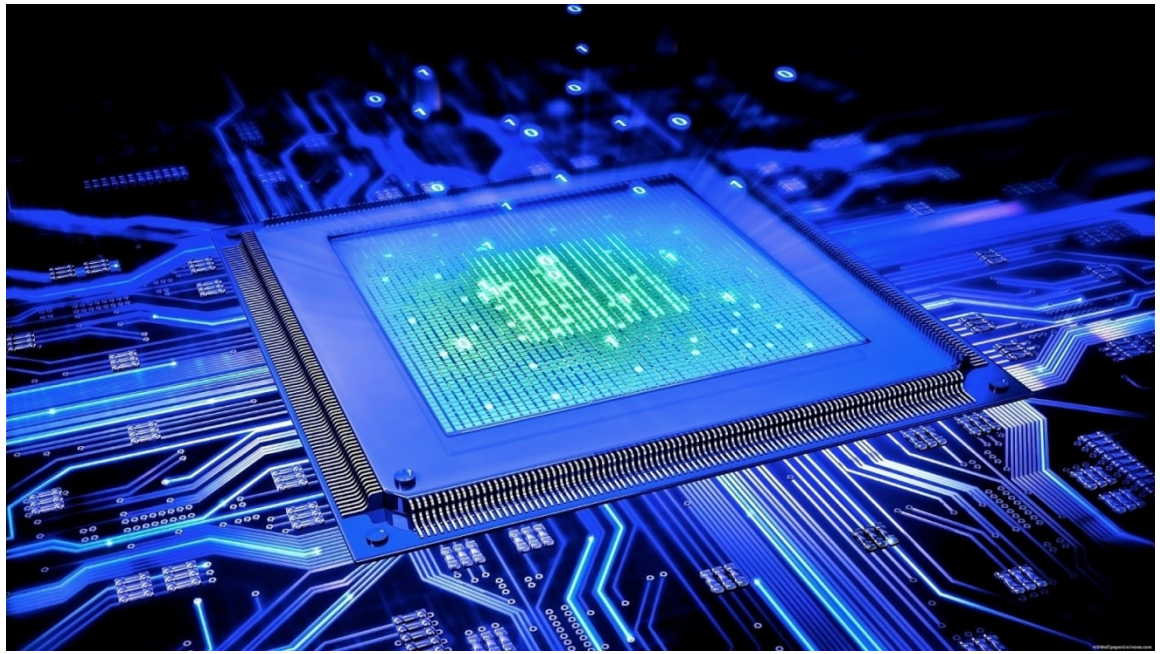
COMPUTER ARCHITECTURE

INVESTIGATION OF MEMORY DEPENDENCE PREDICTION STRATEGIES WITH SIMPLESCALAR



Lorenzo Allas

John Grun

Sanandeesh Kamat

1.0 Introduction

The Question: To Issue Load or not to Issue Load?

- Out-of-Order scheduling enhances ILP by overlapping more instructions
- Out-of-Order scheduling of Loads risks Memory Order Violations against unresolved Stores.
- Early implementations of Dynamic Scheduling (e.g. Tomasulo) issued Loads and Stores in program order to prevent Memory Order Violations.

The Answer: Memory Dependence Prediction (MDP)

- **Guess** whether or not issuing a Ready Load Out-of-Order will trigger a Memory Order Violation
- **Detect** when Memory Order Violation does occur
- **Recover** processor state
- **Update** MDP state to learn from experience

