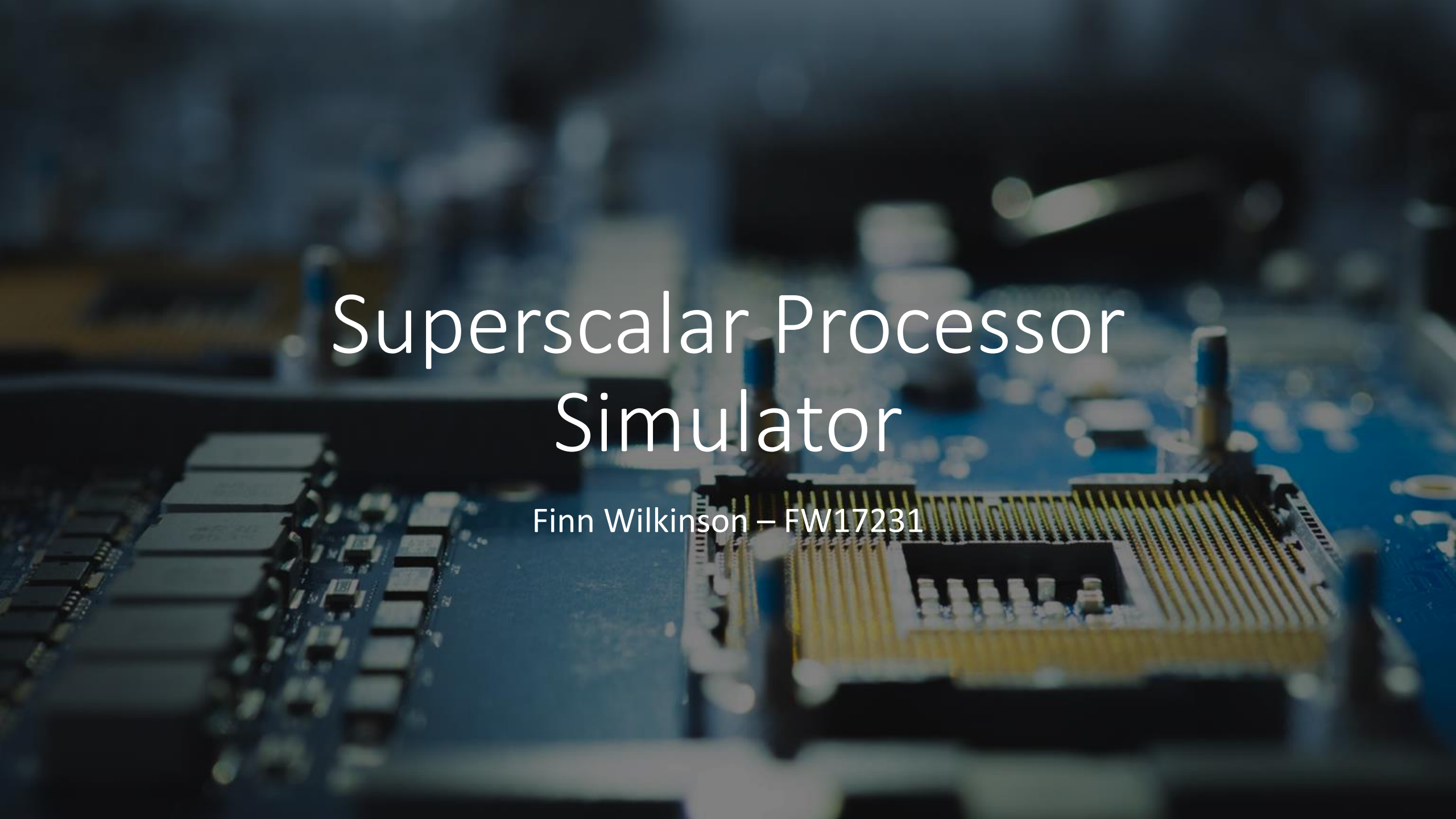
