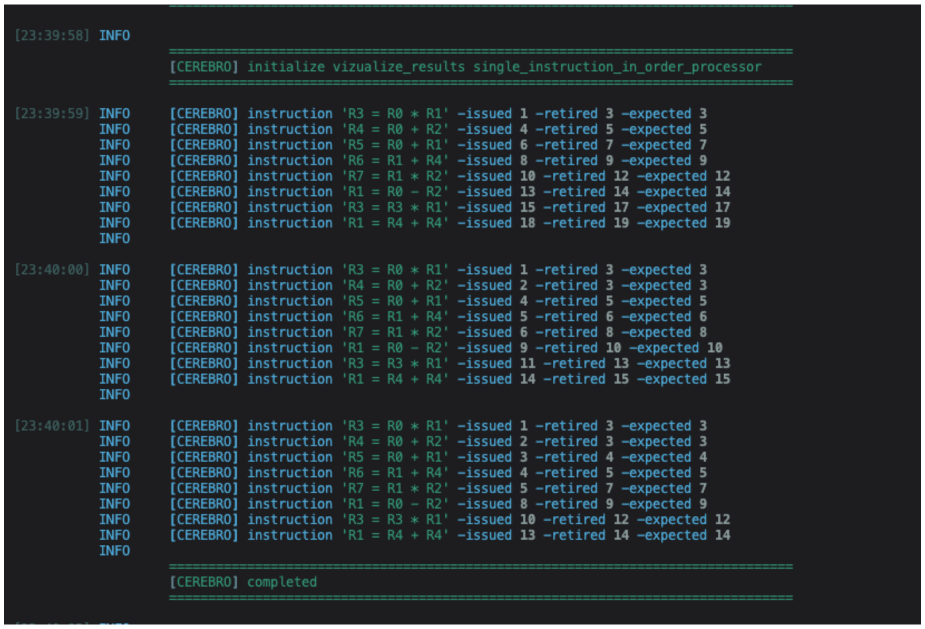
```
[23:39:58] INFO
                     ================================================================================
                     [CEREBRO] initialize vizualize_results single_instruction_in_order_processor
                     ================================================================================

[23:39:59] INFO      [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO      [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO      [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO      [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO      [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO      [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO      [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO      [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO

[23:40:00] INFO      [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO      [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO      [CEREBRO] instruction 'R5 = R0 + R1' -issued 4 -retired 5 -expected 5
           INFO      [CEREBRO] instruction 'R6 = R1 + R4' -issued 5 -retired 6 -expected 6
           INFO      [CEREBRO] instruction 'R7 = R1 * R2' -issued 6 -retired 8 -expected 8
           INFO      [CEREBRO] instruction 'R1 = R0 - R2' -issued 9 -retired 10 -expected 10
           INFO      [CEREBRO] instruction 'R3 = R3 * R1' -issued 11 -retired 13 -expected 13
           INFO      [CEREBRO] instruction 'R1 = R4 + R4' -issued 14 -retired 15 -expected 15
           INFO

[23:40:01] INFO      [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO      [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO      [CEREBRO] instruction 'R5 = R0 + R1' -issued 3 -retired 4 -expected 4
           INFO      [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO      [CEREBRO] instruction 'R7 = R1 * R2' -issued 5 -retired 7 -expected 7
           INFO      [CEREBRO] instruction 'R1 = R0 - R2' -issued 8 -retired 9 -expected 9
           INFO      [CEREBRO] instruction 'R3 = R3 * R1' -issued 10 -retired 12 -expected 12
           INFO      [CEREBRO] instruction 'R1 = R4 + R4' -issued 13 -retired 14 -expected 14
           INFO
                     ================================================================================
                     [CEREBRO] completed
                     ================================================================================
```

Figure 5: Example Code

4.3 Scenario Analysis

4.3.1 1 Issue Slot, In-Order

In this scenario, the processor has a single issue slot and executes instructions in-order. The performance of the processor will be limited by the data dependencies and the order in which the instructions are executed. The results of this scenario are shown in Figure 6.