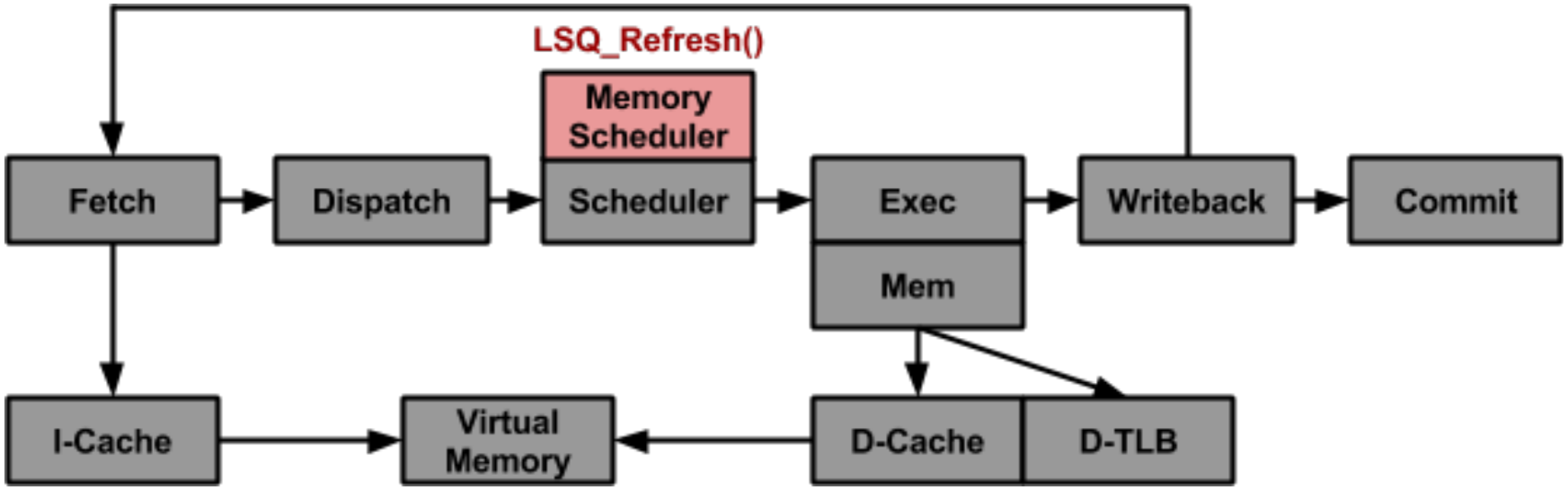
The Purpose of this Project

- Investigate the effectiveness of Store Sets and CDP as MDP schemes.
- Utilize SimpleScalar's Out-of-Order Simulator (simoutorder.c) as the simulation framework.

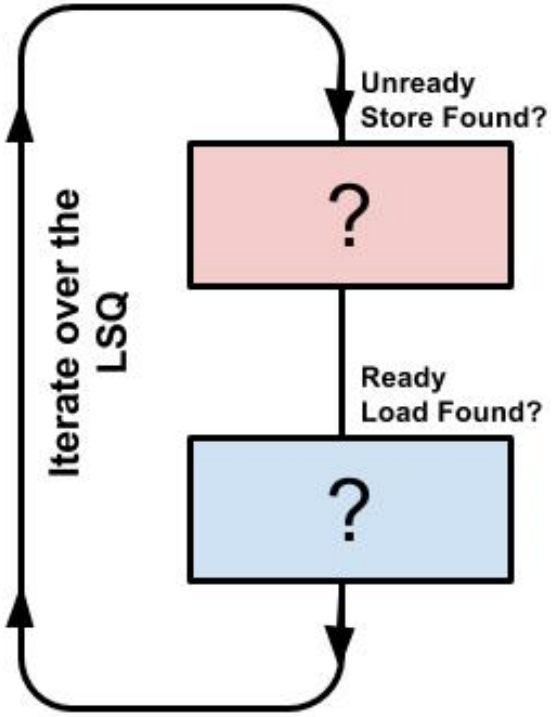
2.0 Methods & Materials

Overview of Memory Dependence Management with Load-Store-Queue (LSQ)

- Memory dependence between Loads and Stores is determined in lsq_refresh()
- Involves iterating over the elements of the LSQ as shown.
- This function served as the backbone for implementing all of the MDP schemes as well as the metric trackers.