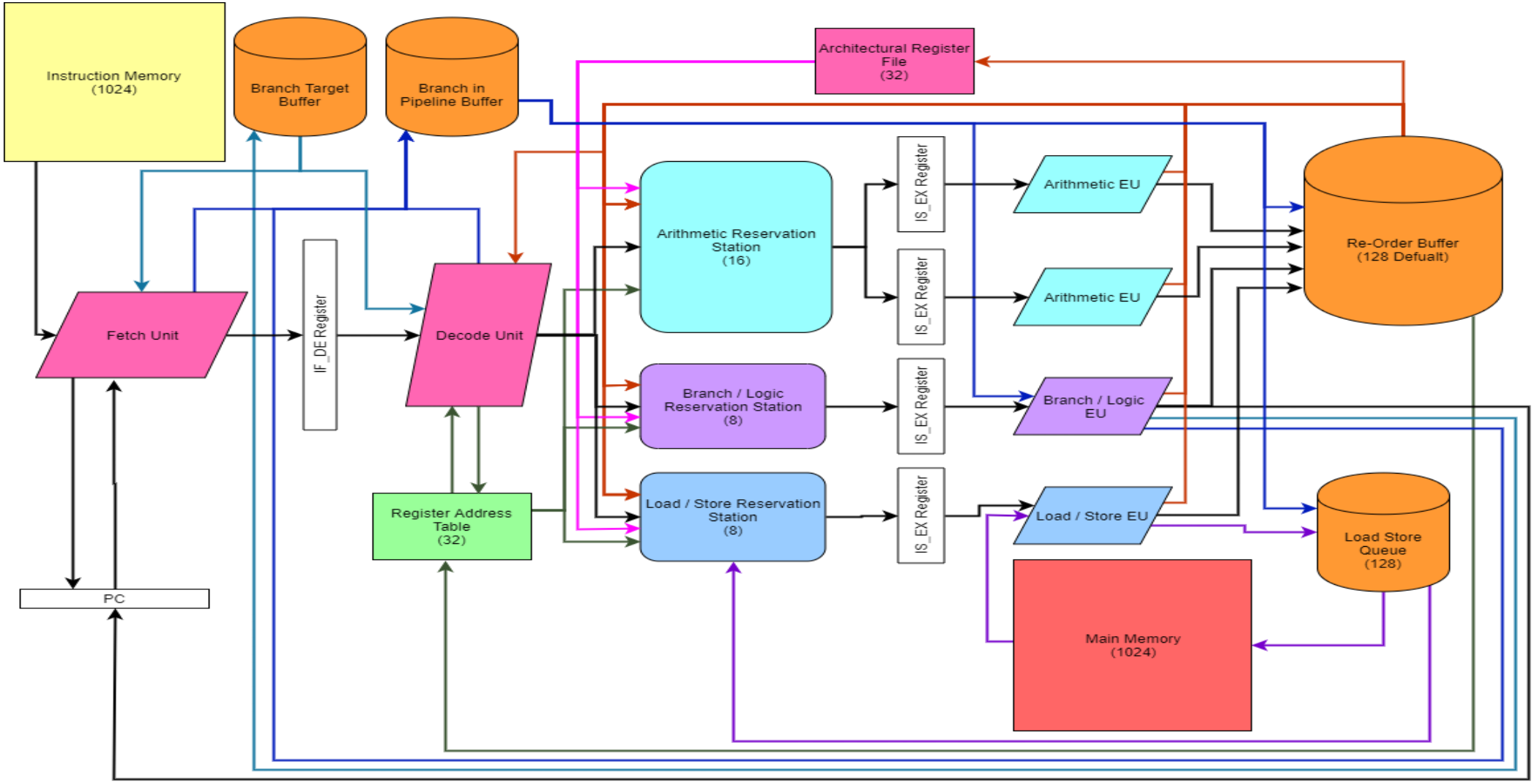


