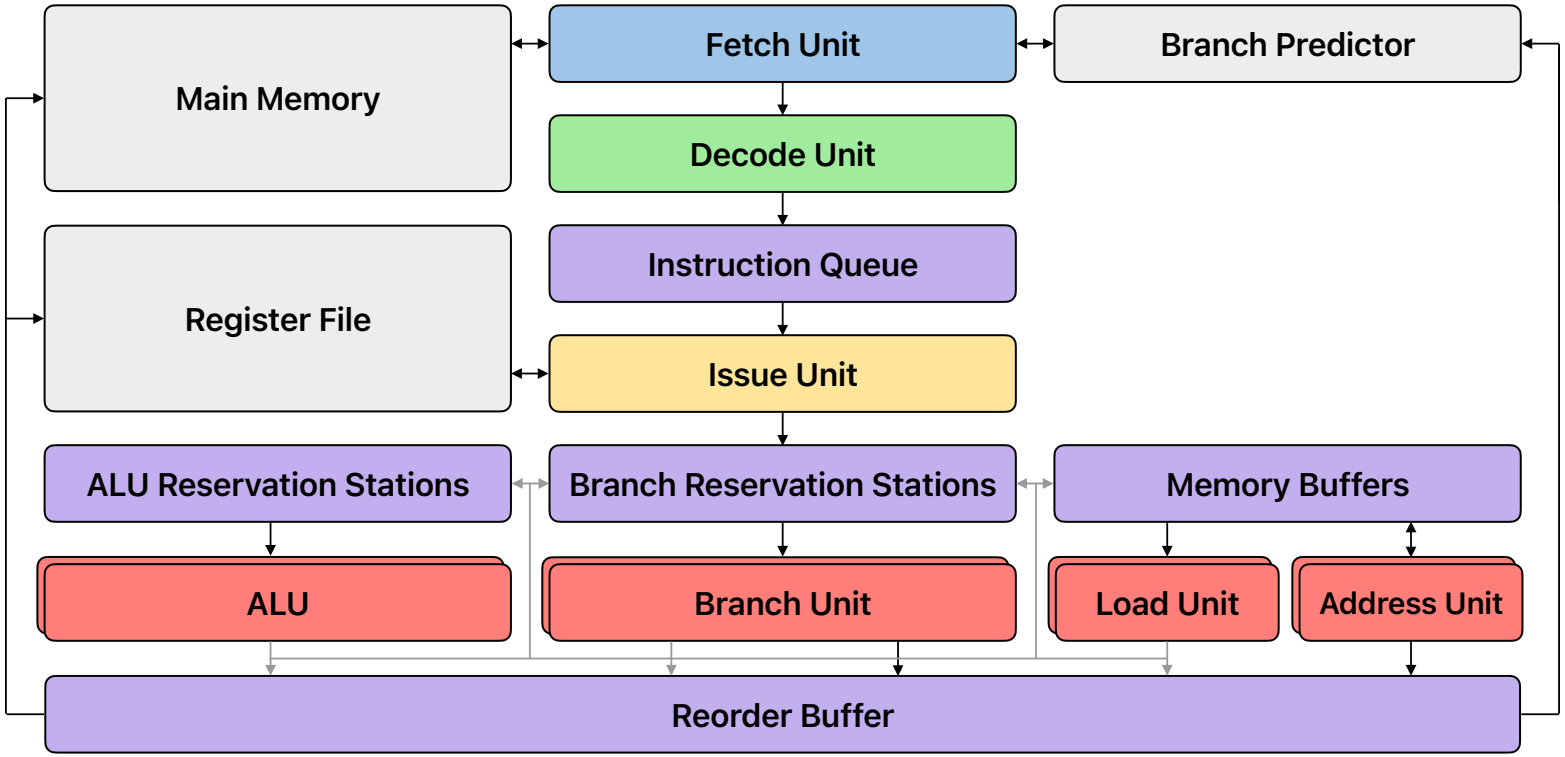
Processor Simulator

George Herbert

3rd May 2023

High-Level Architecture Diagram



Main Features

- *n*-way superscalar
- Speculative execution with a reorder buffer
- Dynamic branch prediction
- Out-of-order execution with Tomasulo's algorithm
- RV32I base integer instruction set