LSQ_Refresh() Framework



2.0 Methods & Materials

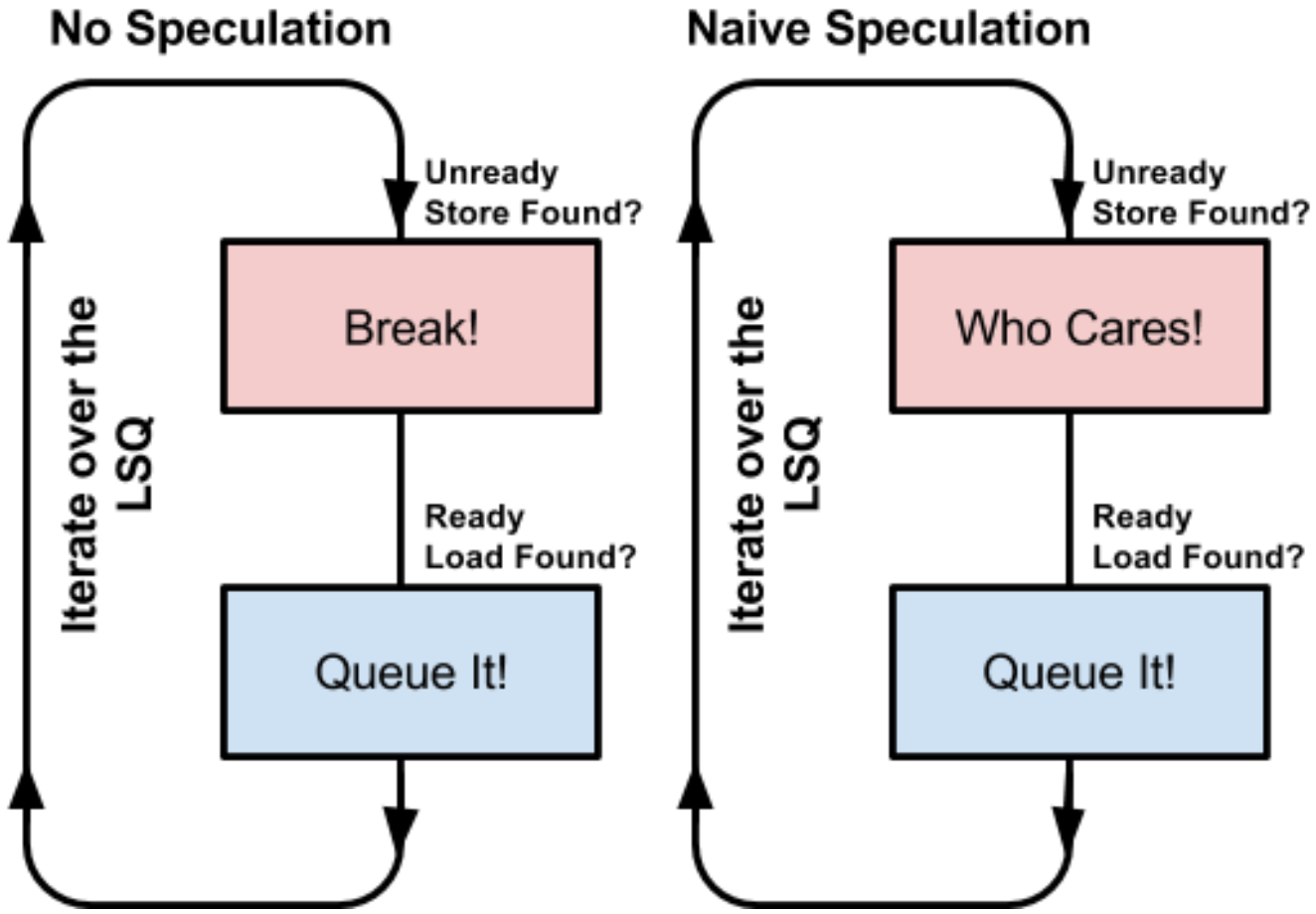
Overview of the Baseline MDP Schemes

No Speculation (Conservative)

- Queues ready Loads **only** if all prior Stores are ready
- Worst-case reference for False Dependencies

Naive Speculation (Aggressive)

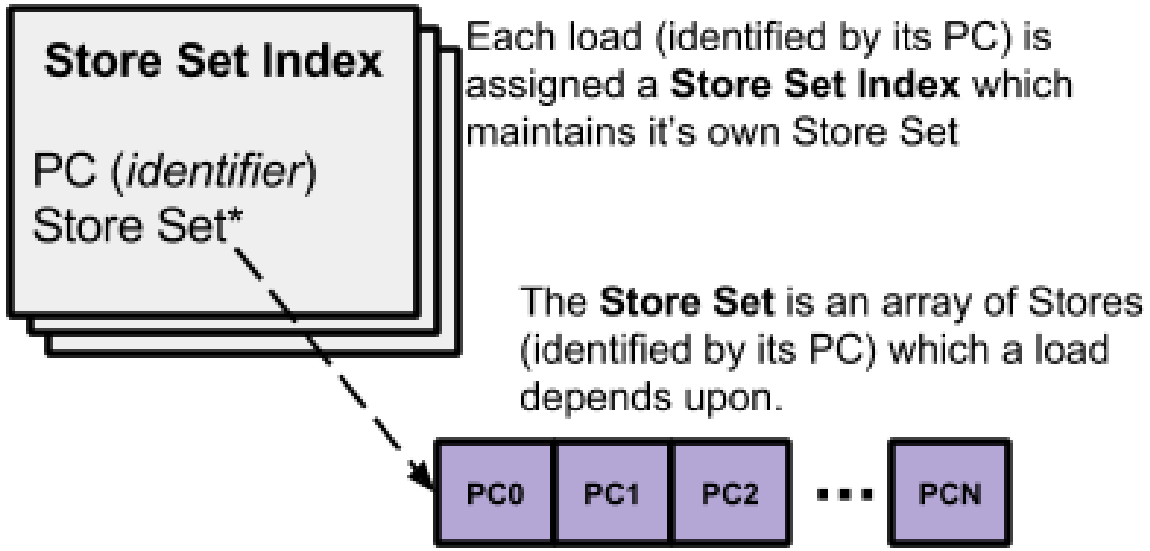
- Queues all ready Loads **regardless** of prior Stores' readiness
- Worst-Case reference for Memory Order Violations