Superscalar Processor Simulator

Finn Wilkinson – FW17231

Architecture Design



Processor Features

General

- 32 General Purpose Registers
- 5 stage pipeline
  - Fetch, Decode, Issue, Execute, Write-Back
- Multi-Cycle instructions
  - Multiply - 2 cycles
  - Divide - 16 cycles
- Out-of-Order execution
- Register re-naming
  - Utilised Register Address Table
  - Prevents Name Dependencies, WAW & WAR hazards
- Scalar, 2-way superscalar, or 4-way superscalar

Memory and Write-Back

- Variable size Re-Order Buffer (default is 128)
  - Ensures program order write back to ARF, preventing RAW hazards
- Load-Store Queue of size 128
  - Ensures Loads and Stores happen in program order
  - Loads wait for all previous store addresses to be known before being issued to prevent RAW hazards

Pipeline Features

- 2 Arithmetic EUs
- 1 Branch and Logic EU
- 1 Load and Store EU
- Grouped Reservation Stations
  - 16 spaces for Arithmetic
  - 8 spaces for Branch and Logic
  - 8 spaces for Load and Store
- Result forwarding

Branch Prediction

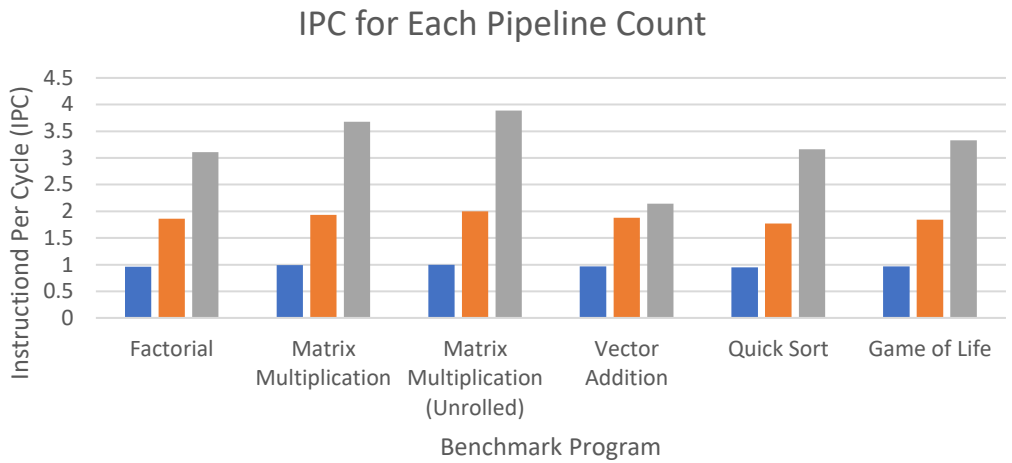
- Can select from
  - Fixed – Always predicts branch is taken
  - Static – If jumping backwards predict taken
  - 1-bit Dynamic – Make prediction on previous result
  - 2-bit Dynamic – Make prediction on previous 2 results
- BTB keeps history of all branches seen before
  - Also keeps track of previous results if dynamic prediction selected
- Branch in Pipeline Buffer
  - Tracks un-executed branches in the pipeline
  - Ensures nothing is committed to ARF or MEM out of program order
  - Allows for easy misprediction recovery and flushing

