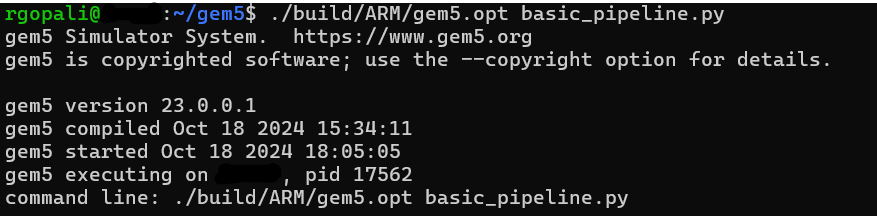
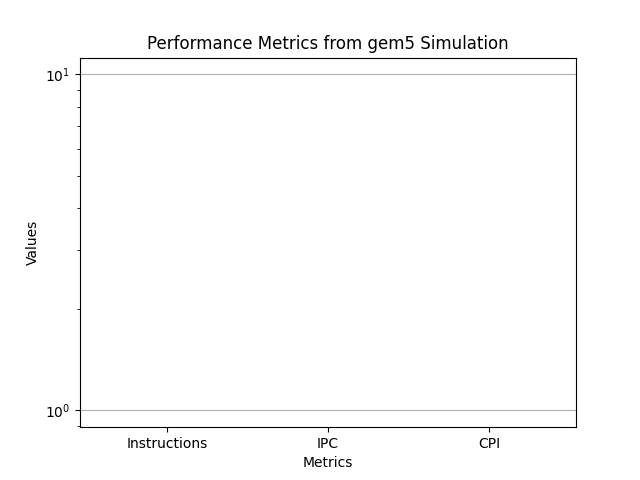
**Part 2: Practical Exploration of ILP Techniques**

In this proposed experiment, a simple ‘hello\_world.c’ program has been written where the binary outcome of the file has been incorporated with the ‘basic\_pipeline.py’. This ‘basic\_pipeline.py’ consists of configuration including performance matrices, branch prediction, super scaler configuration and multithreading. The gem5 simulation output is provided below.



The graphical representation of the configuration stats is provided below.



Considering the experiment, Instruction-Level Parallelism (ILP) interacts with the other’s effects in a complex way where it either amplifies the performance or else there is no significant performance has been found. In our case, no significant performance amplification is found. From the experiment, pipelining and superscalar execution are interconnected. If the pipelining is not performing in a balanced manner, then stalls could be there which reduces the superscalar performance. On the other hand, branch prediction helps enhance the efficiency of the pipeline and if there is any failure in prediction, it can impact on the performance severely with pipeline flush.

The major limitations of these techniques are related to data and control hazards. There are limitations on the complexity of the CPU architectural designs where more complexity increases power consumption. Thus, it is essential to implement the ILP technique based on the workload. Providing a simpler design not only provides high performance but also consumes less power which increases the efficiency of ILP. The usage of feedback mechanisms and focusing on the right granularity of parallelism balances the benefits of complexity and performance of ILP.