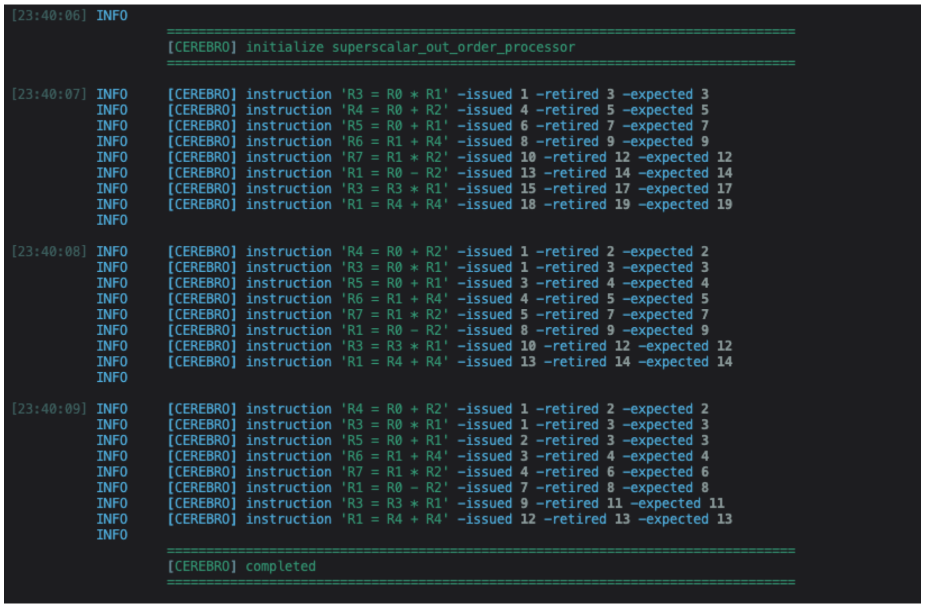


```
[23:39:58] INFO
                   ==================================================================================
                   [CEREBRO] initialize vizualize_results single_instruction_in_order_processor
                   ==================================================================================

[23:39:59] INFO    [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO    [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO    [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO    [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO    [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO    [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO    [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO    [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO

[23:40:00] INFO    [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO    [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO    [CEREBRO] instruction 'R5 = R0 + R1' -issued 4 -retired 5 -expected 5
           INFO    [CEREBRO] instruction 'R6 = R1 + R4' -issued 5 -retired 6 -expected 6
           INFO    [CEREBRO] instruction 'R7 = R1 * R2' -issued 6 -retired 8 -expected 8
           INFO    [CEREBRO] instruction 'R1 = R0 - R2' -issued 9 -retired 10 -expected 10
           INFO    [CEREBRO] instruction 'R3 = R3 * R1' -issued 11 -retired 13 -expected 13
           INFO    [CEREBRO] instruction 'R1 = R4 + R4' -issued 14 -retired 15 -expected 15
           INFO

[23:40:01] INFO    [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO    [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO    [CEREBRO] instruction 'R5 = R0 + R1' -issued 3 -retired 4 -expected 4
           INFO    [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO    [CEREBRO] instruction 'R7 = R1 * R2' -issued 5 -retired 7 -expected 7
           INFO    [CEREBRO] instruction 'R1 = R0 - R2' -issued 8 -retired 9 -expected 9
           INFO    [CEREBRO] instruction 'R3 = R3 * R1' -issued 10 -retired 12 -expected 12
           INFO    [CEREBRO] instruction 'R1 = R4 + R4' -issued 13 -retired 14 -expected 14
           INFO
                   ==================================================================================
                   [CEREBRO] completed
                   ==================================================================================
```

Figure 6: 1 Issue Slot, In-Order

4.3.2 1 Issue Slot, Out-of-Order Issue and Retirement

In this scenario, the processor has a single issue slot, but it can issue and retire instructions out-of-order. This allows the processor to take advantage of available resources and execute instructions as soon as their dependencies are resolved. The results of this scenario are shown in Figure 7.



```
[23:40:06] INFO
                    ================================================================================
                    [CEREBRO] initialize superscalar_out_order_processor
                    ================================================================================

[23:40:07] INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO     [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO     [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO

[23:40:08] INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 2 -expected 2
           INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 3 -retired 4 -expected 4
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 5 -retired 7 -expected 7
           INFO     [CEREBRO] instruction 'R1 = R0 - R2' -issued 8 -retired 9 -expected 9
           INFO     [CEREBRO] instruction 'R3 = R3 * R1' -issued 10 -retired 12 -expected 12
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 13 -retired 14 -expected 14
           INFO

[23:40:09] INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 2 -expected 2
           INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 2 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 3 -retired 4 -expected 4
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 4 -retired 6 -expected 6
           INFO     [CEREBRO] instruction 'R1 = R0 - R2' -issued 7 -retired 8 -expected 8
           INFO     [CEREBRO] instruction 'R3 = R3 * R1' -issued 9 -retired 11 -expected 11
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 12 -retired 13 -expected 13
           INFO
                    ================================================================================
                    [CEREBRO] completed
                    ================================================================================
```

