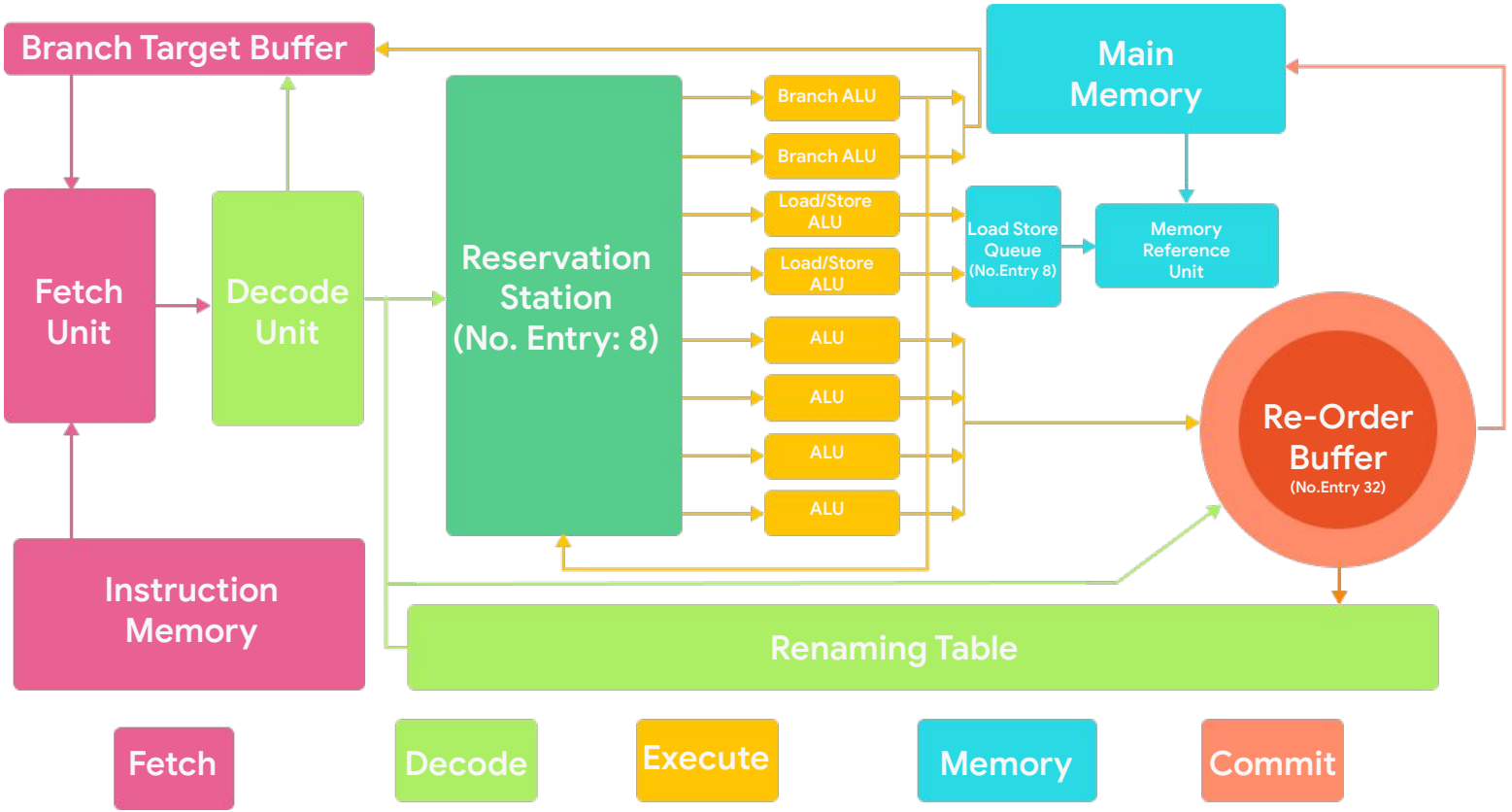
Superscalar Processor Simulator

Conor Mullaney

Implemented System Architecture



Simulator Features

- 5 Stage Pipeline (Fetch, Decode, Execute, Memory Access, Write Back)
- N-way Superscalar
  - Configurable Number of Fetch Decode, Execute, Memory Access and Writeback units each ticking once per cycle. Note: In an 8-way processor we use 2 Load ALU, 2 Branch, 4 Arithmetic in all other superscalar implementations we use generalised ALUs.
- Out-of-Order Execution
  - Tomosulos algorithm
  - 32-entry reorder buffer (configurable)
  - Renaming table to prevent false dependencies
  - Result forwarding and speculative execution
- Reservation Station
  - Unified Reservation Station
  - Stalls when full
- Pipe Flushes
  - Rebuild renaming table, result forwarder and scoreboard from re-order buffer (and completed instructions).