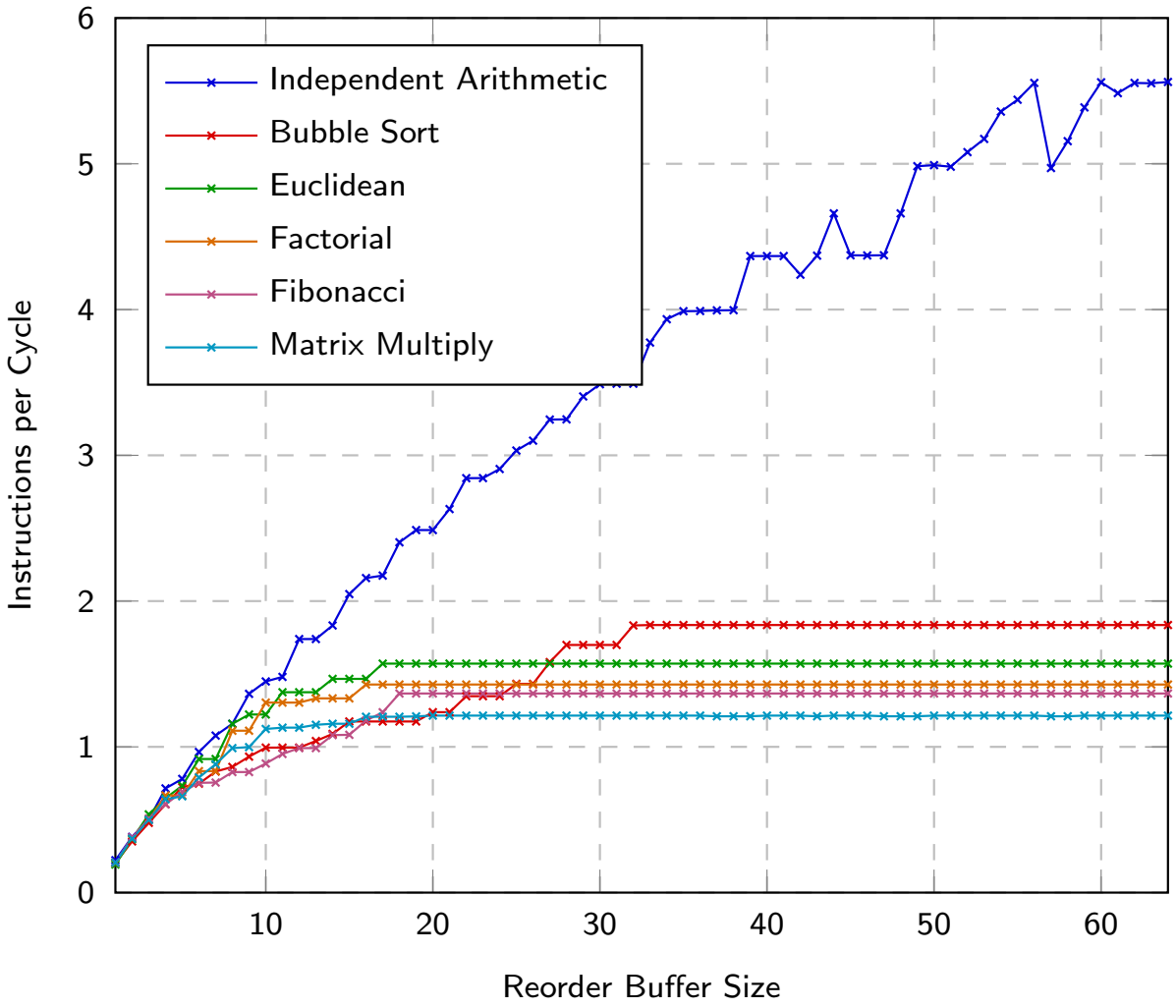
Configurable Components

- Fetch/Decode/Issue width
- Commit width
- Number of ALUs, branch units, address units and load units
- Number of reservation stations and memory buffers
- Reorder buffer size
- Instruction queue size
- Branch target buffer size
- Execution cycles for each instruction

Reorder Buffer Size

Hypothesis: Increasing the reorder buffer size will enhance performance to a certain extent. However, the performance will plateau after reaching a certain point due to dependencies between instructions or bottlenecking elsewhere.

