Computer Organization Project 4 – Superscalar Processor Simulation

Due: 23:59, June 4, 2025

A superscalar processor is capable of issuing and executing multiple instructions per clock cycle by leveraging instruction-level parallelism (ILP). This is achieved through wider datapaths, multiple execution units, and dynamic scheduling mechanisms, such as Tomasulo's algorithm. Key features include register renaming, reservation stations, in-order issue, out-of-order execution, and in-order retirement. The gem5 simulator provides a detailed out-of-order CPU model (O3CPU), which supports superscalar microarchitectural configurations with parameterized widths and buffer sizes. Konata is a visualization tool that illustrates pipeline behavior over time, helping researchers to understand how architectural decisions could impact execution performance at the instruction level.

In this project, you will use gem5 and Konata to explore the behavior of superscalar out-of-order processors. You will analyze how program characteristics and microarchitectural parameters affect instruction-level parallelism and pipeline efficiency. You are encouraged to explore the following microarchitectural parameters in gem5's O3CPU model:

- Fetch Widths: `system.cpu[0].fetchWidth`
- Issue Widths: `system.cpu[0].issueWidth`
- Commit Widths: `system.cpu[0].commitWidth`
- Reorder Buffer Size: `system.cpu[0].numROBEntries`
- Issue Queue Size: `system.cpu[0].numIQEntries`
- Number of Physical Registers: `system.cpu[0].numPhysIntRegs`

[Hint] More microarchitectural parameters can be found via the following Python code.
URL: https://github.com/gem5/gem5/blob/stable/src/cpu/o3/BaseO3CPU.py

In addition, you are encouraged to use Konata to visualize the execution trace of your test programs along with your microarchitectural parameter configurations. Identify and explain some of the following behaviors with Konata trace screenshots. Each phenomenon should include a screenshot with instruction labels and your explanation:

- In-order Fetch and Decode
- Out-of-order Issue and Execution
- In-order Commit (Retirement)
- Multiple instructions issued or committed in the same cycle
- Pipeline bubbles due to data hazards, control hazards, resource stalls, etc.
- [BONUS] Evidence of renaming eliminating write-after-write or write-after-read hazards
- [BONUS] The additional phenomena you have learned/found via Konata.

Please submit your research and experiment report along with your C/C++ test programs and microarchitectural parameter configurations according to the following rules:

1- You need to install the GNU RISC-V 64-bit cross-compiler GCC/G++ toolchain and compile your test programs with `-static` and `-O0` compilation options.
2- [Hint] Using inline assembly in your C/C++ test program can significantly help you control the specific register usage efficiently and effectively during your experiments.
3- Design at least two test programs: one with high ILP (e.g., low data dependencies) and the other with low ILP (e.g., serial data dependencies). Simulate each program under at least two microarchitectural configurations. Compare their performance and pipeline behaviors. Explain how instruction dependency interacts with microarchitecture design and what kinds of programs benefit most from superscalar processors.
4- Write your research and experiment report to discuss your simulation results and summarize your observations.
5- The filename is your student ID (e.g., B12345678.tgz and B12345678.pdf)

Reference:

[1] The gem5 Simulator
URL: https://www.gem5.org/
[2] Learning gem5
URL: https://www.gem5.org/documentation/learning_gem5/introduction/
[3] Konata Visualization Tool
URL: https://github.com/shioyadan/Konata