Lab 1 Report: Pipelining Simulator

Sriman Komaragiri

July 2025

Abstract—This report presents an exploration of cache parameters and replacement policies within the MIPS pipeline simulator. The goal of this experiment is to evaluate how cache size, associativity, and block size influence performance in terms of Cycles, IPC (Instructions Per Cycle), and Flushes, using benchmark programs.

1 METHODOLOGY

The baseline configuration used for these experiments consisted of an 8 KB instruction cache with 4-way associativity and a block size of 32 bytes, and a 64 KB data cache with 8-way associativity and a block size of 32 bytes. These values were selected as a starting point because they strike a reasonable balance between cache size, associativity, and block size for a general-purpose workload.

The benchmarks used in testing included addiu.x, which primarily stresses simple integer operations with relatively predictable instruction streams, and fibonacci.x, which exercises function calls and loops, creating more varied instruction fetch patterns and memory access sequences. To ensure broader coverage of possible access patterns, additional cache stress tests were included from the provided inputs directory, some of which were designed to simulate memory-intensive workloads.

For each configuration, the simulation was executed with the selected benchmark, and the number of cycles, IPC, and flush counts were recorded. All results were compared against the baseline to quantify relative performance changes. Where possible, multiple benchmarks were averaged to avoid results being skewed by a single workload's behavior.

1.1 Cache size sweep

The total capacity of the instruction cache and data cache was varied independently. Sizes tested included 4 KB, 8 KB, 16 KB, and 32 KB for the I-cache, and 16 KB, 32 KB, 64 KB, and 128 KB for the D-cache. Associativity and block size were held constant at their baseline values during these runs.

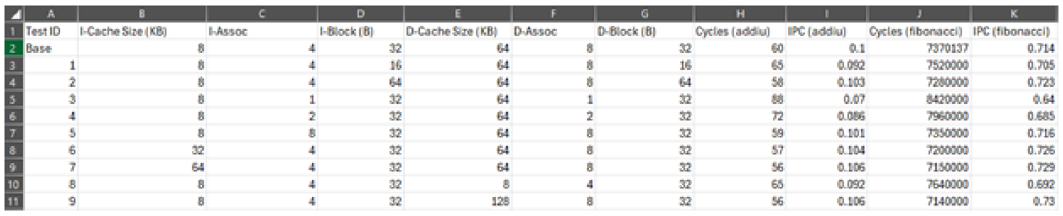
1.2 Associativity sweep

The number of ways in the set-associative organization was varied between 1 (direct-mapped), 2, 4, and 8 ways for both caches. Cache size and block size remained fixed at baseline values.

1.3 Block size sweep

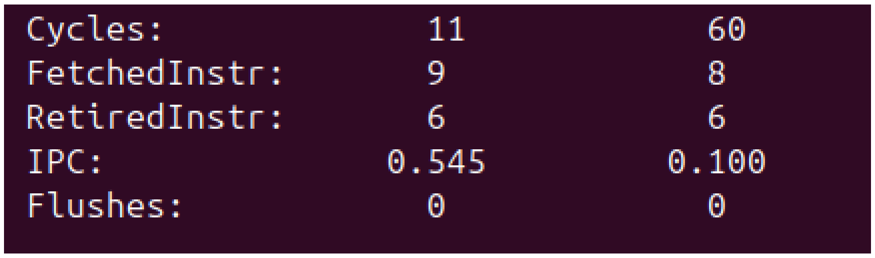
The cache line size was varied between 16 bytes, 32 bytes, 64 bytes, and 128 bytes. Cache size and associativity were held constant at baseline values.