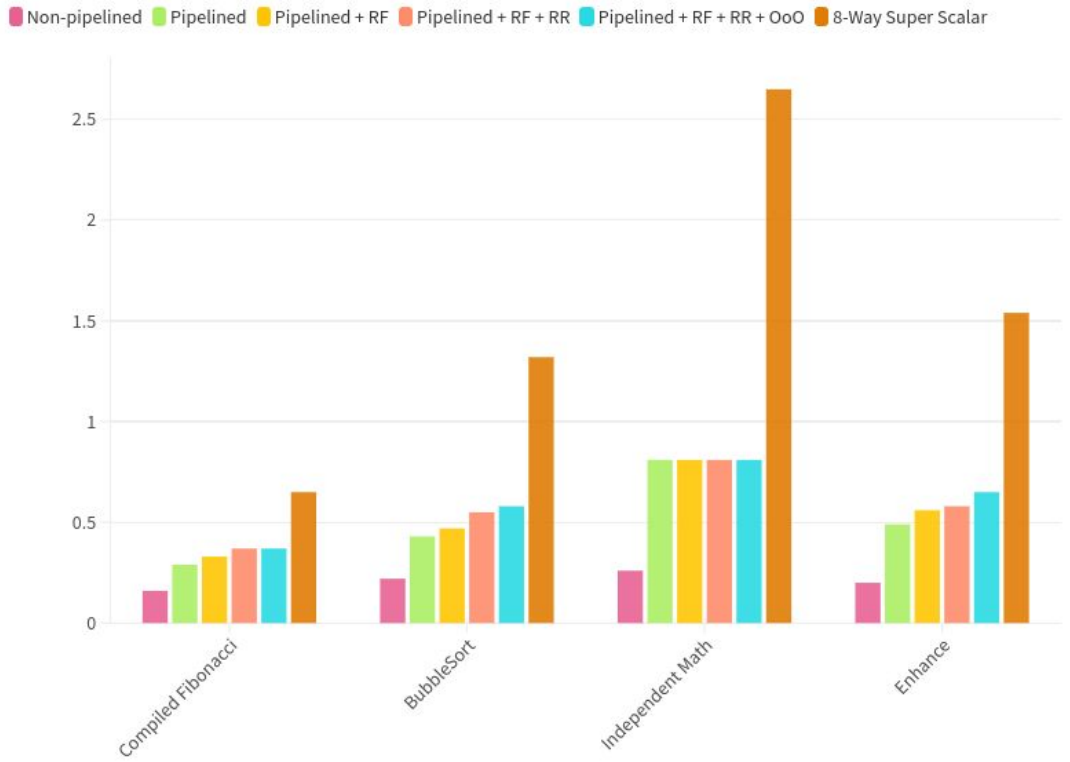
Simulator Features Cont.

- Branch Prediction
  - Static Forwards and backwards
  - Dynamic 1 Bit and 2 Bit
  - A branch target buffer is used for our predictors as we only know that an instruction is a branch after the first decode
- Registers and Memory
  - 32 general-purpose integer registers: $x0 to $x31
  - $x0 (also known as $zero) is hardwired to 0 and cannot be modified
  - $x1 (also known as $ra) is the return address register (used by jal and jalr instructions though didn't end up being used in our benchmarks)
  - $x2 (also known as $sp) is the stack pointer (used by the stack-related instructions e.g. loads and stores)
  - $x3 to $x8 are the argument registers for function calls
  - $x9 to $x31 are temporary registers which may be used freely.
- Runs simple C programs compiled with the Risc-V toolchain.

