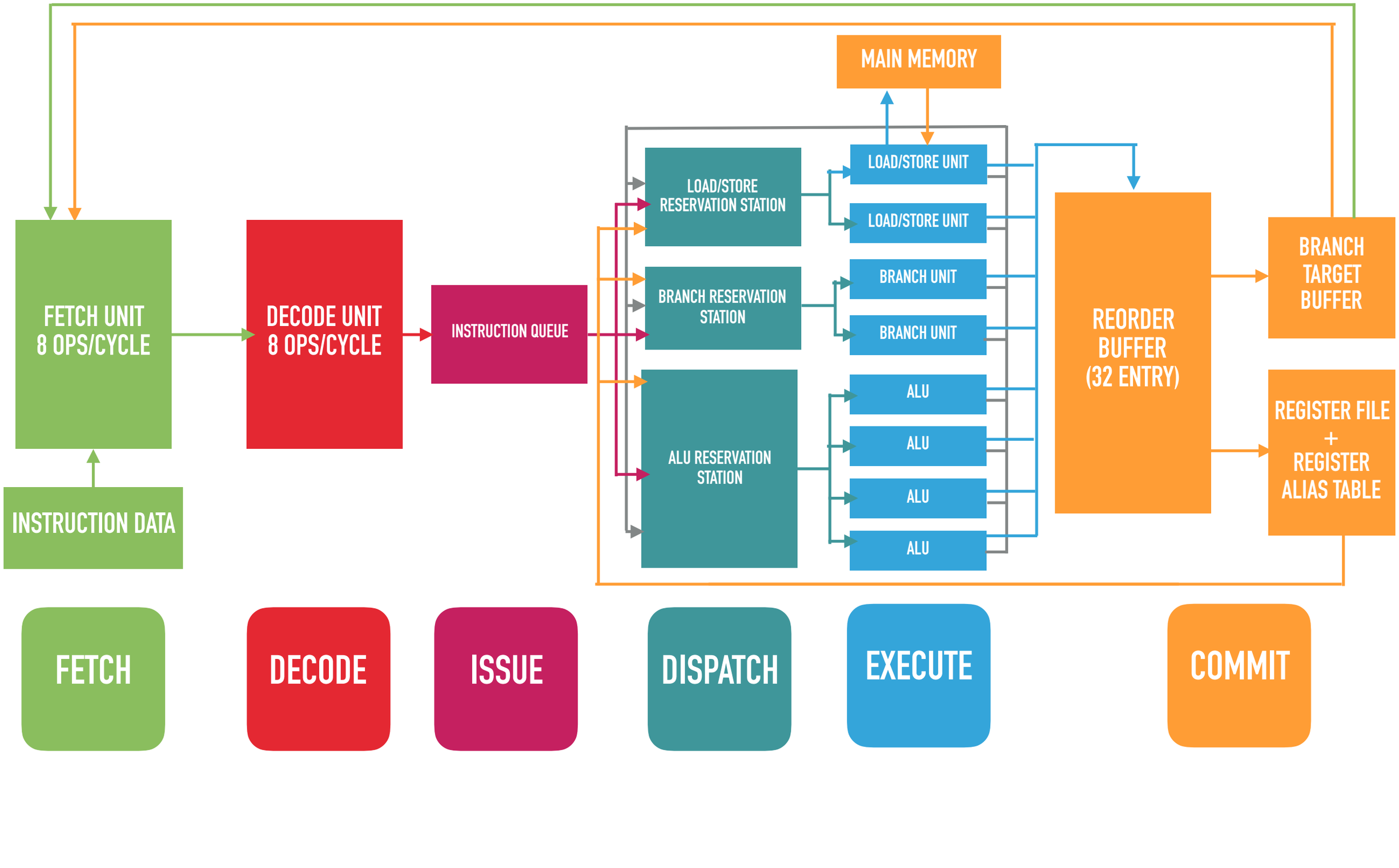
SUPERSCALAR PROCESSOR SIMULATOR

HARRY WAUGH

DIAGRAM OF CHOSEN ARCHITECTURE



SIMULATOR FEATURES

- **3 Modes:** Simulator can be configured to run in Serial, Pipelined and Superscalar modes
- **Fully Pipelined** with 6 stages of execution (Fetch, Decode, Issue, Dispatch, Execute, Commit)
- **N-way Superscalar**
  - Configurable number of Branch, Load/Store and ALU units.