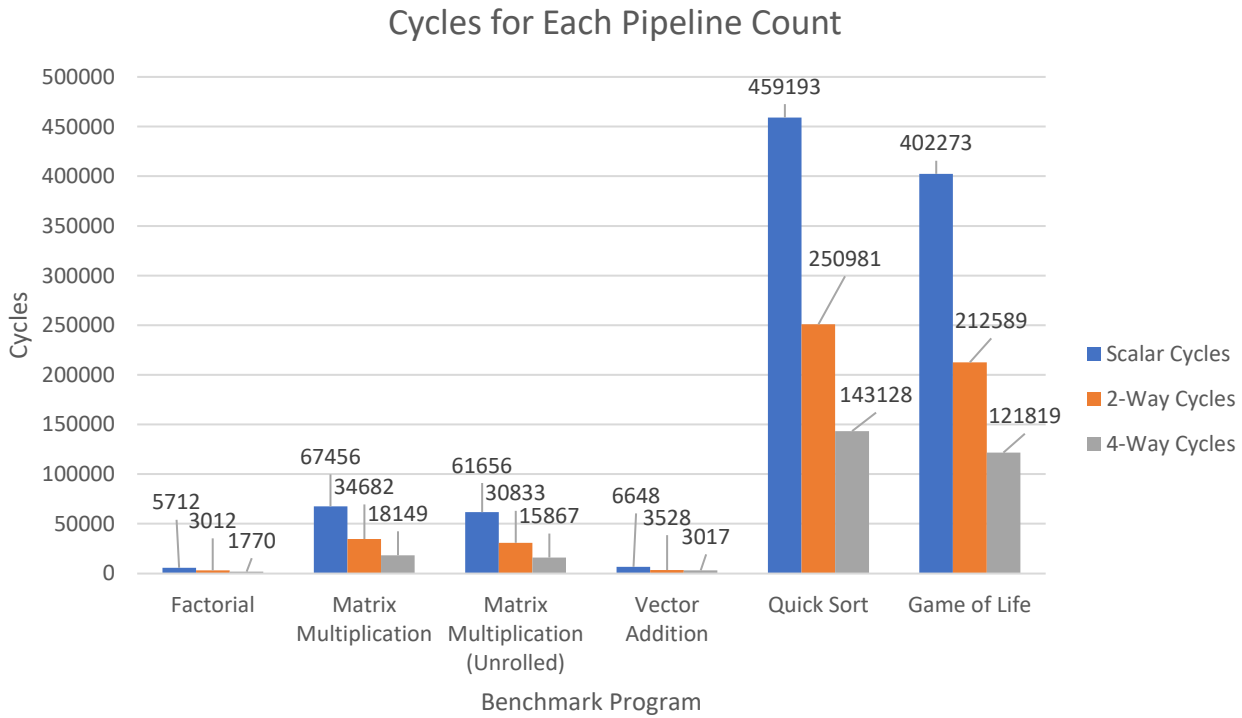
Benchmark Programs

All Scripts are looped in assembly 100 times to gain a more accurate IPC

Factorial	Recursively calculates the factorial of 12
Matrix Multiplication	Multiplies together a (6x4) matrix and a (4x6) matrix using branches to loop round
Unrolled Matrix Multiplication	Multiplies together a (6x4) matrix and a (4x6) matrix with an unrolled loop
Vector Addition	Adds together two vectors of length 10 A[i] = B[i] + c[i]
Quick Sort	Applies the quicksort algorithm to a list of length 30 utilising recursion and an in-memory stack
Game of Life	20 iterations of Conway's game of life on a 16x16 grid in-memory with wrap-around boarders

Experiment #1 – Number of Pipelines





Hypothesis

Doubling the number of pipelines will increase the IPC by x2, as well as reducing the number of cycles taken to execute each script by half.