Reorder Buffer Size



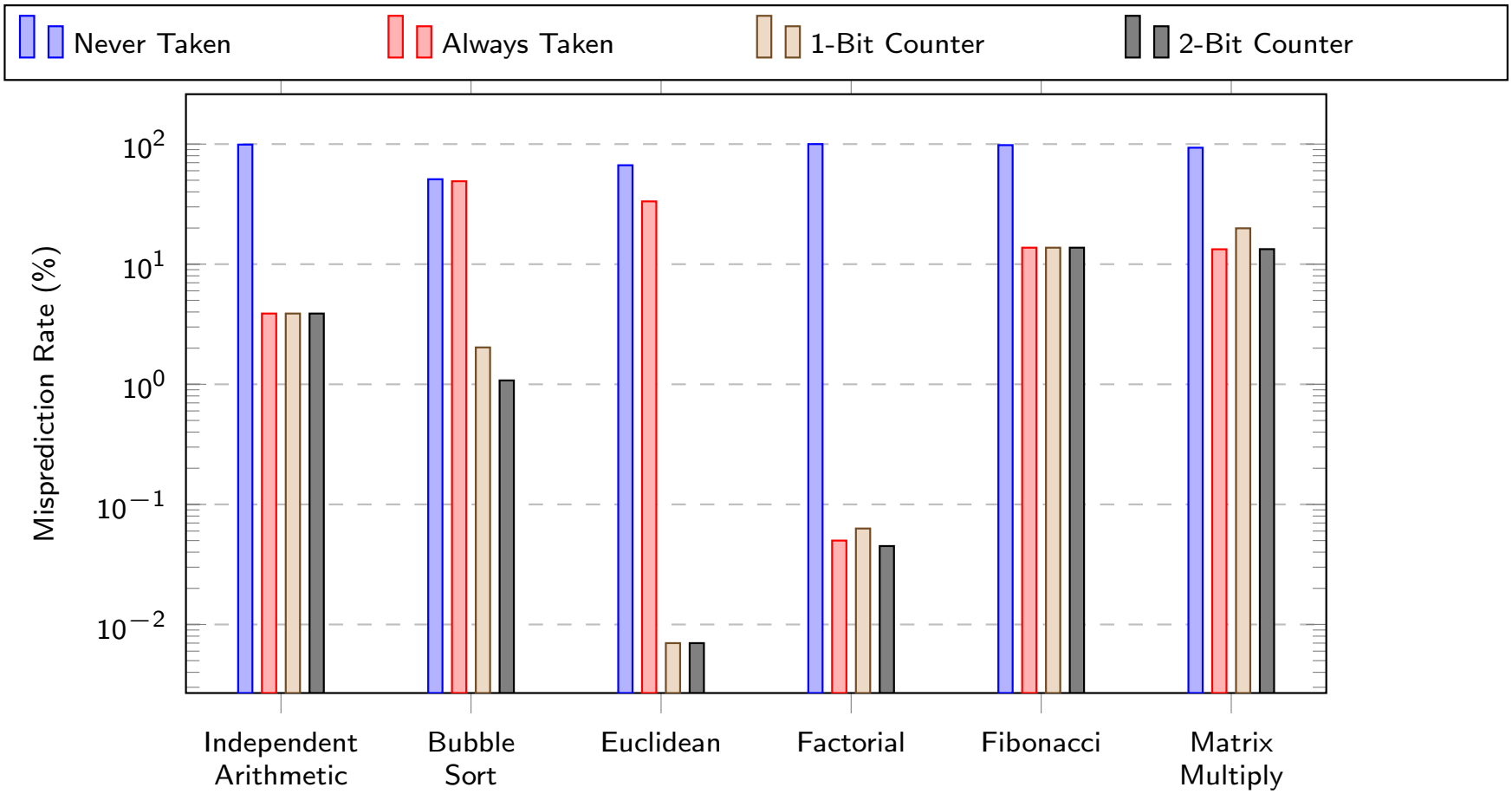
Reorder Buffer Size

Outcome: We broadly observe an increase in performance for all benchmarks as the reorder buffer size increases. The increase is most significant in the case of the arbitrary arithmetic benchmark since it has very few dependencies between instructions. When the reorder buffer size is one, the processor behaves as a single-issue in-order processor.

Branch Prediction

Hypothesis: Dynamic branch prediction will mispredict fewer branches than static branch prediction. Out of the dynamic branch predictors, a 2-bit saturating counter will mispredict fewer branches than a 1-bit saturating counter.

