CS 6354: Graduate Computer Architecture Microbenchmarking Project — Checkpoint 2

Student ID: nfz5mv

1. Overview

This report presents the results of the second checkpoint of the microbenchmarking project, focusing on pipeline-level performance measurements on an Apple M2 Pro processor. The following experiments were implemented and executed successfully:

- 02_fetch_throughput — Instruction Fetch Throughput
- 03_retire_throughput — Effective Instruction Throughput
- 04_load_store_throughput — Load/Store Bandwidth
- 05_branch_penalty — Branch Misprediction Penalty
- 06_exec_unit_throughput — Integer Execution Unit Bandwidth

Benchmarks 00 and 01 (Function Call and Context Switch) were previously completed and verified, but are not the focus of this checkpoint.

2. Environment

- **Hardware:** Apple M2 Pro (ARMv8.6-A, 8 performance + 4 efficiency cores)
- **Operating System:** macOS Sequoia 15.3.1
- **Compiler:** Apple Clang 17.0.0 (clang-1700.3.3.0)
- **Standard:** C11
- **Build:** `make clean && make`
- **Execution:** `./scripts/run_all.sh`

3. Methodology Summary

Each microbenchmark isolates a specific pipeline component using tight loops and high-precision timing through the macOS monotonic clock (`mach_absolute_time`), wrapped in `harness.c` ,which provides similar nanosecond-level accuracy comparable to `rdtsc`-based counters on x86 systems. The following subsections describe the methodology for each test from 02–06.

3.1 Instruction Fetch Throughput (02)

This benchmark measures the number of instructions fetched per cycle by executing a tight unrolled loop of independent `NOP`-like operations. The elapsed time per iteration is converted to cycles using the nominal CPU frequency (3.2 GHz).

Result:

Average time per instruction: 0.555 ns
$\approx$ 1.776 cycles/instruction
$\approx$ **0.563 instructions per cycle**

Analysis: The result shows a fetch width of approximately 0.56 instructions per cycle, which is consistent with front-end bottlenecks such as instruction cache and branch prefetch queues on ARM cores under sequential fetch conditions.

3.2 Effective Instruction Throughput (03)

This test evaluates instruction-level parallelism (ILP) by executing multiple independent ALU operations per iteration, which varies the number of independent chains (1–8).

Table 1: Effective Instruction Throughput Results (ILP Scaling)

ILP	1	2	3	4	5	6	7	8
IPC	0.528	1.210	1.351	1.324	1.593	1.751	1.751	1.783

Summary: Min IPC = 0.528, Max IPC = 1.783, Median IPC = 1.472

Analysis: Throughput increases with ILP until saturation at $\approx$1.8 IPC, which matches Apple's 3-wide out-of-order execution design, indicating the backend can exploit parallelism but is limited by dispatch width and issue queue size.

3.3 Load/Store Throughput (04)

This benchmark alternates between consecutive load and store operations in a memory-resident array to measure memory pipeline bandwidth under full cache residency.

Result:

Load throughput: 0.231 loads/cycle
Store throughput: 2.182 stores/cycle

Analysis: Store throughput is noticeably higher than load throughput, which can be attributed to write-combining and the use of non-blocking store buffers. In contrast, the lower load rate is mainly limited by cache dependencies and load-to-use latency within the memory pipeline.

3.4 Branch Misprediction Penalty (05)

A tight loop with pseudo-random branches induces mispredictions. By dividing total cycles by the number of branches, the average penalty per misprediction is estimated.

Result:

Estimated Branch Misprediction Penalty: **2.81 cycles**

Analysis: This indicates a shallow pipeline with roughly 8–10 stages. Compared to x86 designs (15–20 cycles), Apple's M2 achieves faster recovery from branch flushes due to shorter decode and issue pipelines.

3.5 Integer Execution Unit Bandwidth (06)

Independent integer ADD, MUL, and DIV operations are executed to estimate the maximum number of integer arithmetic instructions per cycle.

Result:

ADD throughput: 0.97 ops/cycle
MUL throughput: 0.28 ops/cycle
DIV throughput: 0.55 ops/cycle

Analysis: The nearly 1.0 ADD ops/cycle suggests a single-cycle ALU unit per performance core. Lower throughput for MUL and DIV is consistent with multi-cycle latencies and shared execution ports.

4. Discussion and Conclusions

All five microbenchmarks executed successfully on the Apple M2 Pro, and the observed results are within realistic architectural expectations for an ARM-based core. Fetch and retire bandwidth confirm a front-end bottleneck below the theoretical peak, while the branch misprediction penalty and integer ALU throughput align with the M2's balanced performance design. Minor numerical variations were observed across repeated runs due to dynamic frequency scaling and the macOS scheduler, but the median-based aggregation ensures stability of the reported values.