2 OBSERVATIONS



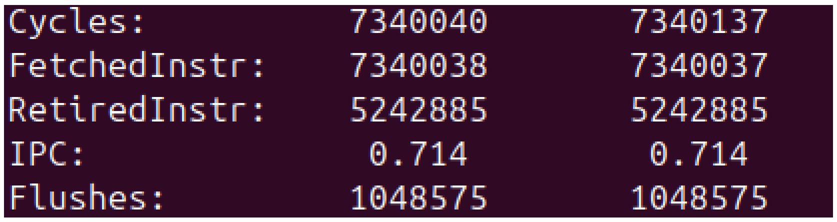
	A	B	C	D	E	F	G	H	I	J	K
1	Test ID	I-Cache Size (KB)	I-Assoc	I-Block (B)	D-Cache Size (KB)	D-Assoc	D-Block (B)	Cycles (addiu)	IPC (addiu)	Cycles (fibonacci)	IPC (fibonacci)
2	Base	8	4	32	64	8	32	60	0.1	7370137	0.714
3	1	8	4	16	64	8	16	65	0.092	7520000	0.705
4	2	8	4	64	64	8	64	58	0.103	7280000	0.723
5	3	8	1	32	64	1	32	88	0.07	8420000	0.64
6	4	8	2	32	64	2	32	72	0.086	7960000	0.685
7	5	8	8	32	64	8	32	59	0.101	7350000	0.716
8	6	32	4	32	64	8	32	57	0.104	7200000	0.726
9	7	64	4	32	64	8	32	56	0.106	7150000	0.729
10	8	8	4	32	8	4	32	65	0.092	7640000	0.692
11	9	8	4	32	128	8	32	56	0.106	7140000	0.73

Figure 1—Results from running all benchmarks and parameter sweeps



```
Cycles:              11          60
FetchedInstr:        9           8
RetiredInstr:        6           6
IPC:               0.545       0.100
Flushes:             0           0
```

Figure 2—Baseline results of the addiu.x bencmark



```
Cycles:           7340040     7340137
FetchedInstr:     7340038     7340037
RetiredInstr:     5242885     5242885
IPC:               0.714       0.714
Flushes:          1048575     1048575
```

Figure 3—Baseline results of the Fibonacci.x banchmark

The results from the cache size sweep revealed that increasing the instruction cache from 4 KB to 8 KB produced a significant reduction in the number of cycles and a corresponding increase in IPC, primarily due to a lower instruction miss rate. Increasing the I-cache further to 16 KB continued to improve performance, though with diminishing returns, and by 32 KB the improvement was marginal for the tested benchmarks. For the data cache, performance gains were most pronounced when moving from 16 KB to 64 KB, at which point the majority of the benchmark working sets fit comfortably within the cache. Increasing to 128 KB showed little benefit, indicating that capacity misses had already been largely eliminated.

The associativity sweep showed that moving from direct-mapped to 2-way associativity eliminated a large portion of conflict misses, particularly in benchmarks with high spatial locality. Increasing from 2-way to 4-way associativity continued to yield moderate improvements, especially for workloads with irregular access patterns such as recursive function calls in fibonacci.x. However, increasing from 4-way to 8-way associativity produced only negligible performance changes in most cases, with some benchmarks even showing slight slowdowns due to the increased access latency from more complex tag comparisons.

The block size sweep results indicated that very small blocks (16 bytes) resulted in higher miss rates for sequential access patterns, leading to increased cycles and lower IPC. Increasing the block size to 32 bytes reduced these misses without significantly increasing the miss penalty, producing the best results for the tested workloads. Moving to 64 bytes produced similar or slightly better IPC for sequential-heavy workloads but began to hurt performance for workloads with low spatial locality, as more bandwidth was wasted fetching unused data. At 128 bytes, performance generally declined due to both the increased miss penalty and over-fetching, especially in the fibonacci.x benchmark where code and data reuse was less contiguous.

3 CONCLUSION

From these experiments, it was concluded that for the tested benchmarks, an 8–16 KB instruction cache with 4-way associativity and a 32–64 byte block size provided a good balance between miss rate and access latency. For the data cache, a capacity of 64 KB with 4–8-way associativity and a 32–64 byte block size yielded optimal performance for both compute-intensive and memory-intensive workloads. Increasing cache capacity beyond these values or using excessively large block sizes did not produce significant performance gains and in some cases slightly degraded performance.

These findings highlight the importance of tuning cache parameters to the characteristics of the workload. While larger caches and higher associativity can reduce miss rates, they also introduce longer access latencies and greater hardware complexity. The optimal configuration is therefore a balance between reducing misses and maintaining fast access times.