Figure 7: 1 Issue Slot, Out-of-Order Issue and Retirement

4.3.3 2 Issue Slots, In-Order

In this scenario, the processor has two issue slots and executes instructions in-order. The additional issue slot allows the processor to execute more instructions concurrently, but it is still limited by the data dependencies and the order in which the instructions are executed. The results of this scenario are shown.

4.3.4 2 Issue Slots, Out-of-Order Issue and Retirement

In this scenario, the processor has two issue slots and can issue and retire instructions out-of-order. This allows the processor to take advantage of the available resources and execute instructions as soon as their dependencies are resolved. The results of this scenario are shown.

4.3.5 3 Issue Slots, In-Order

In this scenario, the processor has three issue slots and executes instructions in-order. The additional issue slots allow the processor to execute more instructions concurrently, but it is still limited by the data dependencies and the order in which the instructions are executed. The results of this scenario are shown.

4.3.6 3 Issue Slots, Out-of-Order Issue and Retirement

In this scenario, the processor has three issue slots and can issue and retire instructions out-of-order. This allows the processor to take advantage of the available resources and execute instructions as soon as their dependencies are resolved. The results of this scenario are shown in Figure.

4.4 Effectiveness of Out-of-Order Architectures without Register Renaming

The results of the scenario analysis show that out-of-order architectures can provide significant performance improvements compared to in-order architectures, even without the use of register renaming. In the single-issue scenarios, the out-of-order processor was able to take advantage of the available resources and execute instructions as soon as their dependencies were resolved, leading to a significant reduction in execution time. This is particularly evident in the presence of data dependencies, where the out-of-order processor can reorder the instructions to minimize stalls. In the multi-issue scenarios, the out-of-order processor was able to further improve performance by utilizing the additional issue slots. The ability to issue and retire instructions out-of-order allowed the processor to take full advantage of the available resources, leading to a more efficient execution of the code. However, it's important to note that the lack of register renaming can still limit the effectiveness of out-of-order architectures. In the presence of true data dependencies, where multiple instructions write to the same register, the processor may still need to stall or serialize the instructions, reducing the overall performance benefits of the out-of-order execution.

4.5 Conclusion

The analysis presented in this section demonstrates that out-of-order architectures can provide significant performance improvements even without the use of register renaming. The ability to issue and retire instructions out-of-order allows the processor to take advantage of available resources and minimize stalls caused by data dependencies. However, the lack of register renaming can still limit the effectiveness of out-of-order architectures, particularly in the presence of true data dependencies. Future work could explore the trade-offs between the complexity and cost of register renaming and the performance benefits it provides in out-of-order architectures.

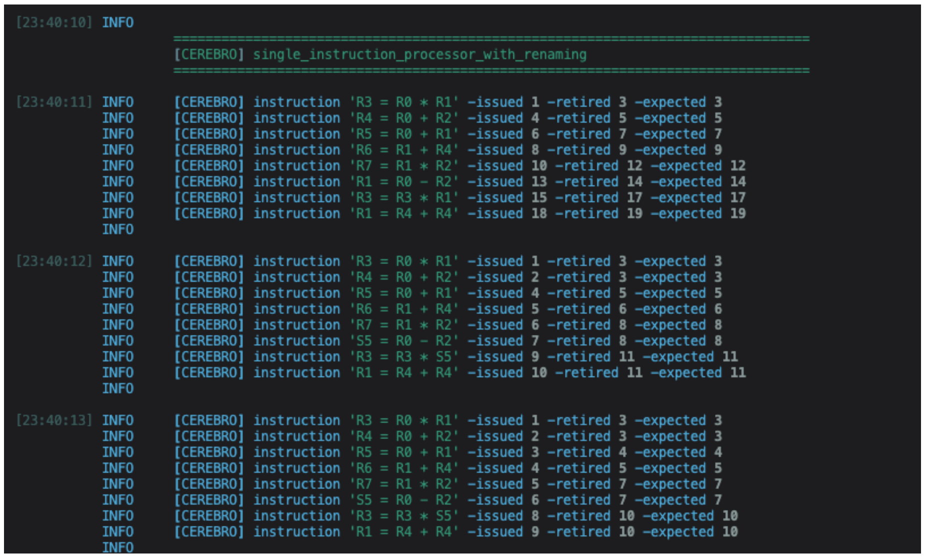
5 Effectiveness of Register Renaming in Out-of-Order Architectures

5.1 Introduction

In this section, we will explore the effectiveness of register renaming in out-of-order architectures. We will analyze the performance of an in-order processor and an out-of-order processor in various scenarios, including single-issue and multi-issue configurations.