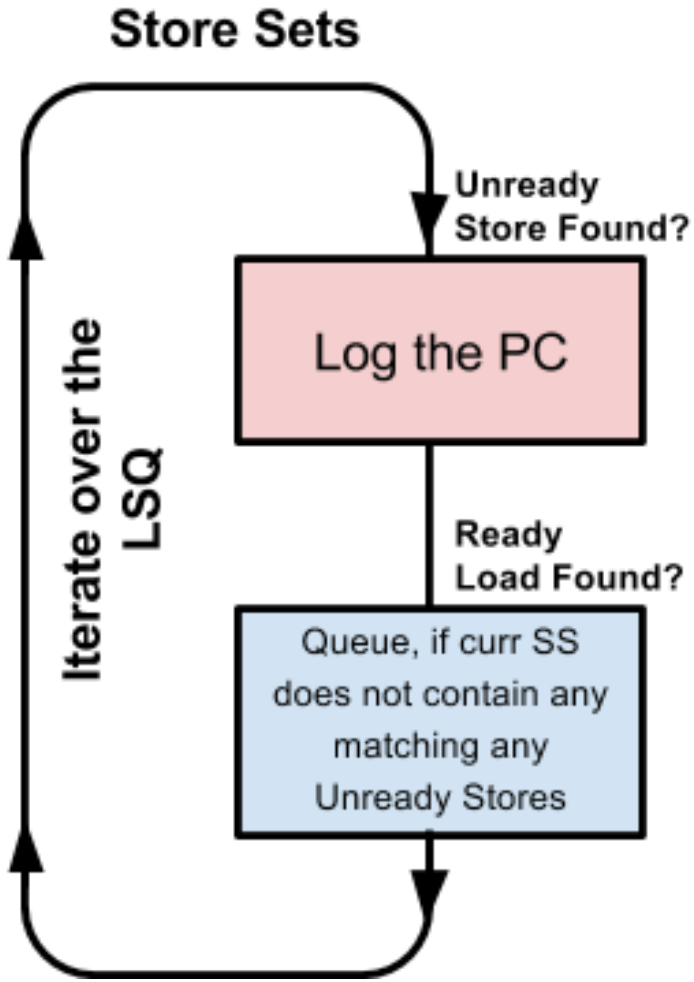


2.0 Methods & Materials

Overview of the Store Sets MDP Scheme

- The processor incrementally logs the PCs of Stores upon which Loads have historically depended to determine the soonest point in time at which a given load may issue.
- As conflicting stores which were initially unaccounted for are encountered (detected by Memory Order Violations), their PCs are added to the Store Set to improve future performance.