  - Configurable number of Fetch/Decode/Issue/Commit instructions per cycle
- **Out-of-Order Execution**
  - Implements Tomosulo's Algorithm
  - Register Renaming with 32-entry Reorder Buffer. Prevents WAR and WAW dependencies.
  - Execution unit results are forwarded to reservation stations for use in next cycle.

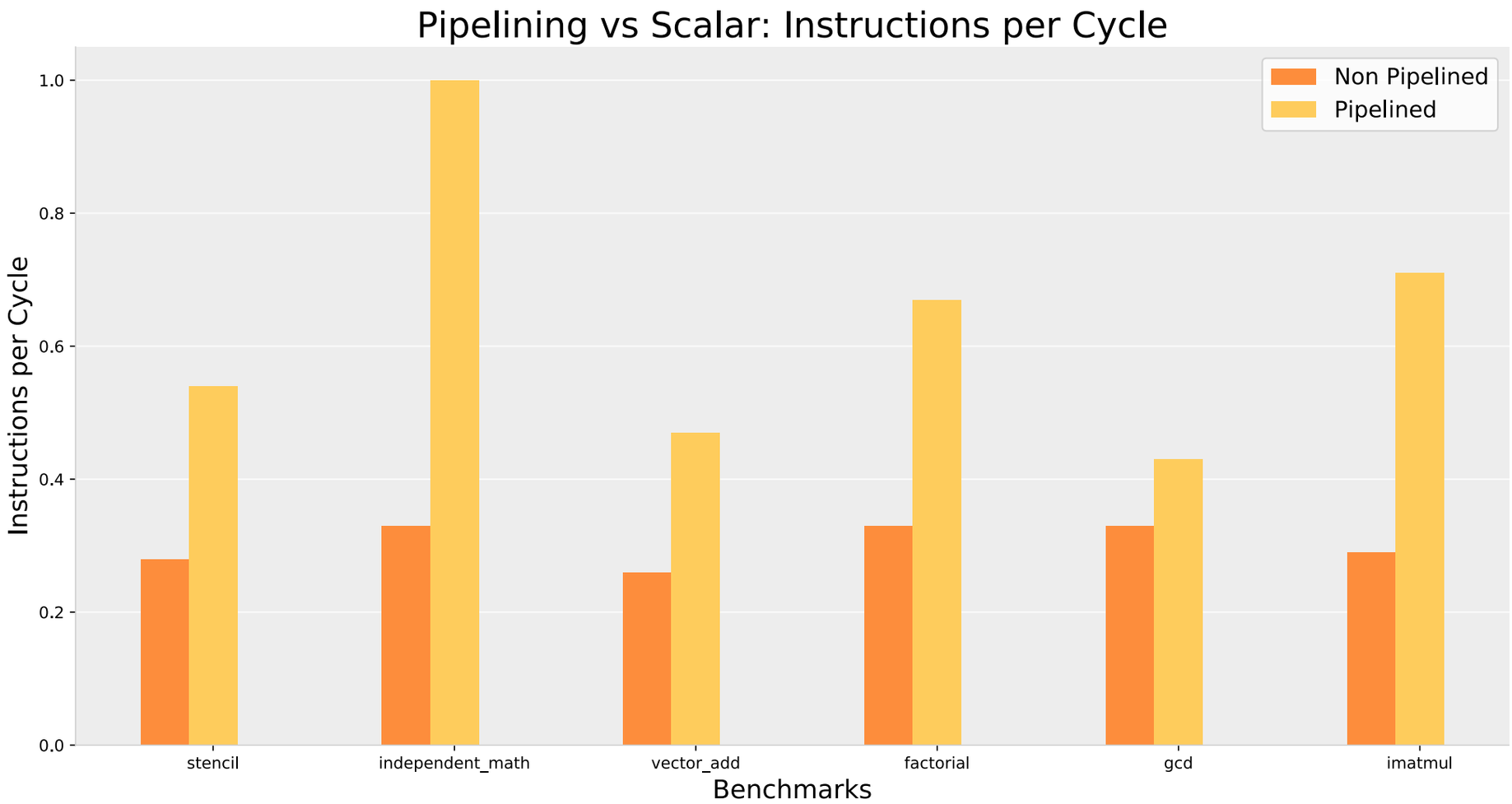
SIMULATOR FEATURES (CONT.)