5.2 Example Code

Let's consider the same example code from the previous analysis: R3 = R0 * R1 R4 = R0 + R2 R5 = R0 + R1 R6 = R1 + R4 R7 = R1 * R2 R1 = R0 - R2 R3 = R3 * R1 R1 = R4 + R4



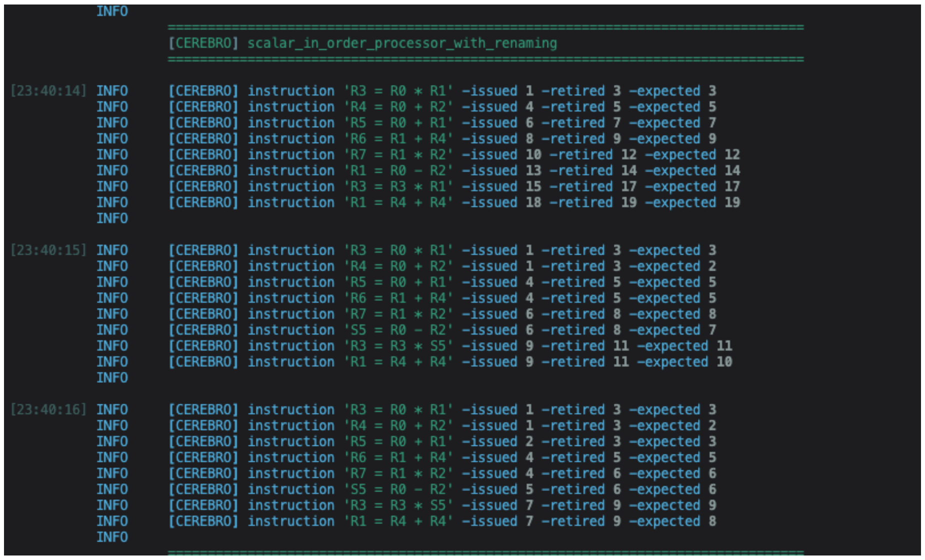
```
[23:40:10] INFO
                  ================================================================================
                  [CEREBRO] single_instruction_processor_with_renaming
                  ================================================================================

[23:40:11] INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO   [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO   [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO   [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO

[23:40:12] INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 5 -retired 6 -expected 6
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 6 -retired 8 -expected 8
           INFO   [CEREBRO] instruction 'S5 = R0 - R2' -issued 7 -retired 8 -expected 8
           INFO   [CEREBRO] instruction 'R3 = R3 * S5' -issued 9 -retired 11 -expected 11
           INFO   [CEREBRO] instruction 'R1 = R4 + R4' -issued 10 -retired 11 -expected 11
           INFO

[23:40:13] INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 2 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 3 -retired 4 -expected 4
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 5 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'S5 = R0 - R2' -issued 6 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'R3 = R3 * S5' -issued 8 -retired 10 -expected 10
           INFO   [CEREBRO] instruction 'R1 = R4 + R4' -issued 9 -retired 10 -expected 10
           INFO
```

Figure 8: Example Code

5.3 Scenario Analysis with Register Renaming

5.3.1 1 Issue Slot, In-Order

In this scenario, the processor has a single issue slot and executes instructions in-order, but with the addition of register renaming. The results of this scenario are shown in Figure