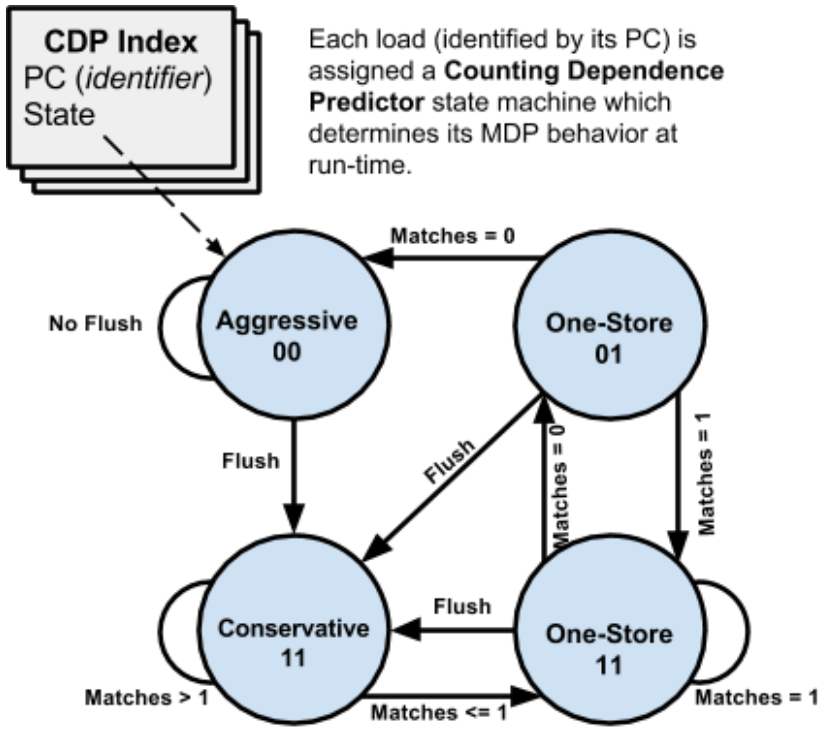


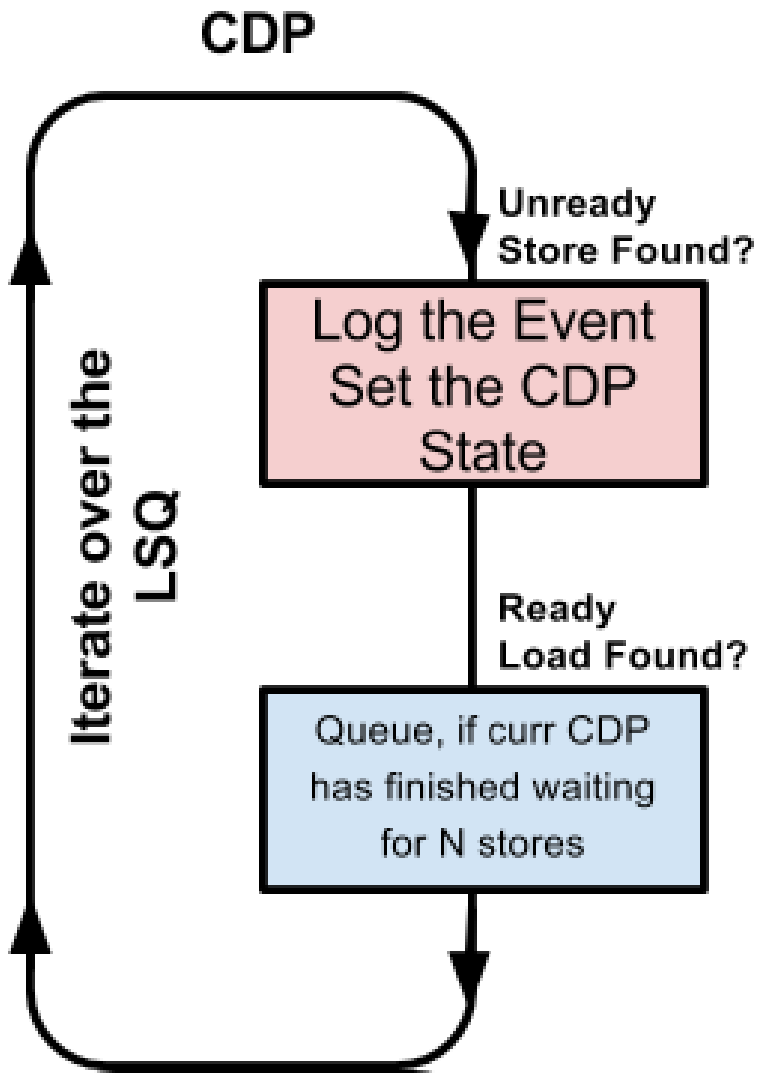


2.0 Methods & Materials

Overview of the Counting Dependence Predictor MDP Scheme

- Behavior of Load is dictated by CDP state machine.
- Predicts the *number of stores* (not specific PCs) which a load must wait for.
- Defaults to *Aggressive* if no dependencies are encountered.
- Defaults to *Conservative* if >N dependencies are encountered





2.0 Methods & Materials

Overview of the Performance Metrics

Memory Order Violation

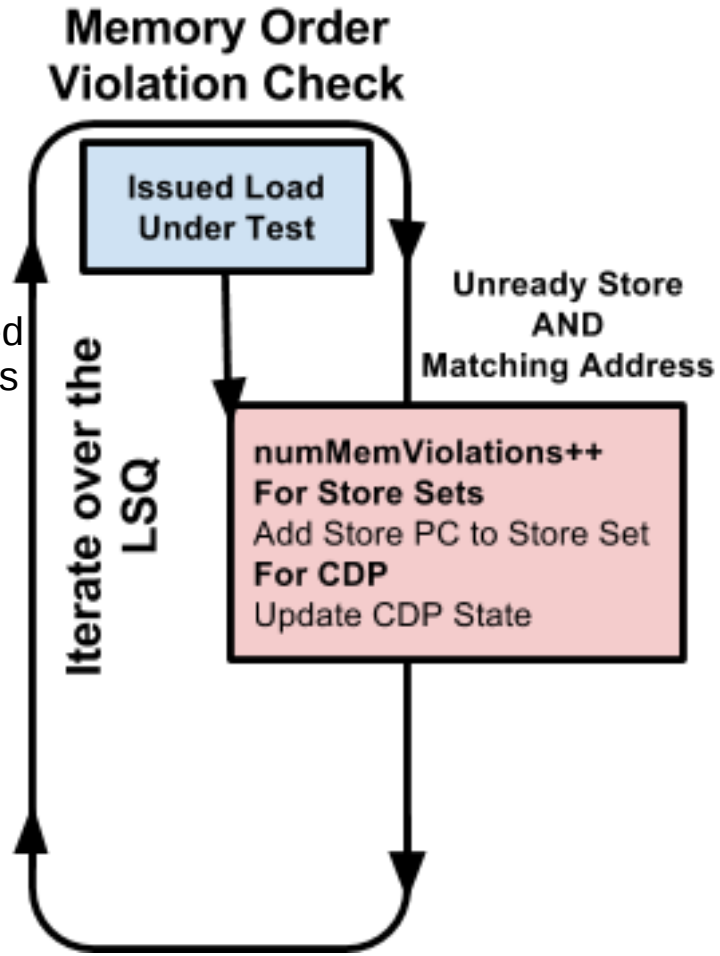
- A program error in which an out-of-order load executes before prior store with a matching effective address.

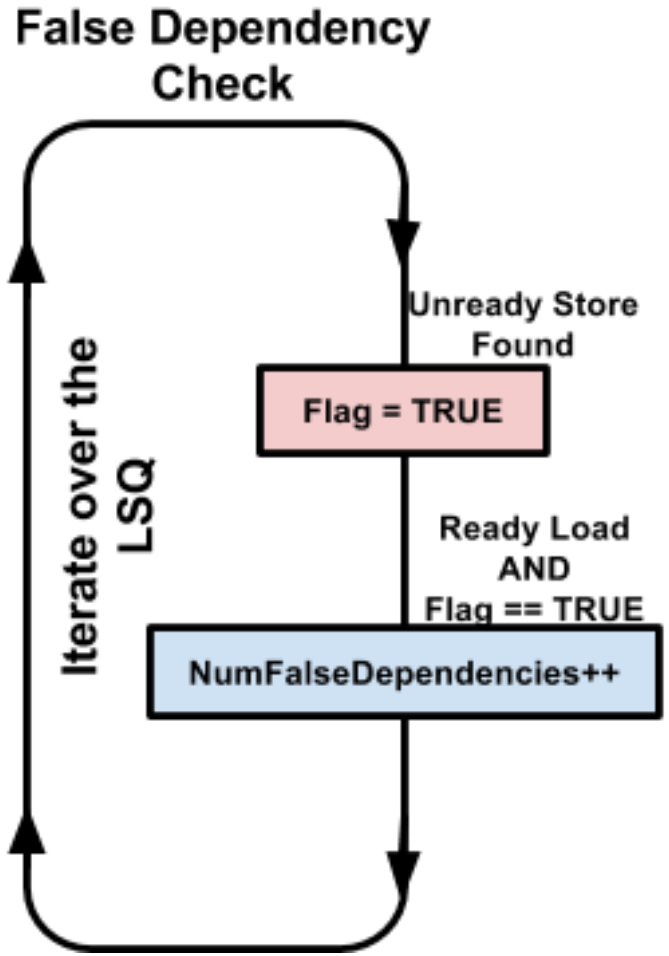
False Dependency

- Wasted clock cycles during which a ready load is stalled due to the detection of a prior unready store which does not have a matching effective address.