Experiment

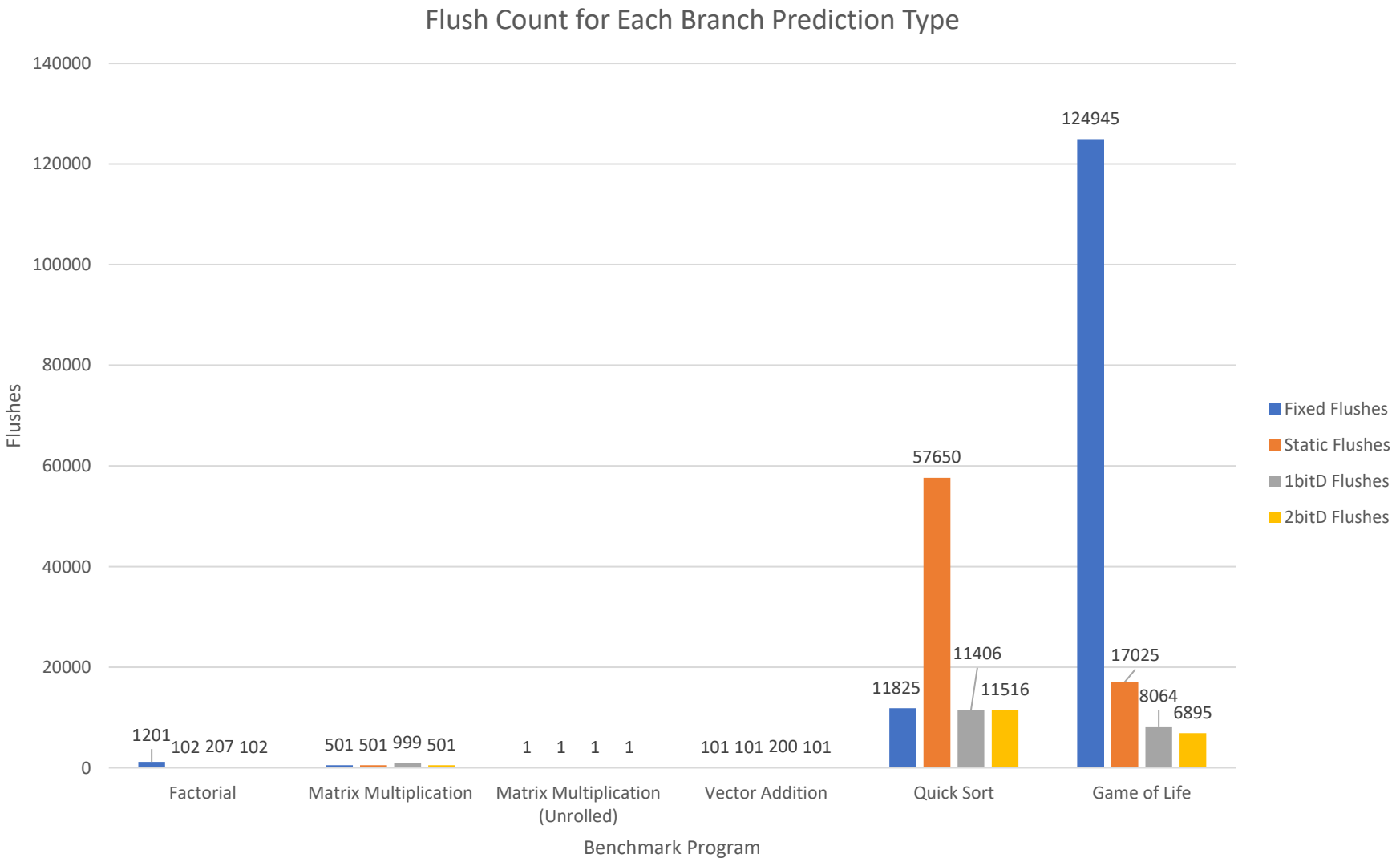
Comparing IPC values and total cycles for 1, 2, and 4 pipelines over all benchmark programs. 2-bit Dynamic Branch prediction will be used as well as a 512-bit ROB to achieve as best results as possible for each test.

Results

We can see that from 1 to 2 pipelines the IPC is roughly doubled for each of the benchmarks. However from 2 to 4 pipelines, this IPC gain slows down to approximately 55%.

Similarly with the cycle count, from 1 to 2 pipelines we see a halving in the number of cycles each benchmark takes. Differently from the IPC measure however, with the exception of Vector Addition, increasing from 2 to 4 pipelines also sees another halving in how many cycles each benchmark takes.

Experiment #2 – Comparing Branch Predictors



Hypothesis

For each of the benchmark programs, each successive branch prediction method will decrease the number of processor flushes required. With Fixed causing the most, then Static, 1-bit Dynamic, then 2-bit Dynamic with the fewest flushes.

Experiment

For a 4 pipeline processor with 512-bit ROB, test each branch prediction type and compare the number of flushes performed.