Benchmarks

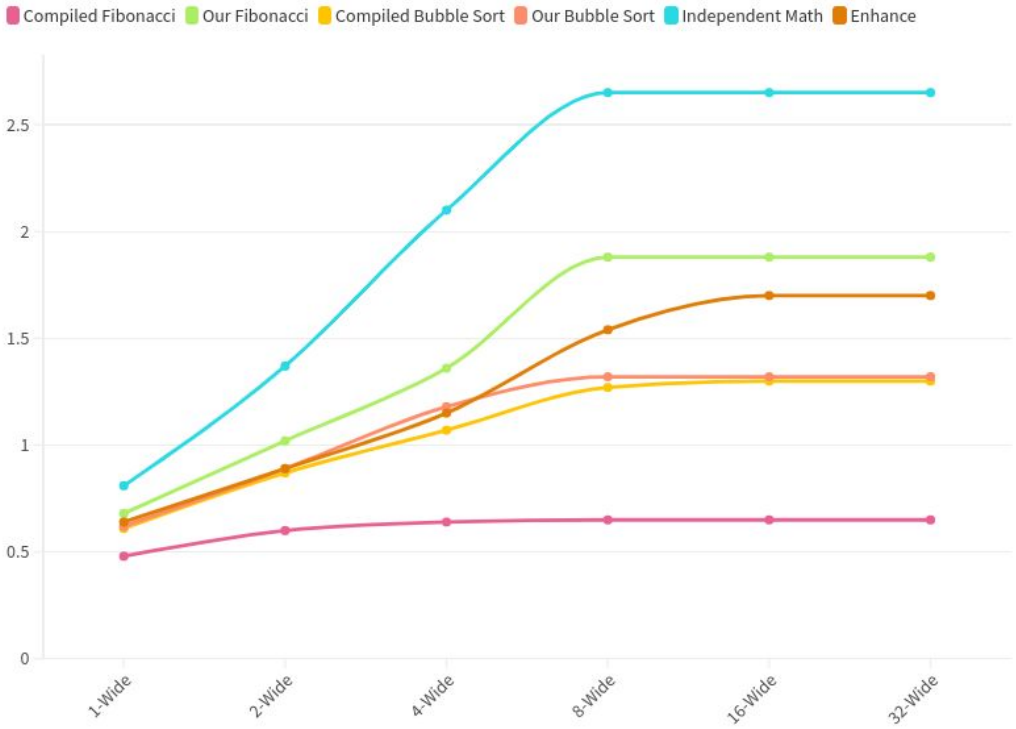
Benchmark	Description
Fibonacci	Computes the Fibonacci sequence. Using both our own and compiled code implementation.
Bubble Sort	Implements the bubble sorting algorithm. Using both our own and the compiled code implementation
Independent Math	Contrived additions to maximise Instructions per Cycle in our processor
Branching	Performs a repeated artificial sequence of branches, to stress-test prediction methods
Enhance.s	Implements thresholding on an MNist Image, from compiled code.

