TDTS10 Computer Architecture

Lab Assignment 3: Superscalar Processors

Axel Strid (axest556)
Dennis Zakrisson (denza625)

Contents

1 Objective

The purpose of this assignment is to get insight on: a superscalar processor; how it can affect the performance of the system; the performance-cost trade-off; and why different programs benefit differently from a superscalar processor.

2 Assignments

In order to perform the assignments, we are looking at the `sim-cycles` output variable from each run in the simulator. This variable represents the total simulation time in cycles and represents the value of performance.

Note: The x-axis on all graphs corresponds to the index of the elements in our data. For example, an x-value of 1 represents the first element, 2 represents the second element, and so on. In this exercise, the x-axis specifically represents values that are powers of two within the allowed range.

Explanation of parameters:

- **decode:width (1-32)**: Defines the maximum number of instructions the CPU can decode in one cycle.
- **issue:width (1-32)**: Specifies the maximum number of instructions that can be sent to execution units in one cycle.
- **commit:width (1-32)**: Refers to the maximum number of instructions that can be completed and written back in one cycle.
- **res:ialu (1-8)**: The number of integer ALUs available for performing arithmetic and logical operations.
- **res:imult (1-8)**: The number of integer multipliers/dividers available for multiplication and division operations.
- **res:fpalu (1-8)**: The number of floating-point ALUs available for performing floating-point arithmetic operations.
- **res:fpmult (1-8)**: The number of floating-point multipliers/dividers available for floating-point multiplication and division.

2.1 Performance/cost trade-off

1. Using scatter type charts, show the impact of each parameter on performance. Make sure Y-axis starts from 0. You can use one chart for decode, issue, and commit; and one for ialu, imult, fpalu, and fpmult.