Instructions Per Cycle (IPC)

- The average number of instructions which are retired per cycle.





3.0 Implementation of the Experiments

- CLI arguments were added to toggle between MDP Schemes
  ./sim-outorder -ALGORITHM_TYPE * ./tests/bin/*
- Each test program was run against each MDP scheme to generate 40 simulation runs.
- The SimpleScalar parameters were fixed as shown across all simulations.
- The performance metrics were logged and analyzed to evaluate the MDP schemes

Test Programs
anagram
test-args
test-dirent
test-fmath
test-llong
test-lswlr
test-math
test-printf

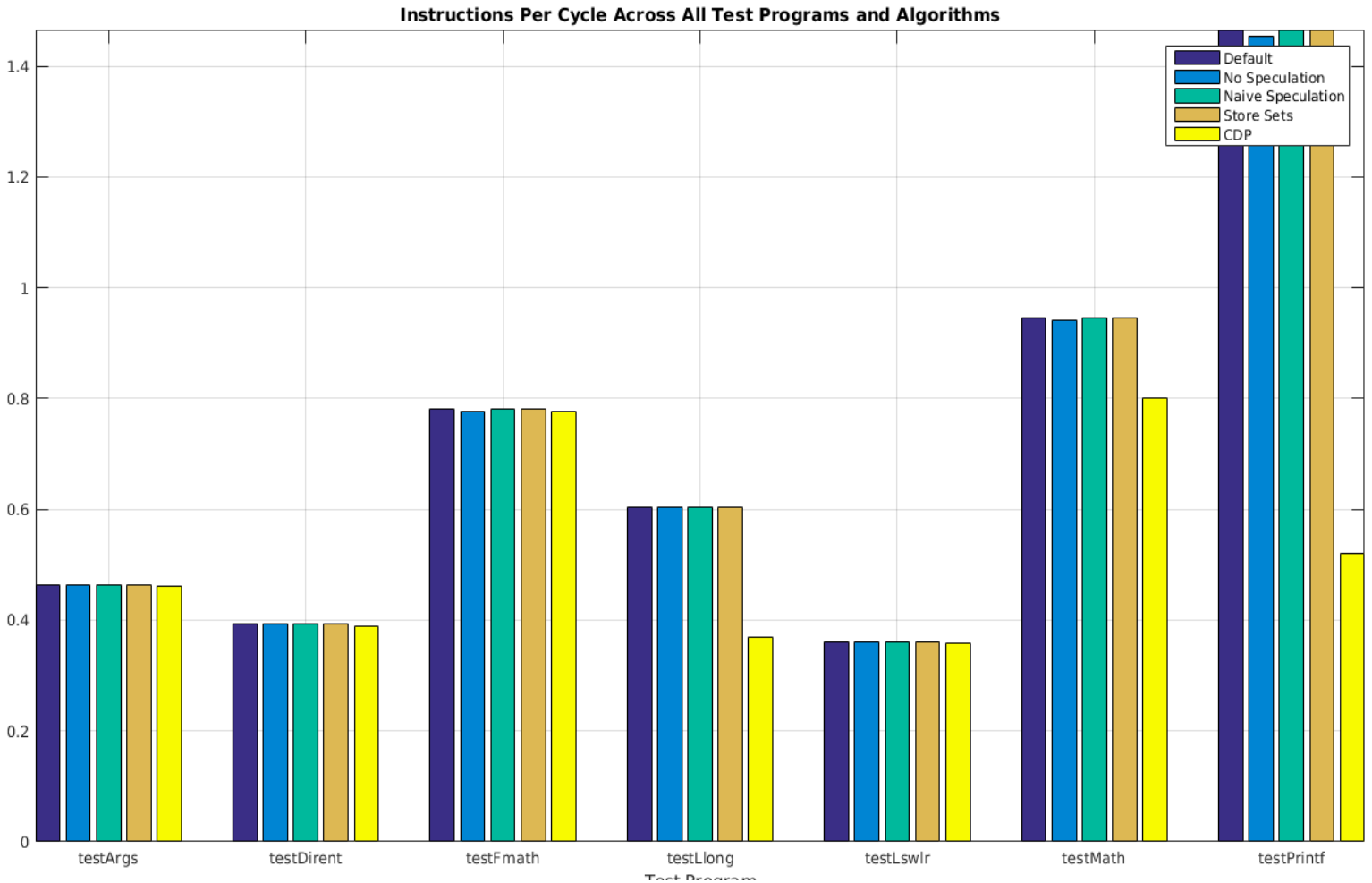
MDP Scheme	Memory Order Violations	False Dependencies	Project Function Name	CLI
Default	None	Many	*lsq_refresh()*	0
No Speculation	None	Many	*lsq_refresh_NoSpeculation()*	1
Naive Speculation	Many	None	*lsq_refresh_NaiveSpeculation()*	2
Store Sets	Few	Few	*lsq_refresh_InfStoreSets()*	3
CDP	Few	Few	*lsq_refresh_CountingDependencePredictor()*	4