Experiment 1 - Processor Evolution



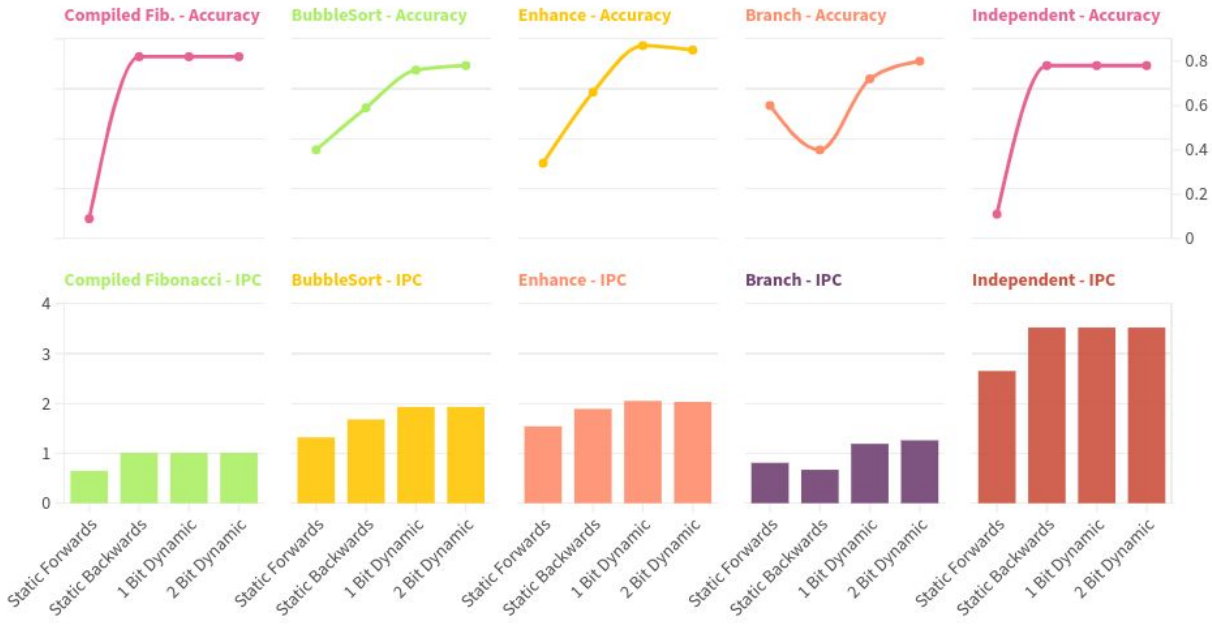
- Hypothesis: As the processor architecture becomes more advanced, with more features such as Pipelining, Result Forwarding (RF), Register Renaming (RR), Out-of-Order execution (OoO), and Superscalar execution, each advancement will significantly improve processor performance.
- Experiment: Measuring our performance in Instructions per Cycle (IPC), we tested each of our benchmarks on a processor with increasingly advanced features. (Demonstrated with a static not taken branch predictor).
- Result: Depending on the Program characteristics, each advancement in our processor provides different benefits. E.g. Fibonacci does not benefit from OoO in a single width pipeline as it is a very dependent program. The independent math benchmark with very little dependencies did not benefit from result forwarding.

Experiment 2 - Reservation Size



- Hypothesis - Due to the potential bottleneck caused by a unified reservation station in our processor, we anticipate that increasing the size of the reservation station will lead to an improvement in the processor's performance.
- Experiment - Using our fully advanced processor from Experiment-1, we calculate the IPC of our processor simulator for each of our benchmarks.
- Results - The majority of programs did not show a significant performance increase beyond a reservation station size of 8. However, our enhanced and compiled bubble sort programs demonstrated a slight improvement when the reservation station size was increased from 8 to 16. This may be due to the limited number of Load/store ALUs, causing the reservation station size to have a more significant effect on programs which use load/store instructions more frequently.

Experiment 3 - Branch predictors



Compiled Fib Correlation	Bubble Correlation	Enhance Correlation	Branch Correlation	Independent Correlation
1.0	1.0	1.0	0.97	1.0

- Hypothesis - Instructions per Cycle (IPC) is directly correlated by the accuracy of our branch predictors, and an increase in predictor accuracy will lead to an increase in IPC.
- Experiment - using our benchmarks on an 8-way SS OoO processor we took metrics for IPC and prediction accuracy to calculate their performance. We then calculated their pearson's correlation coefficient
- Result - Branch prediction success rate and performance are directly correlated with an average correlation of 0.99. The Branch benchmark had the worst performing correlation, this is likely because there are also other significant factors such as increased branch instructions causing bottlenecks for the benchmarks.