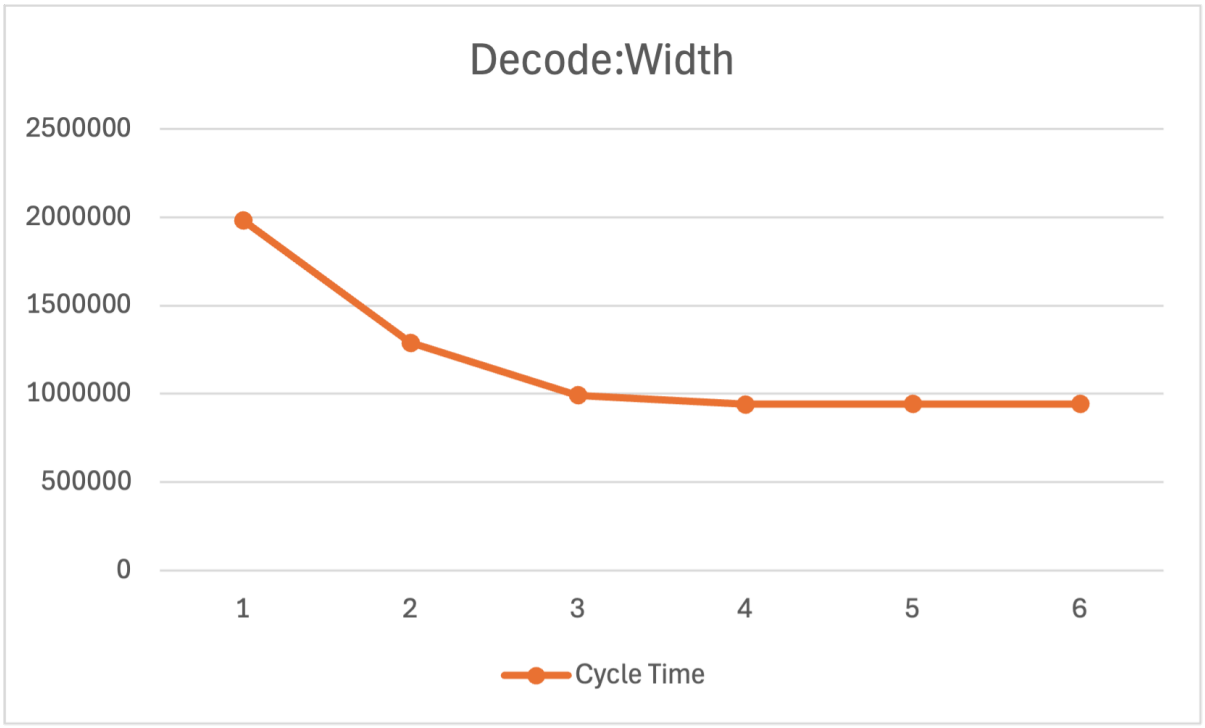

Diagram representing the cycle-time when changing values for decode:width

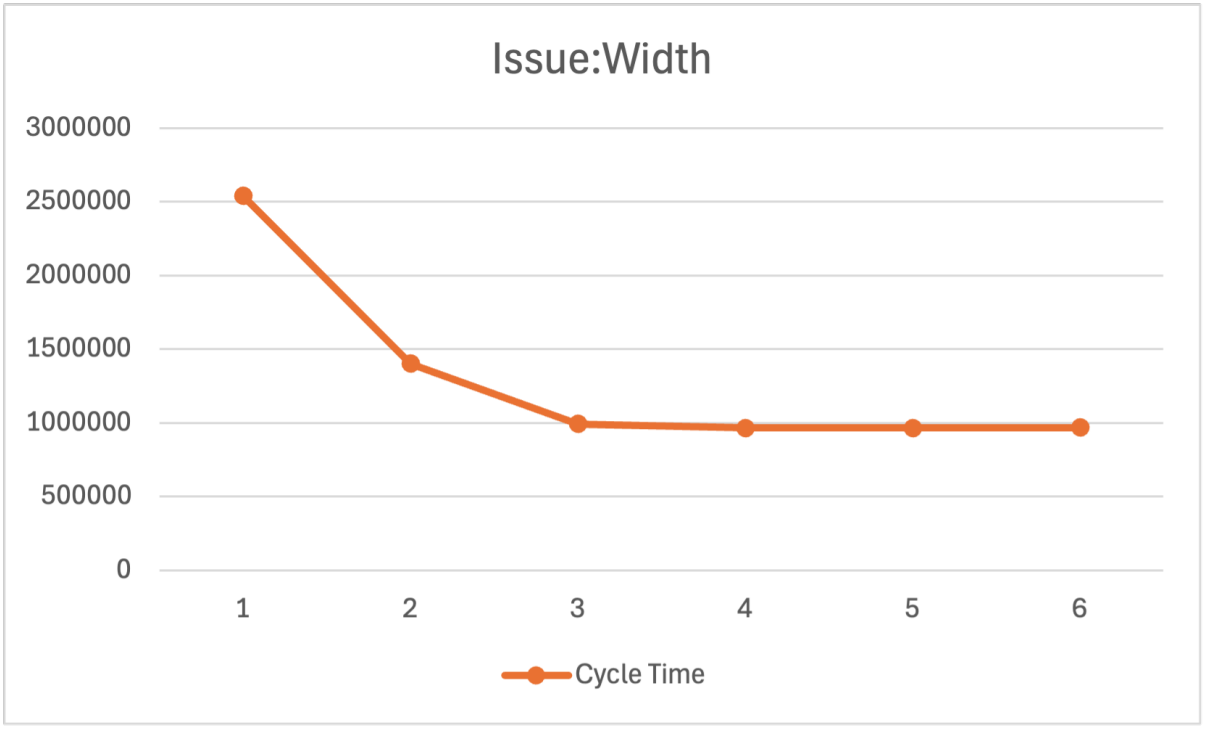

Diagram representing the cycle-time when changing values for issue:width

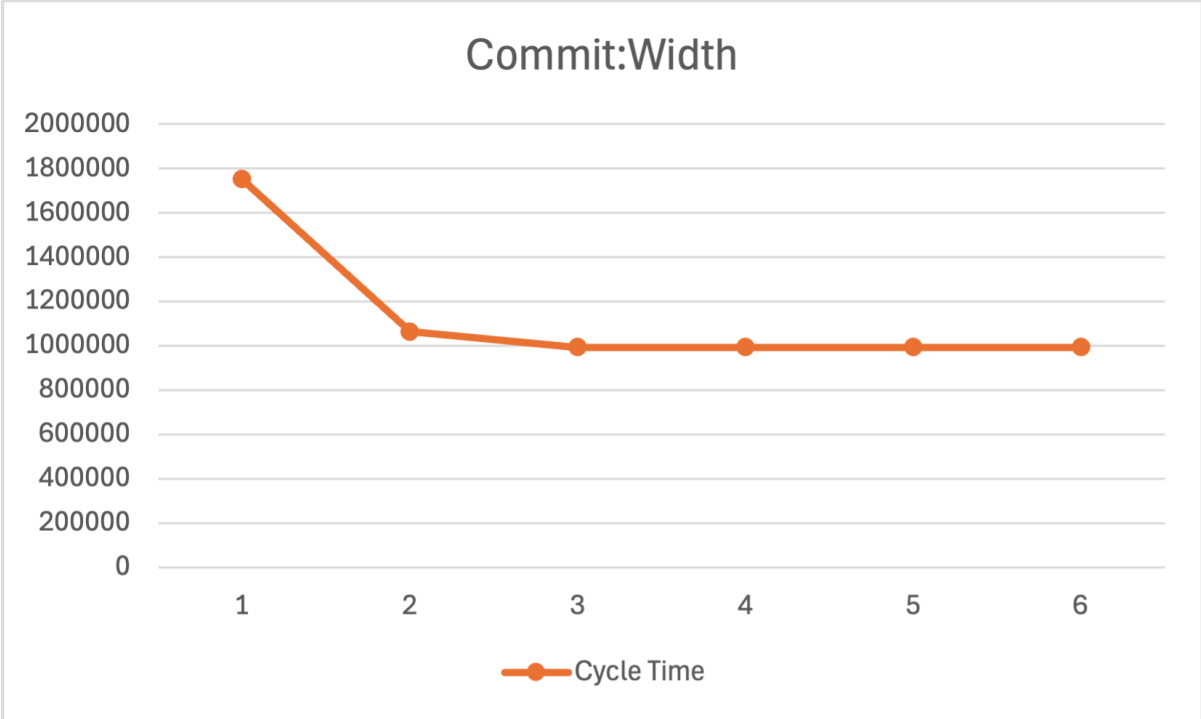

Diagram representing the cycle-time when changing values for commit:width

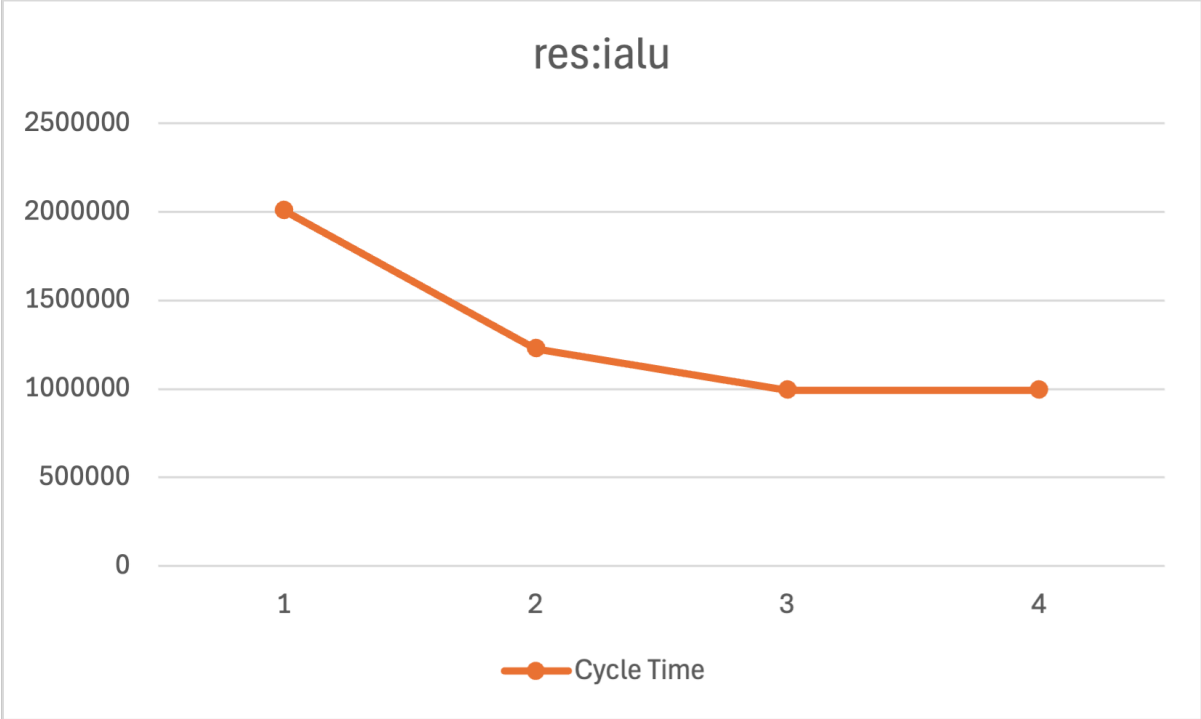

Diagram representing the cycle-time when changing values for res:ialu

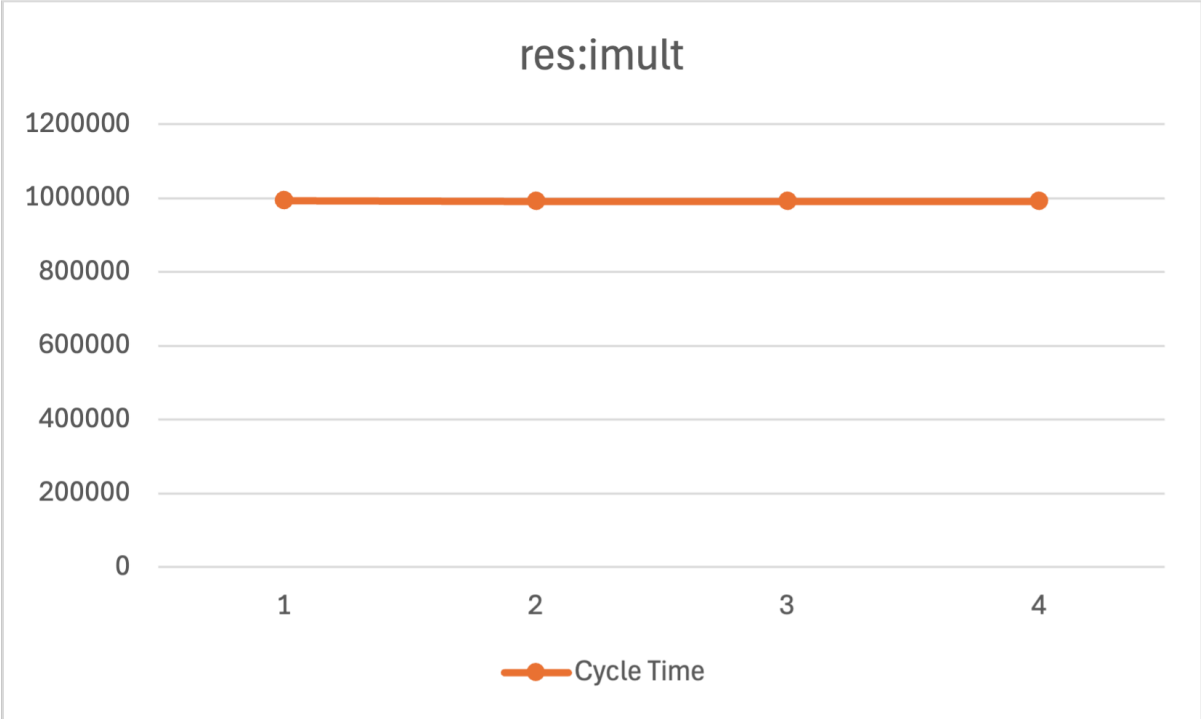

Diagram representing the cycle-time when changing values for res:imult

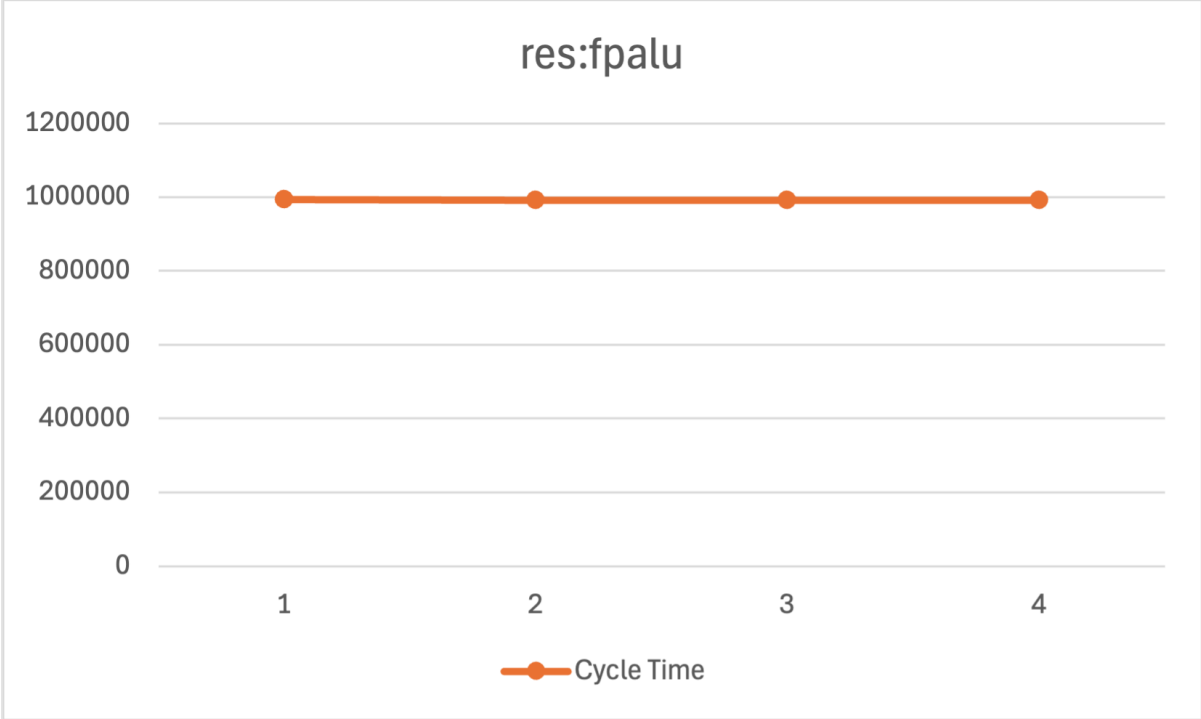