Branch Prediction



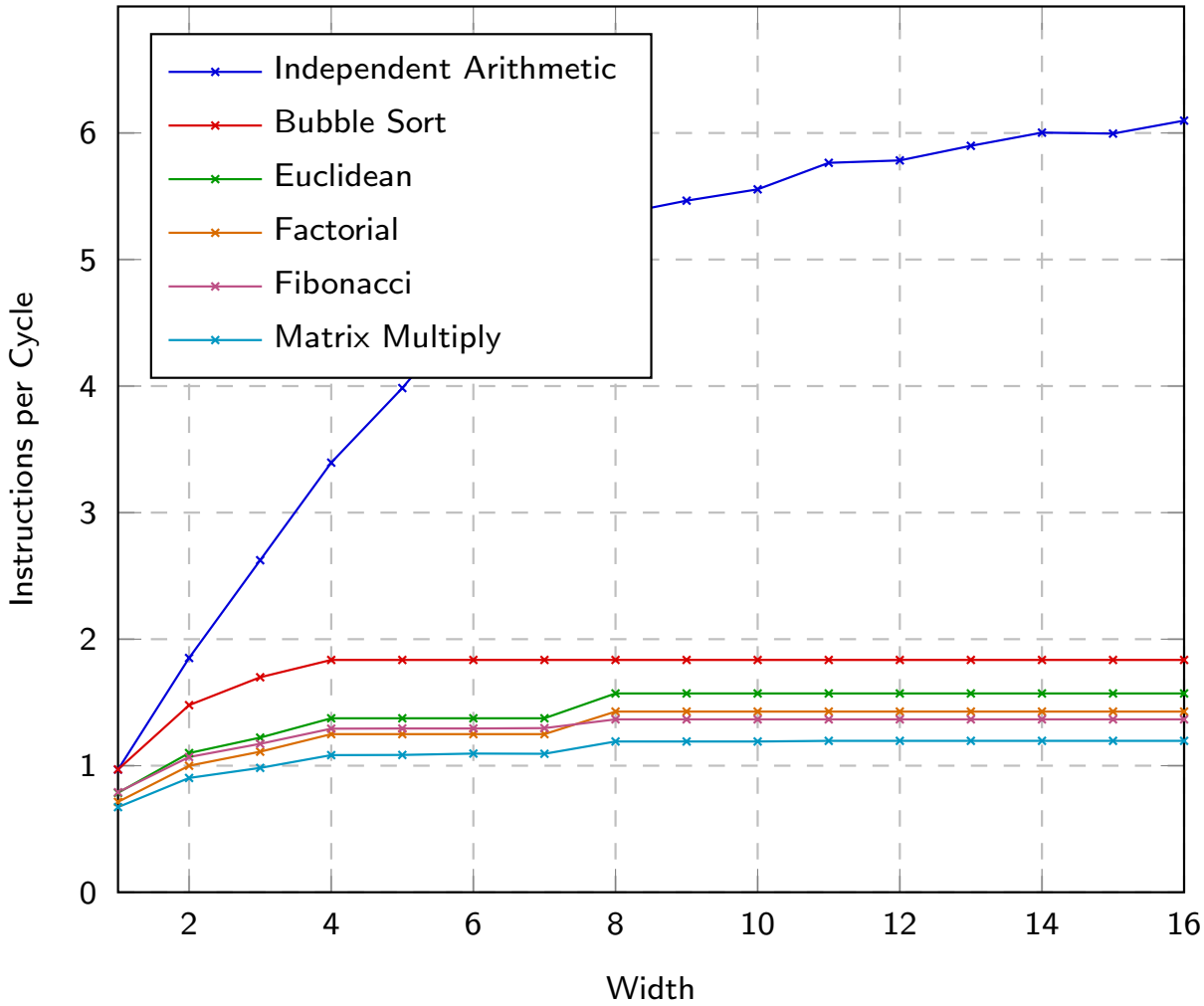
Branch Prediction

Outcome: We observe the 2-bit saturating counter to outperform the 1-bit saturating counter in all benchmarks. We believe this is because the 1-bit saturating counter causes most conditional branches to mispredict twice (e.g. for loop-closing branches, we mispredict once when we do not take it, then again the next time we take it). Interestingly, in some benchmarks, the always-taken predictor performs equally as well as the 2-bit saturating counter. In the case of the independent arithmetic benchmark, this is because there is only a single loop in the benchmark, which we iterate over a fixed number of times.

Fetch/Decode/Issue/Commit Width

Hypothesis: Increasing the fetch/decode/issue/commit width will enhance performance to a certain extent. However, the performance will plateau after reaching a certain point due to dependencies between instructions, the presence of branches, or bottlenecking elsewhere.

Fetch/Decode/Issue/Commit Width



Fetch/Decode/Issue/Commit Width

Outcome: We observe a consistent increase in performance for all benchmarks as the width increases until its plateaus. The increase is most significant in the case of the arbitrary arithmetic benchmark since it has very few dependencies between instructions and very few branches. The width of the pipeline serves as an upper bound on the IPC because we cannot complete more instructions in any given cycle than the width of the pipeline.