- **Reservation Stations**
  - Non-blocking issue (stalls if no space in reservation stations or reorder buffer)
  - Separate reservation stations for ALU, Load/Store and Branch units
- **Branch Prediction (Configurable)**
  - **Static:** Always taken, Always not taken
  - **Dynamic:** 2-Bit Saturating Counter, 2-Bit Saturating Counter with Branch Target Buffer
  - False predictions refresh the pipeline in commit, fetches correct instructions in the following cycle.
- **Registers and Memory**
  - 64 Registers (2 System, 30 General Purpose, 32 Floating Point)
  - 4 KB Main Memory (Configurable)

BENCHMARKS

Benchmark	Description
Stencil	Computes 10 iterations of a 5 point stencil across a chess board pattern. Branches are used to detect edges.
GCD	Computes the greatest common divisor of two integers.
Independent Math	A loop calculating double the sum of n natural numbers.
Branching	Two programs that each perform a sequence of branches designed to highlight the difference between the branch predictors implemented.
IMATMUL	Computes A*B where A and B are both square integer matrices.
Vector Add	Computes the sum of two vectors.

SUPERSCALAR PROCESSOR SIMULATOR



HYPOTHESIS

A 3-stage scalar simulator's performance can be increased by up to 3x by pipelining it.

EXPERIMENT

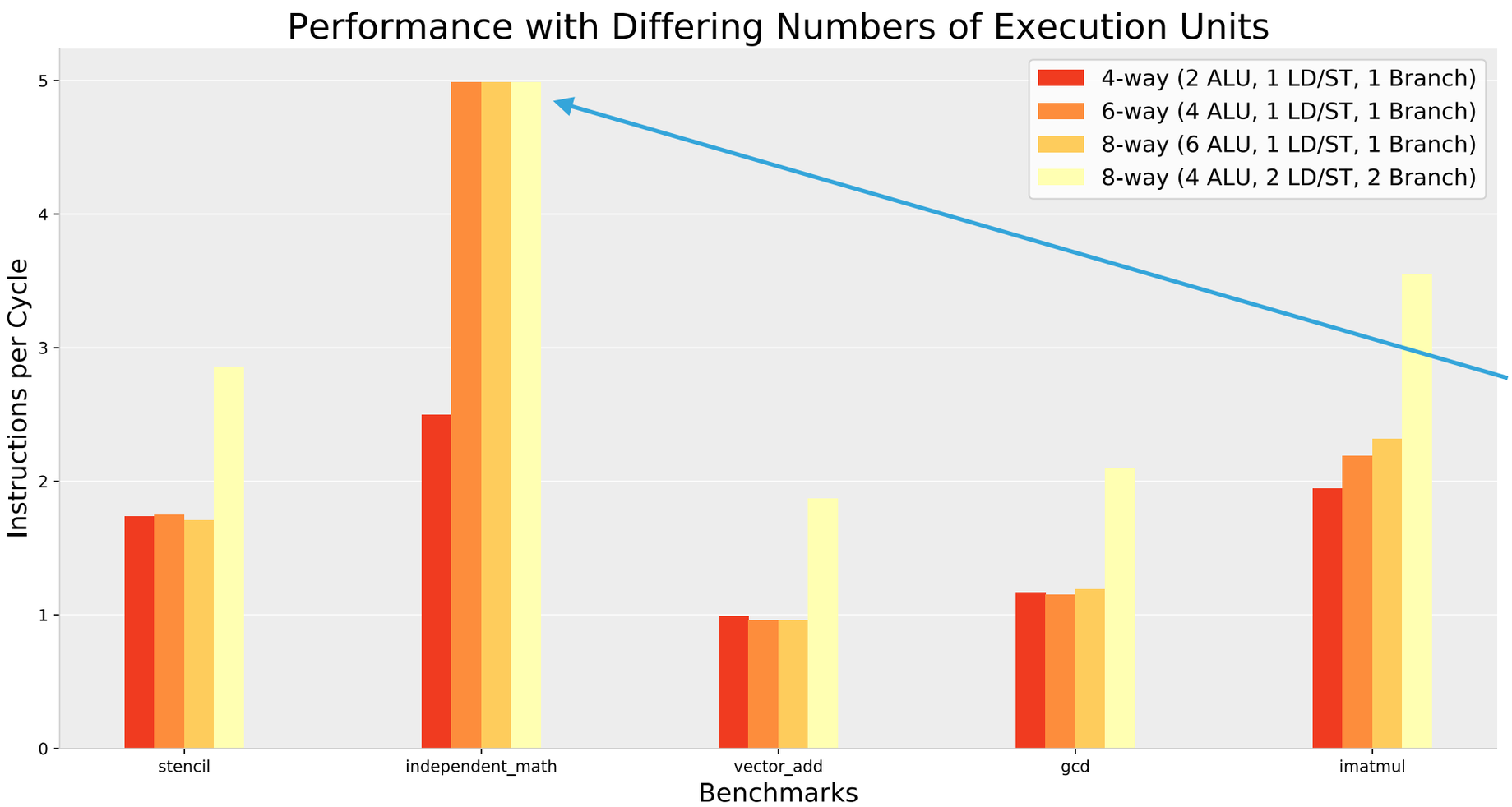
The number of instructions per cycle was measured for different benchmarks on both a scalar and pipelined 3-stage processor simulator.