Diagram representing the cycle-time when changing values for res:fpalu

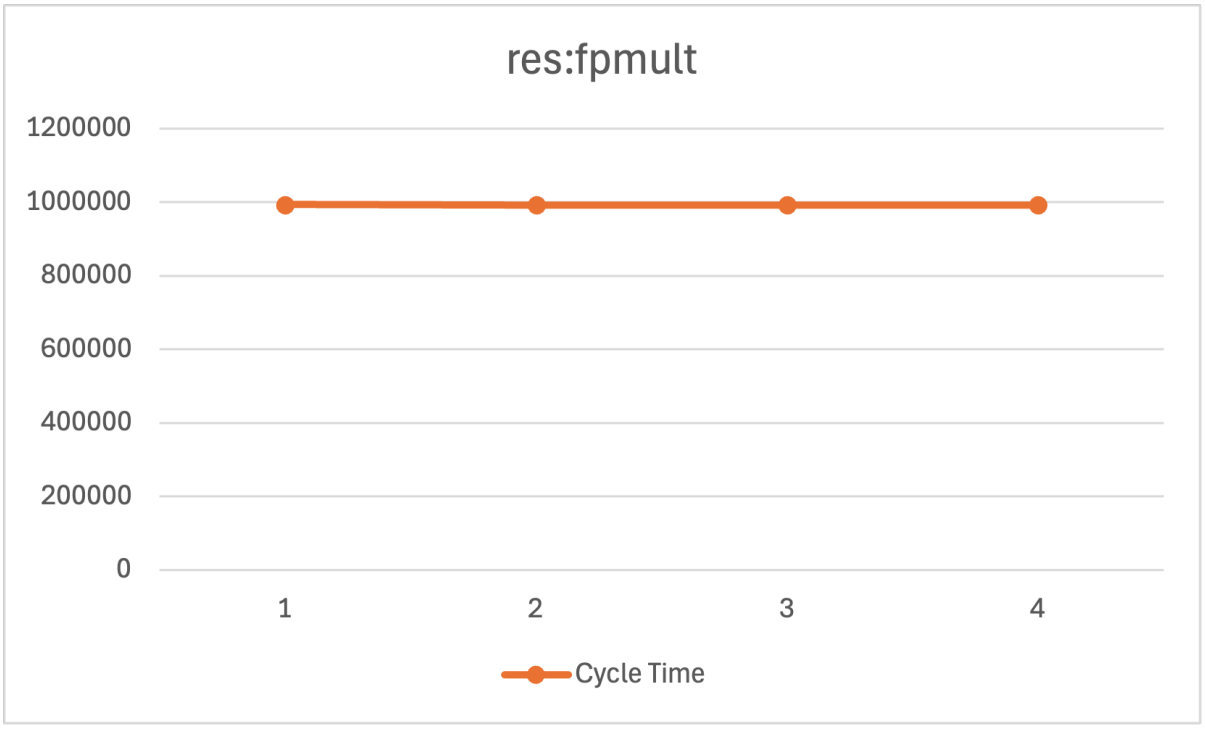

Diagram representing the cycle-time when changing values for res:fpmult

2. Which parameters have the least impacts on performance? Explain the reason.

   Conclusion: The graphs for the parameters of `res:imult, res:fpalu & res:fpmult` remains horizontal across all allowed values, indicating that varying these resources does not significantly affect the overall performance. The workload does not fully utilize these units, and the performance is bottle-necked by other factors.