SimpleScalar Parameter	Val
Instruction Fetch Queue Size (in insts)	4
Instruction Decode Width (insts/cycle)	4
Instruction Issue B/W (insts/cycle)	4
Instruction Commit B/W (insts/cycle)	4
Memory Access Bus Width (in bytes)	8
Register Update Unit Size	8

3.0 Results

Instructions Per Cycle (IPC)

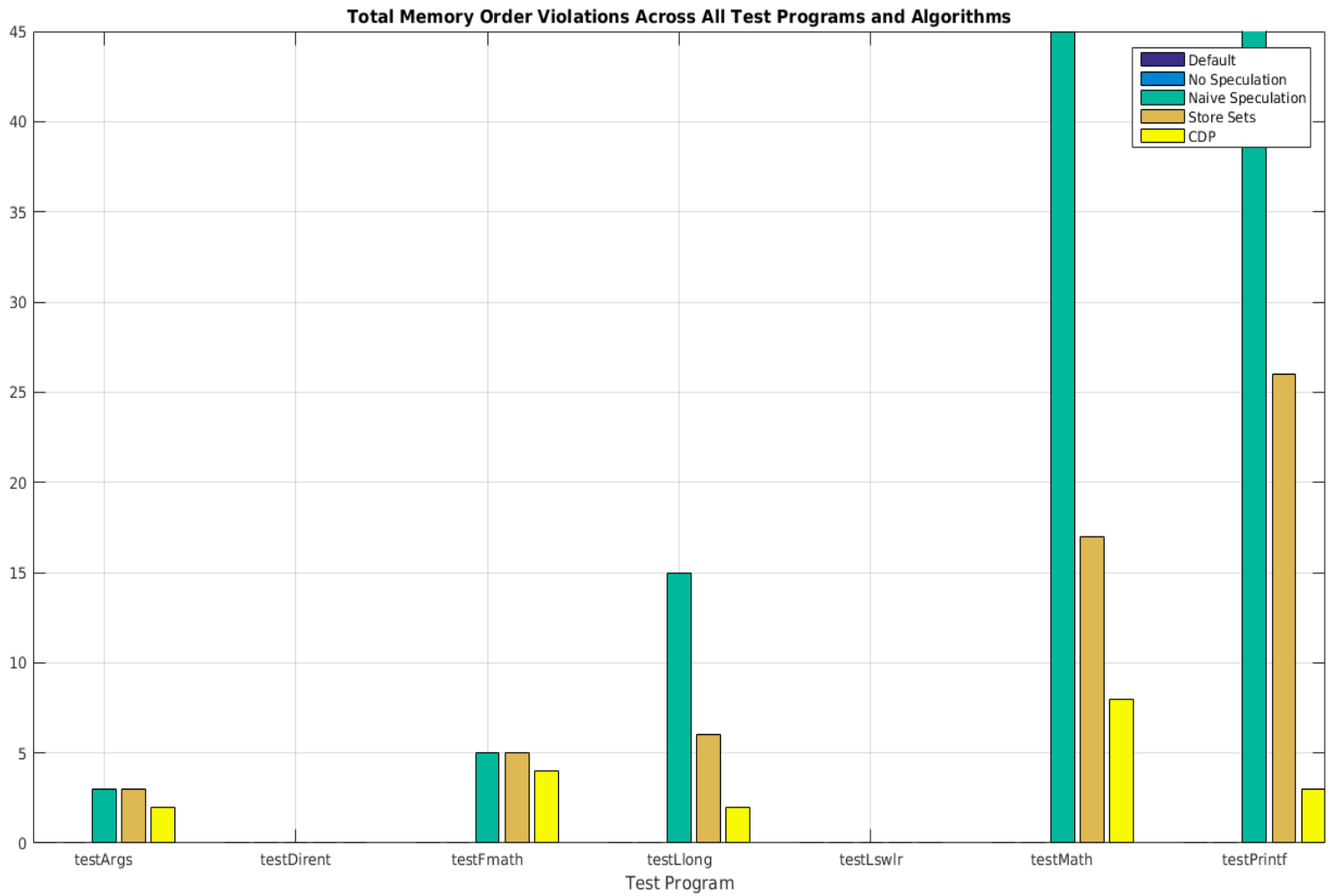
- The various MDP schemes applied to the test data did not result in significant variation in IPC.
- Consistent decrease in IPC for No Speculation which is by definition the most sluggish of all MDP schemes.
- Possible that these results would have greater variance if B/Ws were increased.