RESULT

The pipelined processor simulator achieved better performance on all benchmarks. However, only the 'independent_math' benchmark achieved the maximum performance increase. There is two reasons for this:

- It contains no multi-cycle operations (load, store, etc).
- This pipelined version always predicts branches as taken, which happens to be correct for all branches in this benchmark.

SUPERSCALAR PROCESSOR SIMULATOR



Main Kernel for Independent Math

1.	add	$t1,	$t1,	$a1
2.	add	$t0,	$t0,	$a0
3.	addi	$a0,	$a0,	1
4.	addi	$a1,	$a1,	1
5.	blt	$a1,	$a2,	-4

HYPOTHESIS

Increasing the number of ALU units will increase performance until the performance is either limited by the inherent dependencies in the code or by other processor bottlenecks.

EXPERIMENT

Processor performance on several benchmarks was compared across 4 different execution unit configurations.

Processor performance was measured by calculating the number of executed instructions per cycle.

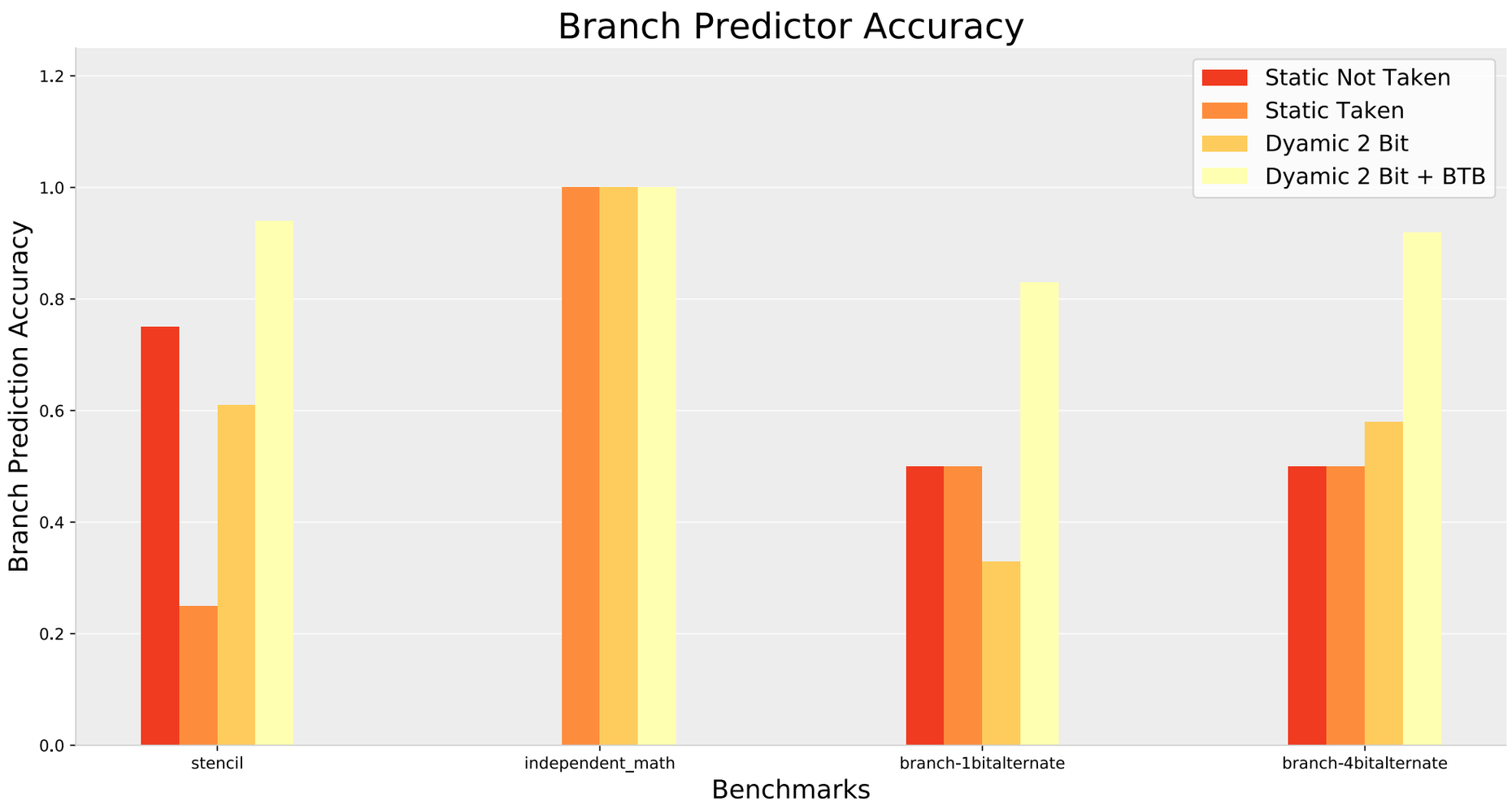
RESULT

It was found that increasing the number of ALU execution units did not improve performance for all benchmarks.

The integer matrix multiplication benchmark can be see to achieve better performance as more ALU units are added, however more significant performance was achieved by adding extra branch and memory units. This result was mirrored across all benchmarks where there was a high ratio of memory to arithmetic operations.

The independent math benchmark has no memory operations in its main kernel, and hence shows performance doubling when increasing from 2 to 4 ALU execution units. This is optimal for this benchmark, as instruction dependencies occur between loop iterations.

SUPERSCALAR PROCESSOR SIMULATOR



HYPOTHESIS

Using a Branch Target Buffer with a 2-bit saturating counter will **improve or not change** branch predictor accuracy when compared to static or global 2-bit saturating predictors.

EXPERIMENT

Branch prediction accuracy was measured for 4 branch predictors across benchmarks with differing branch characteristics.

RESULT

The 2-bit saturating counter with BTB achieves the highest branch prediction accuracy across each benchmark.

The accuracy is equal to the static taken and global 2-bit saturating counter predictors for the ‘independent math’ program as all but the final branch are taken.

For the stencil benchmark, the predictor achieves an accuracy of 94%. This is a much better predictor for stencil than the others as the BTB can make separate predictions for each of the edge checks within the main loop.