3. Considering the following cost function, which configuration provides the least cost provided that the total number of cycles must not surpass 1,000,000 by more than 2%?

   The Cost function:

   $$\text{cost} = (\text{decode} + \text{issue} + \text{commit} + \text{ialu} + \text{imult} + \text{fpalu} + \text{fpmult})$$

   Answer: Based on the performance metrics in each graph we choose the input value of the parameter where the graph starts to stabilize. This give us the following cost:

   $$\text{Cost} = (4 + 4 + 4 + 4 + 1 + 1 + 1) = 19$$

4. Looking at the results, what can you conclude about the ILP (Instruction Level Parallelism) degree of the benchmark?

   Answer: ILP is the ability of the processor to execute multiple instructions in parallel.

   `-decode:width , -issue:width , -commit:width , -res:ialu`: Performance improves initially for lower input values of the parameter, but stabilizes in a plateau for higher values. A plateau suggests that there is insufficient instruction-level parallelism in the benchmark to take advantage of wider decode, issue, and commit

stages. This indicates that the benchmark has limited ILP, as the pipeline can no longer find enough independent instructions to exploit beyond a certain width.

`-res:imult , -res:fpalu , -res:fpmult`: The horizontal lines for these parameters indicate that the benchmark does not have enough independent computational operations to utilize additional resources. This further supports the conclusion that the benchmark has a low to moderate ILP degree, as the dependency between instructions limits parallel execution.

Conclusion: The results indicate that the **ILP degree of the benchmark is relatively low to moderate**.

2.2 Different programs with different performances

For this part, you use the pc.ss benchmark (i.e., pointer chaser) from /simplescalar/cde-root subdirectory. Repeat the same procedure using the same strategy as in part 1 and plot the charts again. Then answer the following questions.