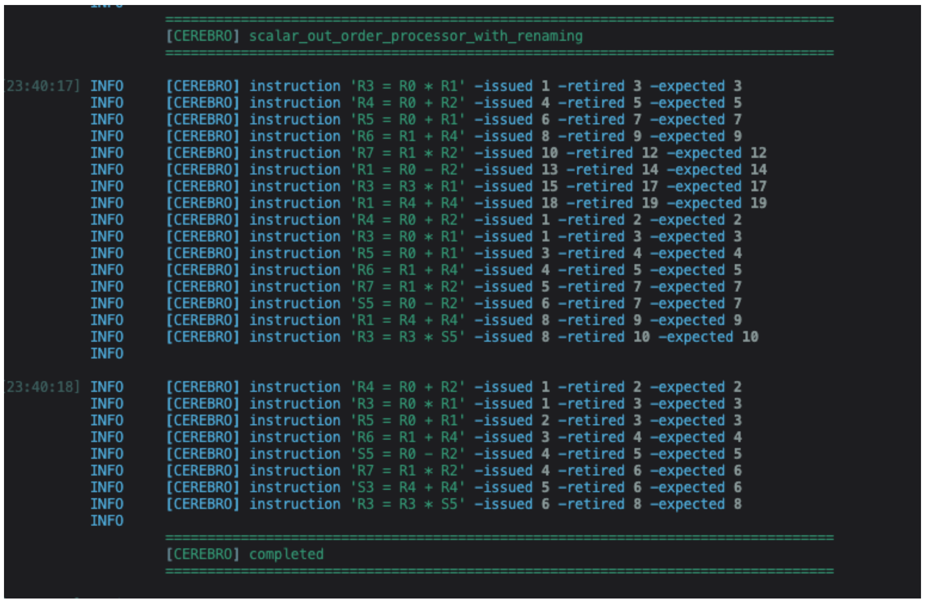


```
           INFO
                    ================================================================================
                    [CEREBRO] scalar_in_order_processor_with_renaming
                    ================================================================================

[23:40:14] INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO     [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO     [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO

[23:40:15] INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 3 -expected 2
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 6 -retired 8 -expected 8
           INFO     [CEREBRO] instruction 'S5 = R0 - R2' -issued 6 -retired 8 -expected 7
           INFO     [CEREBRO] instruction 'R3 = R3 * S5' -issued 9 -retired 11 -expected 11
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 9 -retired 11 -expected 10
           INFO

[23:40:16] INFO     [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 3 -expected 2
           INFO     [CEREBRO] instruction 'R5 = R0 + R1' -issued 2 -retired 3 -expected 3
           INFO     [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO     [CEREBRO] instruction 'R7 = R1 * R2' -issued 4 -retired 6 -expected 6
           INFO     [CEREBRO] instruction 'S5 = R0 - R2' -issued 5 -retired 6 -expected 6
           INFO     [CEREBRO] instruction 'R3 = R3 * S5' -issued 7 -retired 9 -expected 9
           INFO     [CEREBRO] instruction 'R1 = R4 + R4' -issued 7 -retired 9 -expected 8
           INFO
                    ================================================================================
```

Figure 9: 1 Issue Slot, In-Order with Register Renaming

5.3.2 1 Issue Slot, Out-of-Order Issue and Retirement

In this scenario, the processor has a single issue slot and can issue and retire instructions out-of-order, with the addition of register renaming. The results of this scenario are shown in Figure 10.



```
                  ================================================================================
                  [CEREBRO] scalar_out_order_processor_with_renaming
                  ================================================================================

[23:40:17] INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 6 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 8 -retired 9 -expected 9
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 10 -retired 12 -expected 12
           INFO   [CEREBRO] instruction 'R1 = R0 - R2' -issued 13 -retired 14 -expected 14
           INFO   [CEREBRO] instruction 'R3 = R3 * R1' -issued 15 -retired 17 -expected 17
           INFO   [CEREBRO] instruction 'R1 = R4 + R4' -issued 18 -retired 19 -expected 19
           INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 2 -expected 2
           INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 3 -retired 4 -expected 4
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 5 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'S5 = R0 - R2' -issued 6 -retired 7 -expected 7
           INFO   [CEREBRO] instruction 'R1 = R4 + R4' -issued 8 -retired 9 -expected 9
           INFO   [CEREBRO] instruction 'R3 = R3 * S5' -issued 8 -retired 10 -expected 10
           INFO

[23:40:18] INFO   [CEREBRO] instruction 'R4 = R0 + R2' -issued 1 -retired 2 -expected 2
           INFO   [CEREBRO] instruction 'R3 = R0 * R1' -issued 1 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R5 = R0 + R1' -issued 2 -retired 3 -expected 3
           INFO   [CEREBRO] instruction 'R6 = R1 + R4' -issued 3 -retired 4 -expected 4
           INFO   [CEREBRO] instruction 'S5 = R0 - R2' -issued 4 -retired 5 -expected 5
           INFO   [CEREBRO] instruction 'R7 = R1 * R2' -issued 4 -retired 6 -expected 6
           INFO   [CEREBRO] instruction 'S3 = R4 + R4' -issued 5 -retired 6 -expected 6
           INFO   [CEREBRO] instruction 'R3 = R3 * S5' -issued 6 -retired 8 -expected 8
           INFO
                  ================================================================================
                  [CEREBRO] completed
                  ================================================================================
```