3.0 Results

Memory Order Violations

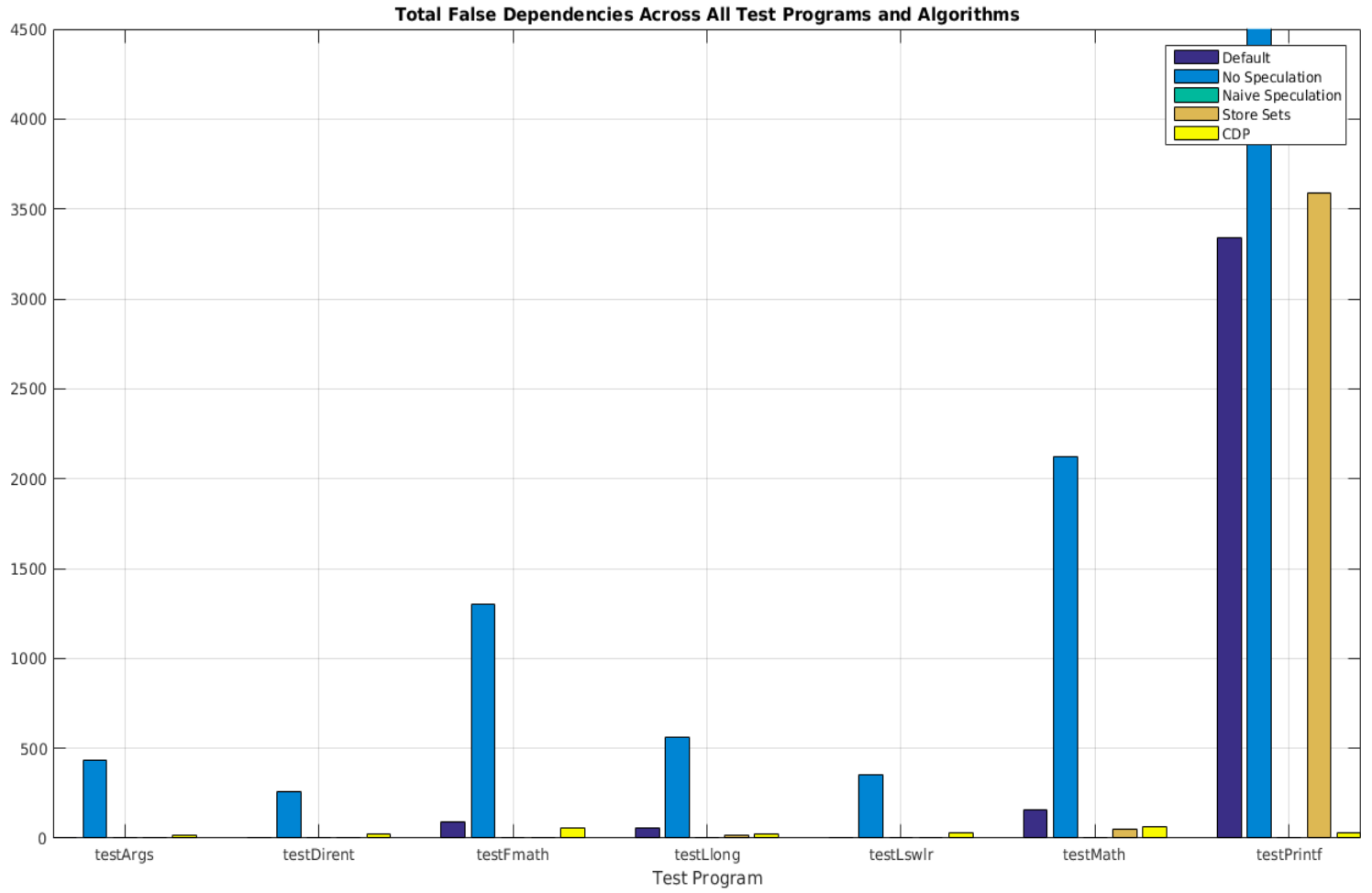
- Default, No Speculation consistently generated zero
- Naive Speculation consistently generated many.
- As expected, Store Sets and CDP consistently generated > 0 but fewer than Naïve Speculation (the worst case baseline)



3.0 Results

False Dependencies

- Naïve Speculation consistently generated zero.
- No Speculation consistently generated the most.
- It is optimistic that the Store Sets and CDP schemes resulted in fewer False Dependencies than the Default and No Speculation.
- However, as there was no significant improvement in IPC, the overall value of these experimental MDP schemes is still undemonstrated.



4.0 Discussion

- IPCs did not indicate enhanced performance of MDP schemes
- As expected, the Baseline MDPs …
  - **For Memory Order Violations**: the Default and No Speculation generated none and the Naive Speculation generated many.
  - **For False Dependencies**: the Default and No Speculation generated many and the Naive Speculation generated none.
- As expected, the Store Sets and CDPs …
  - **For Memory Order Violations**: generated >0 Memory Order Violations, which is the trigger event by which the Store Sets and CDPs are to be initialized in the first place. But fewer than Naïve Spec.
  - **For False Dependencies**: generated fewer than Default, No Speculation, and Naive Speculation.

4.0 Conclusion

- Memory Dependence Predictor (MDP) schemes are developed to permit Loads to execute out-of-order without incurring Memory Order Violations.
- This project sought to demonstrate the performance enhancement capabilities of Store Sets and Counting Dependence Predictor (CDP), within the SimpleScalar simulation framework.
- The project's developed source code toggles between four different MDP schemes:
  - No Speculation and Naive Speculation (**Baseline**) successfully demonstrated the predicted behavior of maximizing False Dependencies and Memory Order Violations, respectively.
  - The Store Sets and CDP (**Experimental**) successfully demonstrated an expected moderate incurrence of Memory Order Violations and False Dependencies which were not as high as worst-case Baseline MDPs
  - However, Store Sets and CDP did not achieve higher IPC which does not support the utility of the implementations.