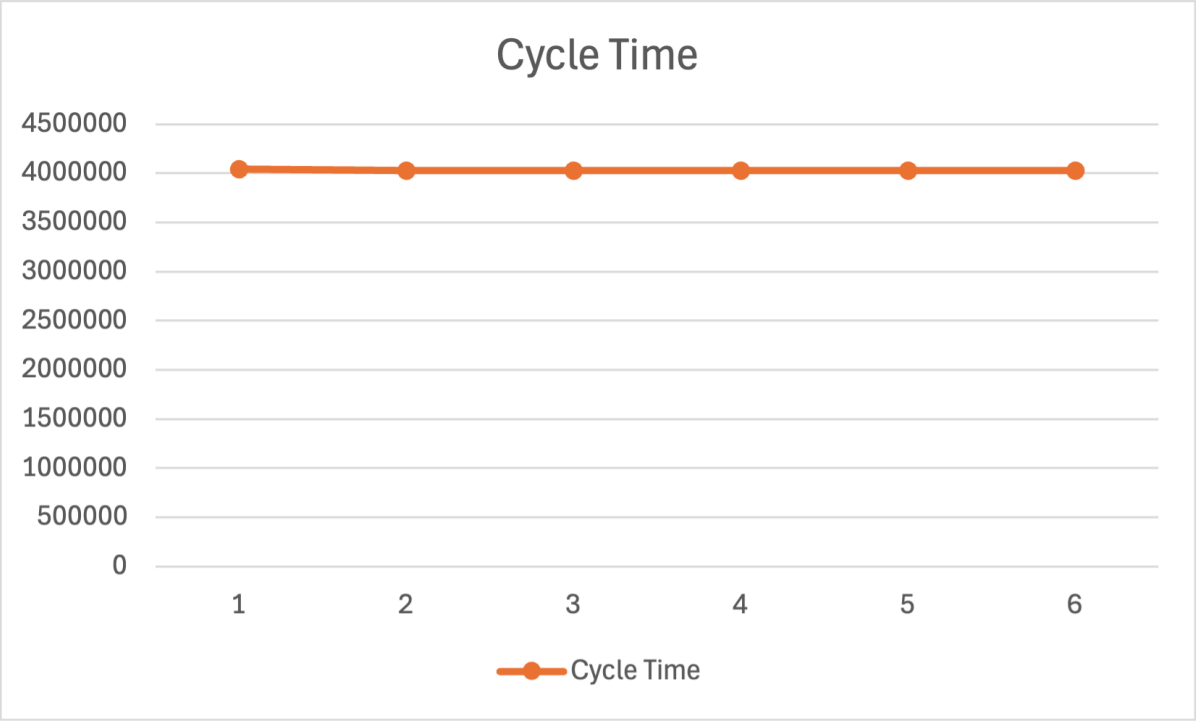

Diagram representing the cycle-time when changing values for decode:width & commit:width