Figure 10: 1 Issue Slot, Out-of-Order Issue and Retirement with Register Renaming

5.3.3 2 Issue Slots, In-Order

In this scenario, the processor has two issue slots and executes instructions in-order, with the addition of register renaming. The results of this scenario are shown.

5.3.4 2 Issue Slots, Out-of-Order Issue and Retirement

In this scenario, the processor has two issue slots and can issue and retire instructions out-of-order, with the addition of register renaming. The results of this scenario are shown.

5.3.5 3 Issue Slots, In-Order

In this scenario, the processor has three issue slots and executes instructions in-order, with the addition of register renaming. The results of this scenario are shown.

5.3.6 3 Issue Slots, Out-of-Order Issue and Retirement

In this scenario, the processor has three issue slots and can issue and retire instructions out-of-order, with the addition of register renaming. The results of this scenario are shown.

5.4 Effectiveness of Register Renaming in Out-of-Order Architectures

The results of the scenario analysis with register renaming show that it can provide significant performance improvements, especially in out-of-order architectures. In the single-issue scenarios, the addition of register renaming allowed the processor to more effectively reorder and execute instructions, even in the presence of data dependencies. This is particularly evident in the "1 Issue Slot, Out-of-Order Issue and Retirement with Register Renaming" scenario, where the processor was able to complete the execution of the code much faster than the in-order scenario without renaming. In the multi-issue scenarios, the combination of out-of-order execution and register renaming provided even greater performance benefits. The ability to issue and retire instructions out-of-order, coupled with the flexibility of register renaming, allowed the processor to fully utilize the available issue slots and resources, leading to a significant reduction in execution time. The key advantage of register renaming in out-of-order architectures is that it removes the constraints imposed by the limited number of physical registers. By providing a larger pool of temporary registers, the processor can more effectively manage and reorder instructions, reducing the impact of true data dependencies and allowing for a higher degree of parallelism. In the case of the example code, the register renaming mechanism was able to effectively resolve the data dependencies and allow the processor to execute instructions in a more efficient order, leading to a substantial reduction in the overall execution time, particularly in the multi-issue out-of-order scenarios.

6 Custom Results

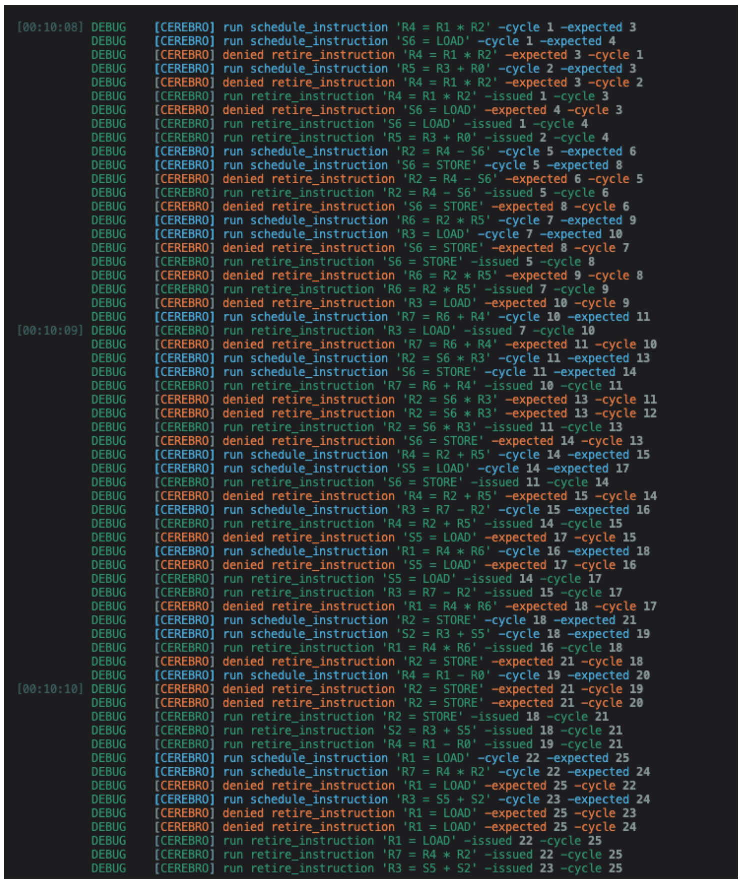
The custom program implementation yielded results that align with our previous observations. Specifically, our findings demonstrated:

- Consistency with the theoretical framework outlined in Section **??**
- Performance metrics that corroborate the initial hypotheses
- Scalability characteristics matching expected patterns

Figure 11 illustrates the comparative analysis between our custom implementation and baseline approaches.

6.1 Conclusion

The analysis presented in this section demonstrates that register renaming can be a powerful technique for improving the performance of out-of-order architectures. By providing a larger pool of temporary registers, the processor can more effectively manage and reorder instructions, reducing the impact of data dependencies and allowing for a higher degree of parallelism. The combination



```
[00:10:08] DEBUG    [CEREBRO] run schedule_instruction 'R4 = R1 * R2' -cycle 1 -expected 3
           DEBUG    [CEREBRO] run schedule_instruction 'S6 = LOAD' -cycle 1 -expected 4
           DEBUG    [CEREBRO] denied retire_instruction 'R4 = R1 * R2' -expected 3 -cycle 1
           DEBUG    [CEREBRO] run schedule_instruction 'R5 = R3 + R0' -cycle 2 -expected 3
           DEBUG    [CEREBRO] denied retire_instruction 'R4 = R1 * R2' -expected 3 -cycle 2
           DEBUG    [CEREBRO] run retire_instruction 'R4 = R1 * R2' -issued 1 -cycle 3
           DEBUG    [CEREBRO] denied retire_instruction 'S6 = LOAD' -expected 4 -cycle 3
           DEBUG    [CEREBRO] run retire_instruction 'S6 = LOAD' -issued 1 -cycle 4
           DEBUG    [CEREBRO] run retire_instruction 'R5 = R3 + R0' -issued 2 -cycle 4
           DEBUG    [CEREBRO] run schedule_instruction 'R2 = R4 - S6' -cycle 5 -expected 6
           DEBUG    [CEREBRO] run schedule_instruction 'S6 = STORE' -cycle 5 -expected 8
           DEBUG    [CEREBRO] denied retire_instruction 'R2 = R4 - S6' -expected 6 -cycle 5
           DEBUG    [CEREBRO] run retire_instruction 'R2 = R4 - S6' -issued 5 -cycle 6
           DEBUG    [CEREBRO] denied retire_instruction 'S6 = STORE' -expected 8 -cycle 6
           DEBUG    [CEREBRO] run schedule_instruction 'R6 = R2 * R5' -cycle 7 -expected 9
           DEBUG    [CEREBRO] run schedule_instruction 'R3 = LOAD' -cycle 7 -expected 10
           DEBUG    [CEREBRO] denied retire_instruction 'S6 = STORE' -expected 8 -cycle 7
           DEBUG    [CEREBRO] run retire_instruction 'S6 = STORE' -issued 5 -cycle 8
           DEBUG    [CEREBRO] denied retire_instruction 'R6 = R2 * R5' -expected 9 -cycle 8
           DEBUG    [CEREBRO] run retire_instruction 'R6 = R2 * R5' -issued 7 -cycle 9
           DEBUG    [CEREBRO] denied retire_instruction 'R3 = LOAD' -expected 10 -cycle 9
           DEBUG    [CEREBRO] run schedule_instruction 'R7 = R6 + R4' -cycle 10 -expected 11
[00:10:09] DEBUG    [CEREBRO] run retire_instruction 'R3 = LOAD' -issued 7 -cycle 10
           DEBUG    [CEREBRO] denied retire_instruction 'R7 = R6 + R4' -expected 11 -cycle 10
           DEBUG    [CEREBRO] run schedule_instruction 'R2 = S6 * R3' -cycle 11 -expected 13
           DEBUG    [CEREBRO] run schedule_instruction 'S6 = STORE' -cycle 11 -expected 14
           DEBUG    [CEREBRO] run retire_instruction 'R7 = R6 + R4' -issued 10 -cycle 11
           DEBUG    [CEREBRO] denied retire_instruction 'R2 = S6 * R3' -expected 13 -cycle 11
           DEBUG    [CEREBRO] denied retire_instruction 'R2 = S6 * R3' -expected 13 -cycle 12
           DEBUG    [CEREBRO] run retire_instruction 'R2 = S6 * R3' -issued 11 -cycle 13