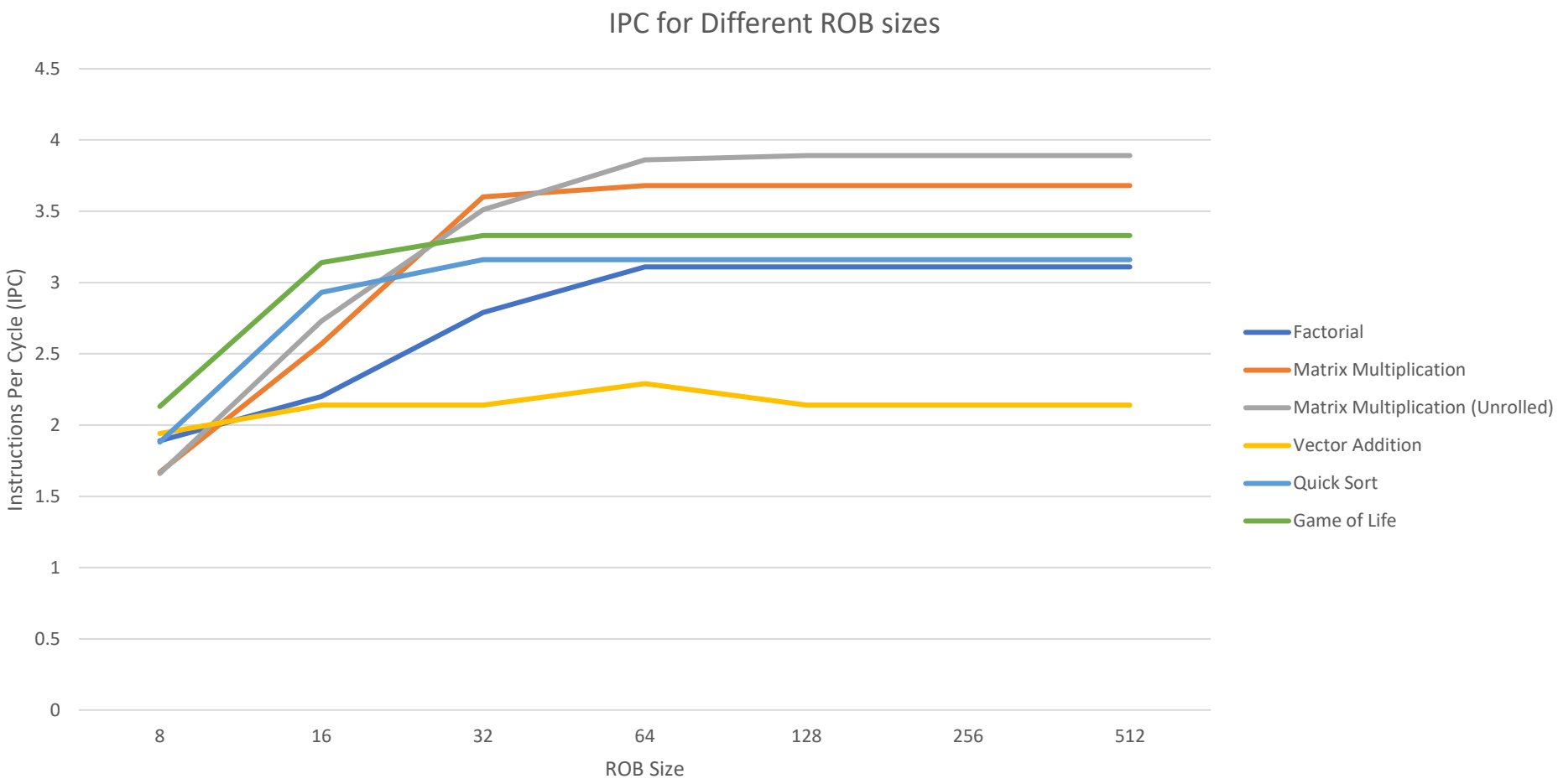
Results

Across the board we see varying results for each branch prediction type. For the first 4 benchmarks, all predictors a yield similar number of flushes which is most likely down to the nature of the programs themselves - mainly having unconditional and backwards jumping branches. This means that all our predictor types will predict taken, which more often than not is correct.

Our Game of Life benchmark is the only one to yield our hypothesised result, probably due to using mostly conditional branches, and being very variable whether they are taken or not each time they are encountered, making it better suited to Dynamic Branch Prediction.

Whilst the Quick Sort static result may seem odd at first, the assembly code includes many conditional forward jumping branches that are often taken inside the loop logic. As our static branch predictor predicts NOT taken for all forward jumping conditional branches, we can justify this huge increase in flushes. A re-ordering of the assembly code could be done in order to reduce the number of flushes needed when using the static branch predictor.

Experiment #3 – Size of Re-Order Buffer (ROB)



Hypothesis

An increased ROB size will increase the IPC and reduce the number of stalls and cycles.

Experiment

For a 4-way superscalar processor with 2-bit dynamic branch prediction, the ROB size will be increased from 8 to 512 in logical steps for all benchmark programs.

Results

As predicted, increasing the number of entries the ROB has also increases the IPC for the most part. From about 64 to 128 entries, we see the IPC level off, which tells us that the system is bottle-necked else where. This is most likely the size of the reservation stations in each pipeline, as if they are full then no instruction can be decoded and allocated a ROB position.

It could also be limited by the number of EUs inside each pipeline, however, it is unclear without testing how much performance could be gained, or how many more reservation station spaces or EUs would be needed to see a further increase in the IPC.