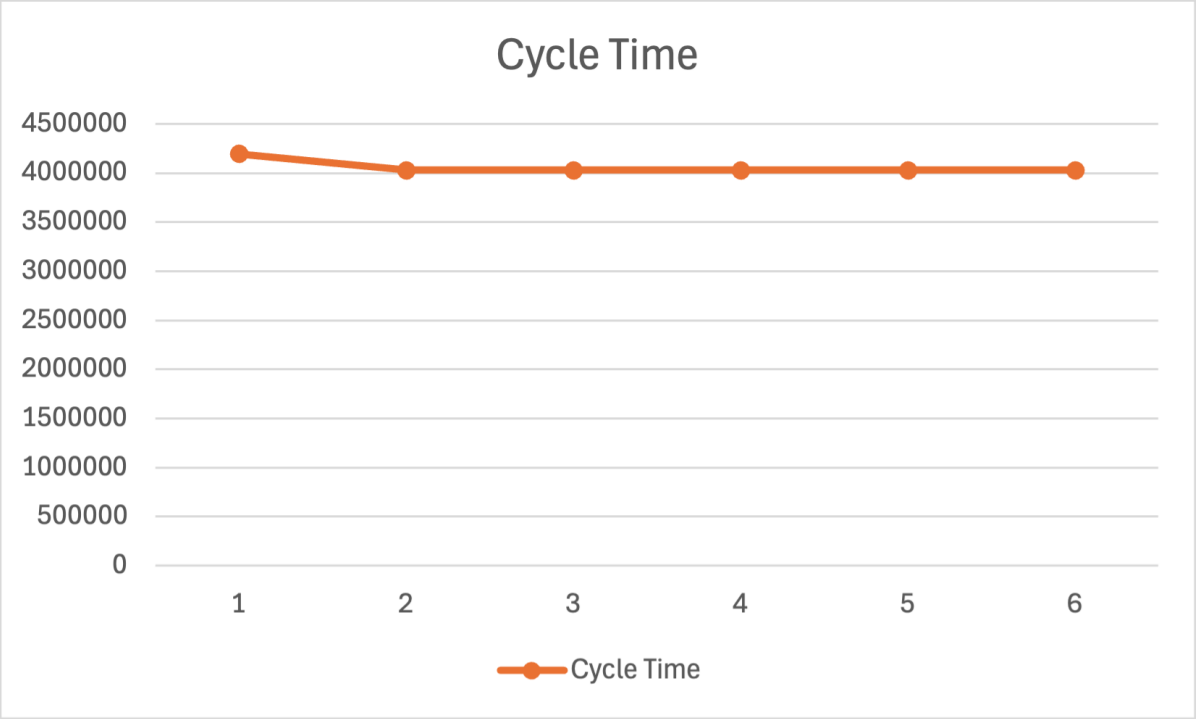

Diagram representing the cycle-time when changing values for issue:width

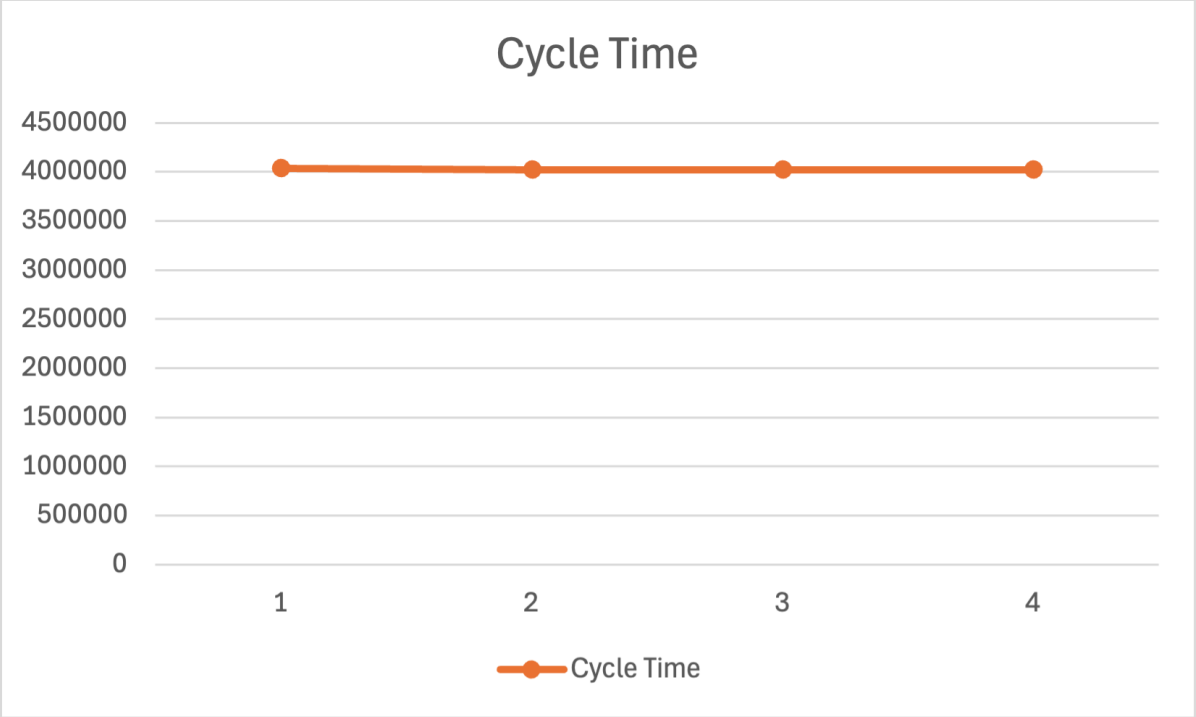

Diagram representing the cycle-time when changing values for res:ialu, res:imult, res:fpalu & res:fpmult

1. Compare the parameter impacts on performance in this benchmark with the previous one. Explain the difference with respect to ILP degree.

Observations:

- **Horizontal Lines for All Parameters:** In the `pc.ss` benchmark, every parameter (`-decode:width`, `-issue:width`, `-commit:width`, `-res:ialu`, `-res:imult`, etc.) has a horizontal performance curve, indicating no performance improvement regardless of parameter changes.
- **High Sim-Cycle Count:** The simulation cycles remain fixed at 4,000,000, indicating that the execution time is significantly higher than the `go.ss` benchmark (which started at around 2,000,000 and improved to ∼950,000).

Explanation with Respect to ILP:

- **Low or Near-Zero ILP in `pc.ss` Benchmark:**
  - The `pc.ss` benchmark is a pointer-chasing workload, characterized by a high degree of data dependencies. Each pointer dereference depends on the result of the previous one, forming a sequential chain of operations.
  - Due to these dependencies, there is almost no opportunity for instruction-level parallelism (ILP), as the processor cannot issue or execute multiple instructions in parallel.

  Answer: Based on the performance metrics in each graph we choose the input value of the parameter where the graph starts to stabilize. This give us the following cost:

  Cost = (1 + 2 + 1 + 1 + 1 + 1 + 1) = 8

2. Which configuration does attain the best performance/cost trade-off?

   Conclusion: The second configuration is the most optimal based on the performance/cost trade-off. The first configuration gives us a cycle time of approximately 1,000,000 with a cost of 19 units. The second configuration has a cycle-time of approximately 4,000,000 with a cost of 7 units. Based on our experiment the time it took to run both programs was almost identical, but the cost of the second one is less than half the price. Therefore, for our project it is more optimal to choose the second configuration.