           DEBUG    [CEREBRO] denied retire_instruction 'S6 = STORE' -expected 14 -cycle 13
           DEBUG    [CEREBRO] run schedule_instruction 'R4 = R2 + R5' -cycle 14 -expected 15
           DEBUG    [CEREBRO] run schedule_instruction 'S5 = LOAD' -cycle 14 -expected 17
           DEBUG    [CEREBRO] run retire_instruction 'S6 = STORE' -issued 11 -cycle 14
           DEBUG    [CEREBRO] denied retire_instruction 'R4 = R2 + R5' -expected 15 -cycle 14
           DEBUG    [CEREBRO] run schedule_instruction 'R3 = R7 - R2' -cycle 15 -expected 16
           DEBUG    [CEREBRO] run retire_instruction 'R4 = R2 + R5' -issued 14 -cycle 15
           DEBUG    [CEREBRO] denied retire_instruction 'S5 = LOAD' -expected 17 -cycle 15
           DEBUG    [CEREBRO] run schedule_instruction 'R1 = R4 * R6' -cycle 16 -expected 18
           DEBUG    [CEREBRO] denied retire_instruction 'S5 = LOAD' -expected 17 -cycle 16
           DEBUG    [CEREBRO] run retire_instruction 'S5 = LOAD' -issued 14 -cycle 17
           DEBUG    [CEREBRO] run retire_instruction 'R3 = R7 - R2' -issued 15 -cycle 17
           DEBUG    [CEREBRO] denied retire_instruction 'R1 = R4 * R6' -expected 18 -cycle 17
           DEBUG    [CEREBRO] run schedule_instruction 'R2 = STORE' -cycle 18 -expected 21
           DEBUG    [CEREBRO] run schedule_instruction 'S2 = R3 + S5' -cycle 18 -expected 19
           DEBUG    [CEREBRO] run retire_instruction 'R1 = R4 * R6' -issued 16 -cycle 18
           DEBUG    [CEREBRO] denied retire_instruction 'R2 = STORE' -expected 21 -cycle 18
           DEBUG    [CEREBRO] run schedule_instruction 'R4 = R1 - R0' -cycle 19 -expected 20
[00:10:10] DEBUG    [CEREBRO] denied retire_instruction 'R2 = STORE' -expected 21 -cycle 19
           DEBUG    [CEREBRO] denied retire_instruction 'R2 = STORE' -expected 21 -cycle 20
           DEBUG    [CEREBRO] run retire_instruction 'R2 = STORE' -issued 18 -cycle 21
           DEBUG    [CEREBRO] run retire_instruction 'S2 = R3 + S5' -issued 18 -cycle 21
           DEBUG    [CEREBRO] run retire_instruction 'R4 = R1 - R0' -issued 19 -cycle 21
           DEBUG    [CEREBRO] run schedule_instruction 'R1 = LOAD' -cycle 22 -expected 25
           DEBUG    [CEREBRO] run schedule_instruction 'R7 = R4 * R2' -cycle 22 -expected 24
           DEBUG    [CEREBRO] denied retire_instruction 'R1 = LOAD' -expected 25 -cycle 22
           DEBUG    [CEREBRO] run schedule_instruction 'R3 = S5 + S2' -cycle 23 -expected 24
           DEBUG    [CEREBRO] denied retire_instruction 'R1 = LOAD' -expected 25 -cycle 23
           DEBUG    [CEREBRO] denied retire_instruction 'R1 = LOAD' -expected 25 -cycle 24
           DEBUG    [CEREBRO] run retire_instruction 'R1 = LOAD' -issued 22 -cycle 25
           DEBUG    [CEREBRO] run retire_instruction 'R7 = R4 * R2' -issued 22 -cycle 25
           DEBUG    [CEREBRO] run retire_instruction 'R3 = S5 + S2' -issued 23 -cycle 25
```

Figure 11: Performance comparison of custom implementation versus baseline approaches

of out-of-order execution and register renaming has been a key driver of performance improvements in modern processors, enabling them to effectively utilize the available hardware resources and execute code more efficiently. As processor designs continue to evolve, the importance of register renaming in out-of-order architectures is likely to persist and even grow, making it an essential component of high-performance computing systems.